UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aleš Vetrih

ODPRTOKODNO OKOLJE ZA CELOVITO IZVAJANJE TESTIRANJA

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: dr. Igor Rožanc

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00439/2009
Datum: 05.04.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ALEŠ VETRIH**

Naslov: **ODPRTOKODNO OKOLJE ZA CELOVITO IZVAJANJE TESTIRANJA**
AN OPEN SOURCE ENVIRONMENT FOR INTEGRAL TESTING EXECUTION

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Testiranje je pomembna aktivnost zagotavljanja kakovosti programske opreme, ki zahteva natančno načrtovanje, izvedbo in nadzor aktivnosti. Obstaja več dragih komercialnih okolij za izvajanje testiranja, ki uporabniku pogosto ne nudijo tistega, kar zares potrebuje.

V diplomski nalogi na podlagi svojih izkušenj zasnujte in sestavite odprtokodno okolje za celovito izvajanje testiranja. V okolje sestavite sistem za upravljanje testnih primerov, sledilec napak, sistem za hranjenje znanja o testiranju in forum za izmenjavo mnenj, pri čemer izhajajte iz uveljavljenih odprtokodnih rešitev. V nalogi predstavite tudi spletno namestitev in izgled opisanega okolja.

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Mentor:

viš. pred. dr. Igor Rožanc

Dekan:

prof. dr. Franc Solina

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a _____Aleš Vetrih_____,

z vpisno številko _____63030014_____,

sem avtor/-ica diplomskega dela z naslovom:

_____Odprtokodno okolje za celovito izvajanje testiranja_____

_____

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  _____dr. Igorja Rožanca_____

  in somentorstvom (naziv, ime in priimek)

  _____

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne ___15.10.2009_____ Podpis avtorja/-ice: _____

Zahvala

Zahvaljujem se svojemu mentorju prof. dr. Igorju Rožancu za potrpežljivost in pomoč pri izdelavi diplomske naloge. Hvala vsem, ki so verjeli vame in me spodbujali ter mi pomagali.

Še posebej bi se zahvalil svoji mami Nadi, bratu Mihi, svojemu dekletu Marjeti ter sosedu Juretu. Hvala za vse.

Moji mami.

Kazalo

Seznam kratic:

SDLC (ang. System Development Life Cycle) - Življenski cikel razvoja programske opreme

CMS (ang. Content Management System) – Sistem za upravljanje vsebin

WWW (ang. World Wide Web) – Svetovni splet

HTML (ang. Hyper Test Markup Language) – Označevalni jezik za izdelavo spletnih strani

XAMPP (ang. X(cross-platform)/Apache/MySQL/PHP/Perl)

GPL (ang. General Public License) – licenca prostega programja

PHP (ang. PHP: Hypertext Preprocessor) – skriptni programski jezik PHP

CSS (ang. Cascading Style Sheets) - Kaskadne Slogovne Predloge

V&V (ang. Verification and Validation) – Verifikacija in Validacija

WYSIWYG (ang. What You See Is What You Get) – Kar vidis to dobiš

SQL (ang. Structured Query Language) - Sestavljeni strukturni jezik

IEEE SCSE (ang. IEEE Standards Collection for Software Engineering) – zbirka standardov programskega inženirstva IEEE

IEEE SGSET (ang. IEEE Standard Glossary of Software Engineering Terminology) – slovar terminologije standardov programskega inženirstva IEEE

Povzetek

Pri zagotavljanju kakovosti programske opreme igra pomembno vlogo aktivnost testiranja, ki je pomembna aktivnost v vsakem življenjskem ciklu programske opreme. Pomembno vlogo pri zagotavljanju kakovosti pa igra tudi okolje, v katerem izvajamo testiranje. Na tržišču je na voljo več komercialnih okolij, ki vsebujejo velik nabor uporabnih orodij za lažje delo pri načrtovanju, izvedbi in nadzoru aktivnosti testiranja, vendar to še ne zagotavlja dobrega prilagajanja našim potrebam.

V diplomski nalogi smo na podlagi izkušenj opisali sestavo odprtokodnega okolja za celovito izvajanje testiranja, ki združuje izbran nabor uveljavljenih samostojnih orodij. Okolje temelji na spletnih aplikacijah, zato je edino zahtevano delovno orodje na računalniku spletni brskalnik.

V prvem delu diplome smo bralcu predstavili aktivnost testiranja v posameznih fazah razvoja programske opreme. Zatem sledi kratek pregled nekaterih razvojnih modelov, metod testiranja, nivojev testiranja, uveljavljenih terminov v testiranju ter dokumentacije, ki se uporablja pri aktivnosti testiranja v stroki danes. S tem želimo prikazati razsežnost področja testiranja programske opreme ter posredno nakazati načine, kako z uporabo učinkovitega delovnega orodja pripomoremo k uspešnemu izvajanju testiranja programske opreme.

V drugem delu prikazujemo postopek namestitve posameznih orodij ter potrebnih sprememb, s katerimi omogočimo sodelovanje sicer samostojnih orodij v združeno okolje. Prikazani so postopki združevanja med sistemom Joomla! ter sledilcem napak MantisBT, orodjem za upravljanje s testnimi primeri TestLink, orodjem za hrambo znanj MediaWiki in forumom za diskusije.

V nadaljevanju prikažemo še končno podobo okolja, njegov namen, prednosti in slabost. V sklepnem delu pa poleg zaključnih ugotovitev še nekaj možnosti za nadaljnje razširitve.

Ključne besede:

- testiranje
- okolje za izvajanje testiranja
- joomla
- mantisBT
- testLink
- mediaWiki
- kunena

Abstract

When assuring software quality, the process of testing plays an important role. A large part of that is the choice of testing environment in which the process is executed. There are several commercial tools on the market that provide a large array of useful tools for planning, executing and monitoring the testing process. This, however, does not necessarily guarantee a good adaptation to our needs.

In this thesis we describe a bundle of an open-source tools for holistic test execution, consisting of a selection of established standalone tools. The test environment is entirely web-based, so the only requirement on the computer is the web browser.

In the first part we present the role of testing in various phases of software development process. Follows a short summary of some development models, testing methods, levels of testing, established terms, and documentation in use these days. With this we aim to show the dimensions of software testing and indirectly suggest ways to improve the testing process by choosing the right tool.

In the second part we presented the process of installation of individual tools and the implementation of necessary changes, which enable cooperation of otherwise independent tools to the merged environment. Processes explained are the following: integration processes between the Joomla! system and bug tracker MantisBT, TestLink tool for a test cases management, MediaWiki tool for a information storage and a discussion forum.
Hereafter, a final image of the environment is presented, as well as its purpose, strengths and weakness. At the end, apart from the final conclusions we note certain possibilities for further extensions.

In conclusion, the present conclusions, and outlines some options for further extensions.

Keywords:

- testing
- test execution environment
- joomla
- mantisBT
- testLink
- mediaWiki
- kunena

1. Uvod

Pomembno vlogo pri zagotavljanju kakovosti pri razvoju programske opreme predstavlja aktivnost testiranja. Pri tej aktivnosti lahko stroški razvoja znatno narastejo v primeru, ko se napake ne odkrijejo dovolj zgodaj. Zato je pomembno, da se pri aktivnosti testiranja uporabi celovito okolje za izvajanje testiranja, s katerim imamo pregled nad celotno aktivnostjo, dosežki, napredki in podobno.

Diplomska naloga je razdeljena na dva dela. V prvem delu želimo prikazati razsežnost aktivnosti testiranja ter nabor opravil, ki se izvajajo v njem. Zato je pomembno, da uporabljamo pri tem čim bolj prilagodljivo programsko opremo, ki jo lahko oblikujemo po svojih potrebah, kar je lahko drago, če uporabljamo komercialne programske resitve.

Najprej bomo v teoretičnem delu predstavili obseg in zahtevnost področja testiranja, ki se izvaja v vsaki fazi pri razvoju programske opreme. Pri tem bomo predstavili nekaj najbolj pogostih razvojnih modelov, metod in termine, ki so v rabi pri testiranju programske opreme.

V drugem delu bomo predstavili in opisali postopek namestitve okolja za izvajanje testiranja. V bistvu gre za združitev odprtokodnih rešitev v neko smiselno celoto z namenom postavitve cenovno ugodnega okolja. To tudi pripomore k lažjemu pregledu celotne aktivnosti testiranja, saj omogoča upravljanje s testnimi primeri, testnimi načrti, sledenje mejnikom, upravljanje z dokumentacijo, dodeljevanje opravil in odgovornosti članom testne skupine, hrambo pridobljenega znanja in podobno. Ker je celotna rešitev spletna aplikacija, je neodvisna od platforme, na kateri uporabnik izvaja svoje delo, in deluje tako na Windows, Linux ali Mac operacijskih sistemih. To seveda olajša delo pri vpeljevi novih članov, saj za ta vidik usposabljanja ne potrebujejo skoraj nobenega prilagojevalnega obdobja.

Konkretneje bo prikazan postopek združitve dveh samostojnih rešitev, sledilca napak ter sistema za upravljanje s testnimi primeri in načrti. Predstavili bomo tudi možno izvedbo centralnega dela našega sistema z uporabo sistema CMS, Joomla, ki služi za povezovanje vseh orodij, ter predlagali nadaljnje razširitve, ki bi bile možne za večjo prilagoditev posamezni organizaciji.

Cilji diplomske naloge:

- predstavitev področja aktivnosti testiranja programske opreme,
- sestaviti odprtokodno okolje za celovito izvajanje testiranja in
- kratka predstavitev podobe okolja.

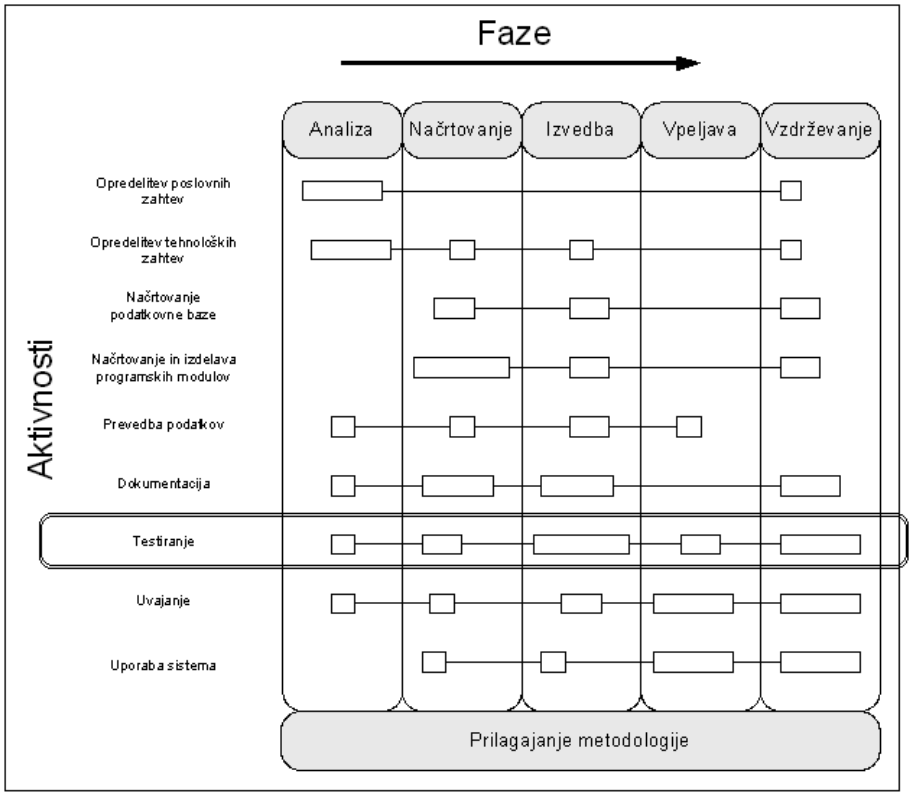
2. Testiranje programske opreme

Pogosto, ko poteka pogovor o testiranju programske opreme, naletimo na ljudi, ki imajo napačno mišljenje o sami razsežnosti, ki jo aktivnosti testiranja pokrivajo pri razvoju programske opreme. Mnogi zmotno mislijo, da za izvajanje testiranja ne obstajajo nobene posebne metode ter pristopi k testiranju. Preden se poglobimo v opis posameznih pristopov, ki jih lahko uporabimo pri testiranju, bomo najprej opravili splošen pregled skozi življenjski cikel programske opreme. S tem želimo bralcu postopno prikazali razsežnost aktivnosti testiranja, ki se izvaja pri razvoju programske opreme. Šele potem bomo podrobneje predstavili metode ter nivoje testiranja.

2.1 Uvrstitev aktivnosti testiranja v življenjski cikel programske opreme

Testiranje igra pomembno vlogo v življenjskem ciklu programske opreme (SDLC[1]). Testiranje naj bi se izvajalo na sistematičen način, kar pripomore k temu, da bi odkrili čimveč okvar v posameznih fazah razvoja programske opreme. Pri tem sledimo določenemu življenjskemu ciklu, oziroma razvojnemu modelu, ki določa zaporedje faz razvoja. Znotraj vsake faze se nahaja aktivnost testiranja (slika 1).

2.2 Faze in aktivnosti pri razvoju programske opreme



Slika 1: Aktivnost testiranja v fazah razvoja programske opreme.

[1] SDLC - System Development Life Cycle

Razvoj programske opreme je proces, ki se sestoji iz več faz in aktivnosti (slika 1). Vsaka faza se sestoji iz določenih aktivnosti, v katerih je potrebno opraviti določena opravila. Ta opravila lahko potekajo vzporedno ali zaporedno, kar je odvisno predvsem od izdelkov, ki v njih nastanejo. Vsaka aktivnost ali opravilo na vhodu prejme kot vir določene izdelke, na izhodu pa izdela nek nov izdelek.

Aktivnost testiranja najdemo znotraj vsake faze razvojnega cikla programske opreme (slika 1). Faze razvoja, ter opis obsega aktivnosti testiranja so [2]:

- **Analiza zahtev programske opreme.** Cilj te faze je, da popolnoma razumemo naloge ter programske zahteve. Pri tem je potrebno vse zahteve, ugotovitve in opredelitve skrbno dokumentirati, predvsem pa se o njih posvetovati z naročnikom. Pri tej fazi si odgovarjamo na vprašanje tipa: KAJ? Z vidika testiranja se v tej fazi določi strategija testiranja.

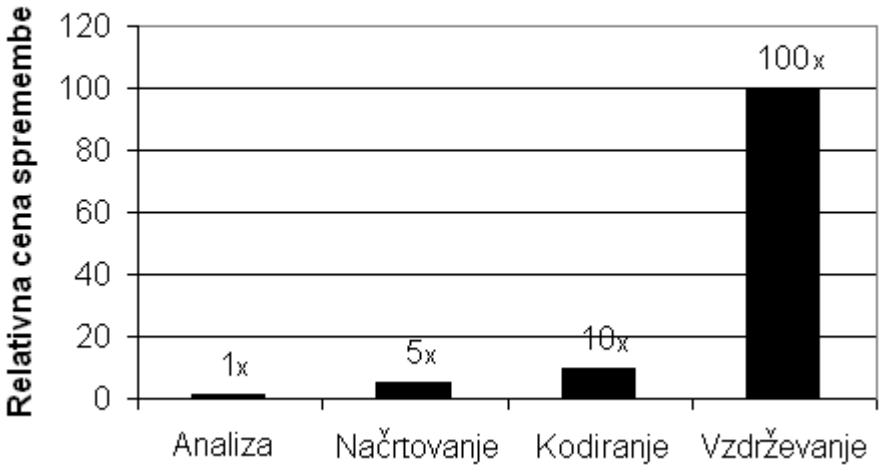
- **Načrtovanje.** V tej fazi je cilj izdelati načrt programske opreme na podlagi modelov ter specifikacij, ki smo jih pridobili v fazi analize, upoštevajoč tudi tehnološke omejitve sistema. Vprašanje te faze je: KAKO? Za aktivnosti testiranja se v tej fazi izdelata model testiranja in načrt testiranja.

- **Izvedba ali kodiranje.** Cilj te faze je izdelati kakovostno ter preverjeno programsko opremo glede na načrt, ki je bil izdelan v fazi načrtovanja. Izdela se tudi dokumentacija programske opreme. Aktivnosti testiranja obsegajo izvedbo testiranja na različnih nivojih ter pripravo testnega okolja.

- **Vpeljava.** Cilj te faze je vpeljati aplikacijo v okolje, za potrebe katerega je bila razvita, z drugimi besedami jo spraviti v produkcijo. Opravila znotraj aktivnosti testiranja obsegajo pripravo okolja za potrditveni test ter samo izvedbo potrditvenega testa.

- **Vzdrževanje.** Po končanem razvoju spremljamo delovanje programske opreme ter po potrebi odpravljamo odkrite napake, izvajamo nadgradnje in planiramo dodelave. Pri aktivnosti testiranja se v sklopu faze vzdrževanja opravijo testiranja na različnih nivojih, pripravijo novi testni scenariji posodobitve, izdelajo plani vzdrževalnega testiranja, izvajanja ponovnega potrditvenega testa novih funkcionalnosti in podobno.

Očitno je, da testiranje ni le izolirana aktivnost, pri kateri naj bi se samo preverjali rezultati kodiranja. Testiranje je potrebno izvajati v celotnem času razvoja programske opreme. Rezultate vsake aktivnosti, oziroma faze, je potrebno preveriti, saj zgodnje odkrivanje napak v razvoju pomeni lažje in cenejše odpravljanje le-teh (slika 2).



Slika 2: Naraščanje cene sprememb od faze analize do faze vzdrževanja.

Postopek testiranja po nivojih (ang. level testing) se začne s testiranjem programske kode. Začnemo s testiranjem vsakega elementa ali modula programske kode posebej. Ko postopek zaključimo, nadaljujemo na višjem nivoju po opravljenem združevanju modulov, združitvi (ang. integration). V tem delu se postopek testiranja izvaja postopno, po vsakem dodanem modulu pa znova testiramo. Po končani združitvi sledi še vrsta sistemskih testov, nazadnje pa še potrditveni test, ki se izvede v fazi vpeljave.

Pomembna termina, ki ju srečujemo pri aktivnosti testiranja čez celotni razvojni cikel, sta verifikacija (ang. verification) ter validacija (ang. validation) – V&V[2]. Z verifikacijo preverimo pravilnost rezultatov posamezne faze v razvoju, torej če program dela tisto, kar je bilo določeno z zahtevami in načrtom. Z validacijo po drugi strani preverimo, če rezultati posameznih faz res rešujejo problem, ki smo si ga postavili na začetku razvoja.

V praksi se kot zelo dober pristop pri postopku V&V izkaže tako imenovana *neodvisna verifikacija in validacija programske opreme*. To pomeni, da verifikacijo in validacijo opravi zunanji izvajalec (ang. outsider), oziroma nekdo, ki ni bil neposredno vključen v načrtovanje ali programiranje programske opreme [1]. Najem zunanjega izvajalca ali kar organizacije zna sicer predstavljati velik strošek, ki pa načeloma vodi h kakovostnejšemu končnem izdelku.

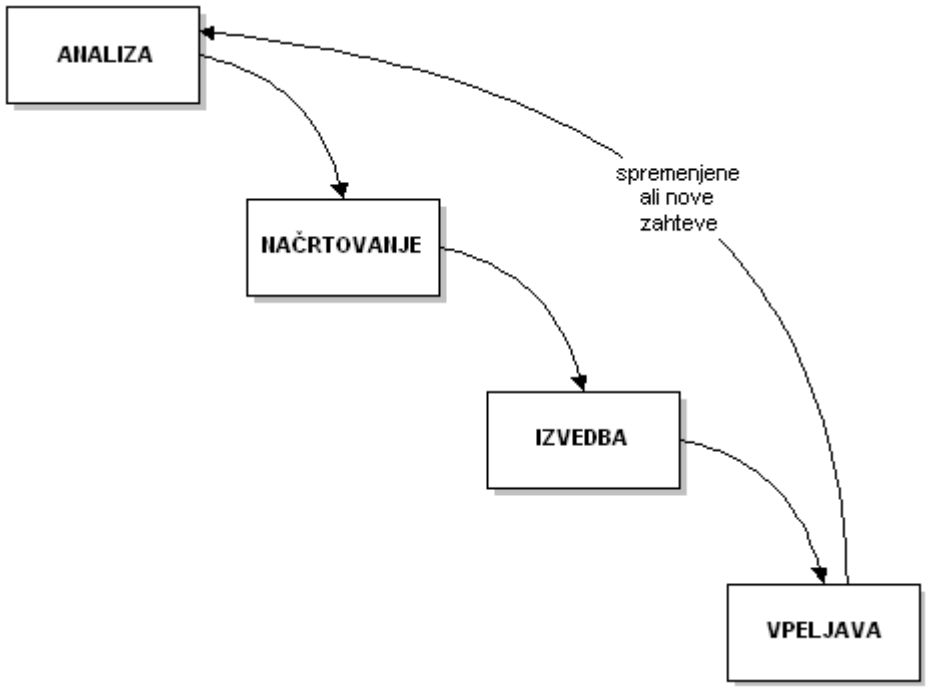
2.3 Razvojni modeli programske opreme

Razvoj programske opreme sledi določenemu življenjskemu ciklu programske opreme, oziroma razvojnemu modelu, ki določa zaporedje faz razvoja. Vsi razvojni modeli v grobem zajemajo aktivnosti, kot so: analiza, načrtovanje ter izvedba, vendar se med seboj razlikujejo predvsem pri podrobnejši delitvi faz po aktivnostih in načinu ter zaporedju njihovega izvajanja [4].

[2] V tuji strokovni literaturi se par verifikacije in validacije označuje tudi z V&V

Poznamo različne razvojne modele: zaporedni ali slapovni model, inkrementalni model, V-model, iterativni model, prototipni model in druge. V praksi se pogosto uporablja kombinacija različnih modelov.

2.3.1 Zaporedni ali slapovni model



Slika 3: Zaporedni ali slapovni model.

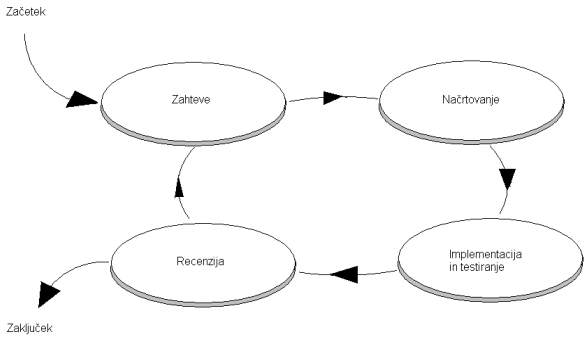
Zaporedni ali slapovni model (ang. Waterfall model) je znan kot prvi splošno sprejet razvojni model. Vse aktivnosti (analiza, načrtovanje, kodiranje, testiranje in implementacija) si sledijo zaporedno in se ne prekrivajo (slika 3). Ker se pri tem razvojnem modelu ni možno vračati na predhodne faze, je zelo pomembno, da so zahteve na začetku vsake faze popolnoma in nedvoumno definirane.

Zaporedni model uvaja disciplinirano izvajanje aktivnosti posameznih faz, ki morajo biti zato dobro dokumentirane, pregledane in testirane. Faza testiranja je pri tem razvojnem modelu samostojna faza.

Glavna pomanjkljivost tega modela je v tem, da ne odraža resničnega razvojnega procesa, saj ob prehodu v naslednjo fazo skoraj nikoli zares ne zaključimo predhodne faze, temveč se po potrebi vračamo (npr. iz načrtovanja in programiranja v analizo). Ravno to pa pri zaporednem načinu ni mogoče. V današnjih poslovnih okoljih, kjer prihaja do pogostih sprememb, je to velika težava.

Po drugi strani pa se je zaporedni model v preteklosti izkazal kot učinkovit pristop pri gradnji velikih in kritičnih sistemov, kjer so bile zahteve znane ali vsaj dobro definirane.

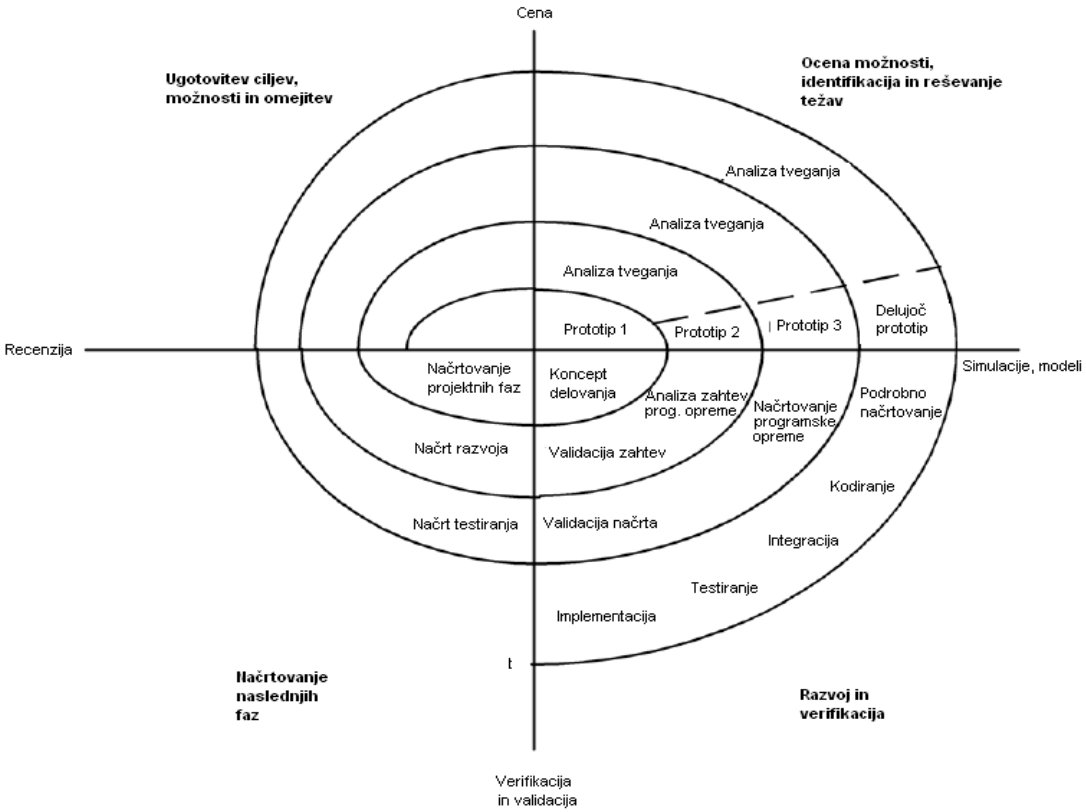
2.3.2 Iterativni modeli



Slika 4: Iterativni model.

Značilnost iterativnega modela (ang. Iterative model) je postopen razvoj. To pomeni, da za razliko od zaporednega pristopa ne končujemo faz v celoti, marveč le delno, cel cikel pa ponavljamo, dokler aplikacija ni zaključena (slika 4). Prednost iterativnih modelov je v tem, da upoštevajo naravo razvojnega procesa, ki večkrat zahteva, da se vračamo v predhodne faze. Takih modelov je več, eden izmed najbolj znanih iterativnih modelov pa je Spiralni model.

2.3.2.1 Spiralni model



Slika 5: Spiralni model.

Spiralni model (ang. Spiral model) združuje značilnosti tako prototipnega kot zaporednega modela. Namenjen je bolj velikim, cenovno obsežnim in zapletenim projektom.

Spirala predstavlja skupne stroške projekta od začetka do trenutnega stanja na projektu [19]. Model predpisuje, da se za vsak cikel spirale izvaja iste korake (slika 5). Ti koraki so sledeči:

- ugotovitev ciljev, možnosti in omejitev (II. kvadrant),
- ocena možnosti, identifikacija in reševanje tveganja (I. kvadrant),
- razvoj in verifikacija (IV. kvadrant) in
- načrtovanje naslednje faze (III. kvadrant).

Vsak cikel se začne z ugotavljanjem:

- ciljev: ti izražajo, kaj naj posamezna faza doseže,
- možnosti: predstavljajo različne poti za doseganje zastavljenih ciljev, in
- omejitve: obsegajo količine, ki omejujejo izvedbo posamezne faze.

Zatem se v naslednjem koraku oceni možnosti ter ugotavlja tveganja na projektu glede na zastavljene cilje in omejitve v predhodnem koraku. Ugotovljena tveganja in njihova ocena določajo nadaljnje aktivnosti projekta. Sledenje spirale nas pripelje do validiranih zadev, načrta razvoja in overjenega načrta testiranja. Pri tem se testiranje nahaja znotraj koraka razvoja in verifikacije (slika 5) in se izvaja že od prvega cikla naprej. Z vsakim ciklom se obseg testnih aktivnosti povečuje.

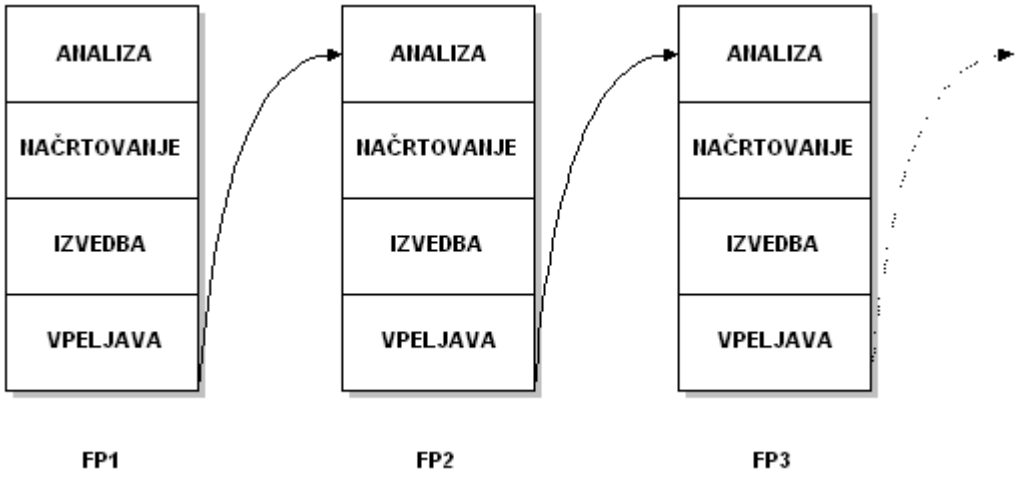
Prednosti se izražajo pri lažjem načrtovanju in ocenjevanju virov (npr. denarna sredstva, čas). Na začetku namreč težko dobro načrtujemo, predstava pa postane bolj realistična kasneje, ko je delo v teku in se odkrije vse pomembne pomanjkljivosti. Poleg tega se lažje prilagajamo spremembam, ki so pogoste pri razvoju programske opreme.

Pomankljivosti spiralnega modela se odražajo v prevelikem prilagajanju na trenutni projekt in omejeni ponovni uporabi na drugih projektih. Obstaja tudi tveganje, da ne ostanemo v okviru finančnih in časovnih rokov.

2.3.3 Inkrementalni model

Inkrementalni ali postopni model je model, pri katerem celoten problem razdelimo na podprobleme ali module, te pa rešujemo vsakega posebej in neodvisno od ostalih modulov (slika 6). Na tak način se programska oprema razvija po delih, od katerih vsak del zajema določeno funkcionalnost sistema. Pri tem morajo biti ti deli dovolj samostojni, da jih lahko vključimo v produkcijo.

Razvoj posameznih modulov razumemo kot samostojne podprojekte. Na tak način sčasoma pokrijemo celotno funkcionalnost sistema ter s tem zaključimo razvoj.



Slika 6: Inkrementalni pristop.

Dobra lastnost inkrementalnega pristopa je v tem, da dele končnega izdelka predamo v uporabo že dovolj zgodaj, kar pripomore k hitremu odkrivanju in odpravljanju napak ali pomanjkljivosti. V primerjavi z razvojem aplikacije v enem delu je razvoj po inkrementalnem modelu v splošnem cenejši ter manj tvegan.

Inkrementalni pristop zahteva razdelitev problema na podprobleme, zato je ob rešitvi vsakega podproblema znotraj ene faze potrebno tega testirati in združiti z ostalimi deli sistema.

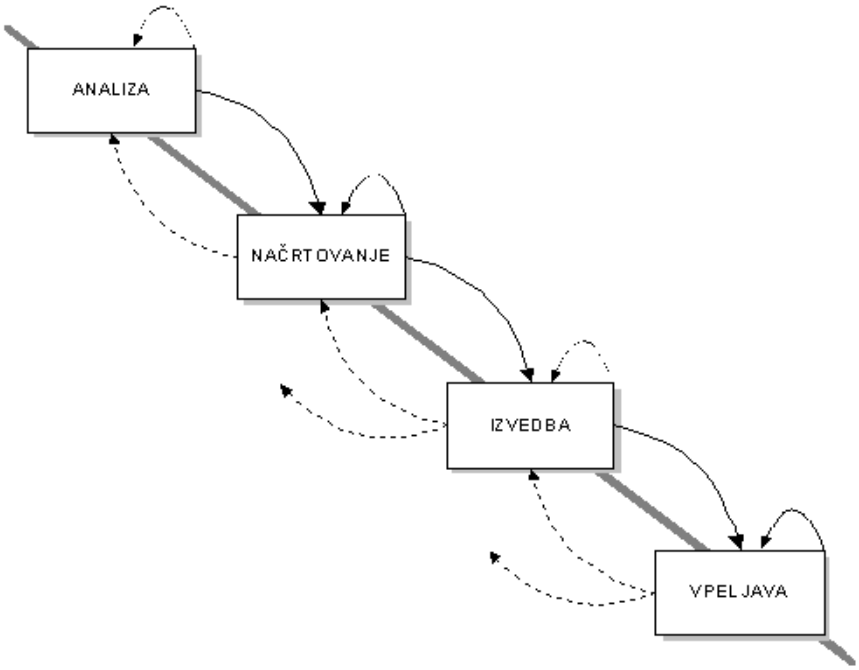
Slabost takega modela je v tem, da ni primeren za vse vrste projektov; sistem za kontrolo zračnega prometa recimo ne more normalno delovati v praksi, dokler ni v celoti dokončan.

2.3.4 Kombinirani model

V veliki meri je izbira ustreznega razvojnega modela odvisna od vrste projekta, včasih pa pridemo do ugotovitve, da bi potrebovali kombinacijo različnih razvojnih modelov.

Kombiniran model je kombinacija zaporednega in iterativnega modela na način, da je omogočeno vračanje v predhodne faze (slika 7). Na tak način ohranimo dobre lastnosti zaporednega modela ter odpravimo njegovo bistveno pomanjkljivost z uvedbo iterativne zanke.

Tak model se v praksi pogosto uporablja, saj je zelo blizu običajnemu procesu razvoja, ki omogoča prehajanje med posameznimi fazami razvojnega procesa, še vedno pa ohranja osnovno zaporedje izvajanja aktivnosti. Testiranje se ne izvaja več samo na koncu neke faze, temveč se lahko ob spremembah pri analizi zahtev ali pri načrtovanju, hitro prilagodi in izvede nove teste za pokritje novih zahtev.



Slika 7: Kombinirani model.

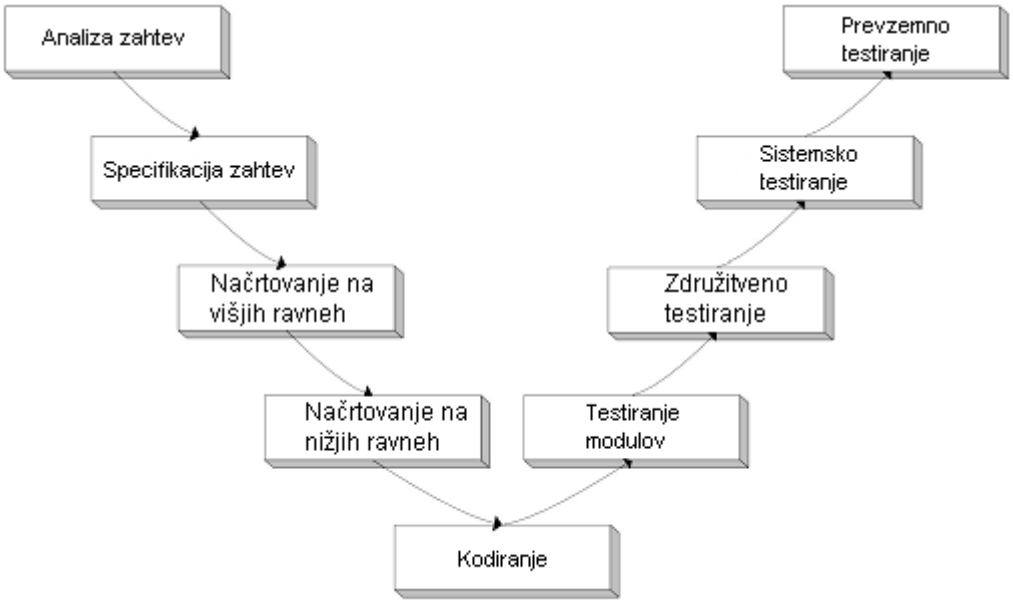
2.3.5 V-model

V-model se lahko jemlje kot razširjena verzija zaporednega modela. Jedro V-modela predstavlja kodiranje, pri čemer so analiza in načrtovanje na levi strani, testiranje ter vzdrževanje pa na desni strani črke modela (slika 8).

V model nakazuje, da se testiranje modulov in združitveno testiranje lahko uporabi pri verifikaciji načrtovanja programa. Med testiranjem modulov in združitvenim testiranjem morajo razvijalci ter člani testne skupine zagotoviti, da so vsi vidiki načrtovanja programske opreme pravilno upoštevani v programski kodi.

Pri sistemskem testiranju je treba na podoben način verificirati načrt sistema, s čimer potrdimo, da so vsi vidiki projektiranega sistema pravilno izvedeni.

Povezave med levo in desno stranjo pri V-modelu nakazujejo tudi, da v primeru nepravilnosti med verifikacijo in validacijo ponovimo aktivnosti na levi strani V-modela. Te se lahko ponovijo zaradi popravkov ali izboljšav pri zahtevah, načrtovanju ali pri programski kodi, še preden se testiranja na desni strani ponovno izvršijo.



Slika 8: V-model.

V primerjavi z zaporednim modelom so v V-modelu bolj vidni popravki in ponovitve.
Če je pri zaporednem modelu poudarek na dokumentaciji in izdelkih, se pri V-modelu pozornost usmeri na aktivnosti in pravilno delujoči sistem.

2.4 Osnovni pojmi testiranja

V nadaljevanju bomo na kratko opisali nekaj najbolj pogostih terminov, s katerimi se lahko srečujemo pri testiranju programske opreme. Nekaj teh definicij je povzetih iz standardov IEEE SCSE[3] [3]. V zbirki se nahaja tudi slovar terminov z nazivom IEEE SGSET[4], v katerem so opisani aktualni termini s tega področja, ki se rabijo tako v akademskih kot v industrijskih krogih.

Napaka
Napaka (ang. error) je pomota, nepravilnost ali nerazumevanje s strani izvajalca.

V to kategorijo spadajo razvijalci, inženirji, programerji, analitiki ter testerji. Ti izvajalci lahko naredijo napake z napačnim tolmačenjem dela načrta ali napačnim poimenovanjem spremenljivke.

Okvara
Okvara (ang. fault) je neposreden rezultat (posledica) napake. To je anomalija v programski opremi, ki lahko povzroči, da se programska oprema obnaša nepravilno, oziroma ne tako kot je določeno v specifikacijah.

[3] IEEE SCSE - IEEE Standards Collection for Software Engineering
[4] IEEE SGSET - IEEE Standard Glossary of Software Engineering Terminology

Okvare se včasih označuje tudi kot “hrošči” (ang. bugs).

Odpoved
Odpoved (ang. failure) je nezmožnost programskega sistema ali njegove komponente, da opravi svojo funkcijo, ali doseže performans,e kot je bilo zahtevano v zahtevah.

V teku izvajanja programskega sistema ali komponente, tester, razvijalec ali uporabnik opazi, da sistem ali komponenta ne vrača pričakovanih rezultatov.

Med razvojem programske opreme se odpovedi običajno opazijo med testiranjem, medtem ko razvijalec odkrije izvor okvare in jo odpravi. Ko je programska oprema v uporabi se lahko zgodi, da se tudi uporabnik sreča z odpovedjo, ki jo sporoči nazaj v razvojno hišo oziroma oddelek.

Lahko se tudi zgodi, da okvara v programski kodi ne povzroči odpovedi. Okvarjena programska oprema lahko obratuje dolgo časa brez očitnih nepravilnosti, ko pa so izpolnjeni določeni pogoji, se napaka nepričakovano pojavi kot odpoved.

Testni primer
Običajen pristop pri odkrivanju nepravilnosti v programski opremi je tak, da tester na določeni zbirki vhodnih podatkov izvrši program pod točno določenimi pogoji. Da bi lahko po izvajanju prepoznal uspeh ali neuspeh izvedenega testa, mora imeti tester na voljo tudi pravilne izhodne rezultate - ti. referenčne podatke. Skupku vseh treh elementov (vhodni podatki, pogoji izvajanja ter izhodni podatki) pravimo testni primer.

Testni primer (ang .test case) se potemtakem sestoji iz naslednjih elementov:

- *zbirke testnih vhodnih podatkov,*
- *pogojev izvajanja in*
- *pričakovanih izhodnih podatkov.*

Sodelovanje med razvijalci, testerji ter osebjem za zagotavljanje kakovosti programske opreme je ključnega pomena pri načrtovanju in pripravi specifikacij za testne primere. Tako se do potankosti določi, kaj naj se nahaja v posameznem testnem primeru.

Test
Test (ang. test) je zbirka sorodnih testnih primerov ter testnih procedur.

Zbirki sorodnih testov se občasno pravi tudi množica testov (ang. test set). Skupini sorodnih testov, ki so povezani s podatkovno bazo in se običajno izvajajo skupaj, pa imenujemo testni nabor (ang. test suite).

Testni prerok
Testni prerok (ang. test oracle) je dokument ali del programske opreme, s katerim se določi, ali se je test uspešno ali neuspešno zaključil.

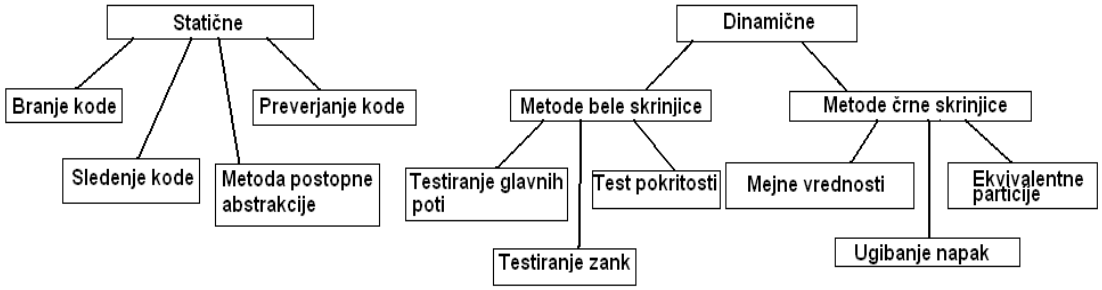
Pogosto je to dokument z zahtevami ali stari program, ki določi, če so pokrite določene funkcionalnosti nove verzije. Delujoči program nam lahko služi tudi kot napoved, kar je posebej uporabno, če prenašamo program v novo okolje.

Testno okolje

Testno okolje (ang. test bed) je okolje, ki vsebuje vso potrebno strojno ter programsko opremo, na kateri lahko kasneje izvajamo test programskih komponent ali programskega sistema.

Okolje vsebuje celotno testno okolje, opremljeno s simulatorji, emulatorji, programskimi orodji, preizkuševalci spomina, sondami ter sploh vsem, kar bi bilo potrebno pri izvedbi testiranja [3].

2.5 Metode testiranja



Slika 9: Tehnike testiranja.

V zadnjih dveh desetletjih in pol so bili razviti štirje pristopi k testiranju:

- statično testiranje,
- metoda bele skrinjice ali strukturna analiza,
- metoda črne skrinjice ali funkcionalna analiza in
- testiranje zmogljivosti ali performančno testiranje.

Metode testiranja lahko delimo na dva načina (slika 9). Prva delitev je delitev na *statične metode* ter *dinamične metode*. Statične metode ne zahtevajo izvajanja programa, torej gre za sledenje programski kodi ter branje dokumentov, zaradi česar govorimo tudi o ročnih metodah testiranja. Pri dinamičnih metodah izvajamo program ter njegove rezultate primerjamo s pričakovanimi rezultati.

Druga je delitev na:

- metodo bele skrinjice ali strukturno analizo (ang. structure-based) in
- metodo črne skrinjice ali funkcionalno analizo (ang. specification-based).

Ena izmed poglavitnih razlik med skupinama je ta, da se pri načrtovanju testnih primerov pri metodi bele skrinjice upošteva notranja struktura kode, saj je programska koda na voljo, torej testerju vidna. V nasprotju s tem je pri metodi črne skrinjice koda zastrta, zato testiramo samo izvršljivi program. Izbiramo lahko različne vhodne podatke in rezultirajoče izhodne podatke primerjamo s pričakovanimi [2].

2.5.1 Metoda črne skrinjice ali funkcionalna analiza

Z metodo črne skrinjice lahko testiramo le funkcionalne zahteve programske opreme, saj je notranja struktura programske opreme zastrta. Pri testiranju izbiramo različne vhodne podatke, dobljene izhodne podatke pa nato primerjamo s pričakovanimi rezultati. Testne primere poskušamo napisati na tak način, da bi s čim manjšim številom testov odkrili čim več možnih napak. Poznamo več metod za testiranje po načelu črne skrinjice: metoda ekvivalentne particije, analiza mejnih vredosti ter ugibanje napak.

Pri tem je potrebno opozoriti, da princip črne skrinjice ni nadomestilo za strukturno testiranje, saj odkriva druge vrste napak, kot so recimo napačne ali manjkajoče funkcije, napake vmesnika, napake pri branju podatkov iz podatkovnih baz in podobno.

2.5.1.1 Ekvivalentne particije

Da bi zmanjšali število testnih primerov, a vseeno pokrili vsa potrebna preverjanja, vhodne podatke razdelimo na ekvivalentne particije ali razrede. Idealno bi bilo, če bi vsakemu razredu zadoščal en testni primer. Razred predstavlja množico veljavnih ali neveljavnih vhodnih podatkov; ti so običajno določene numerične vrednosti, množica vrednosti, interval vrednosti ali Boolova spremenljivka. Pomembno je, da v en ekvivalentni razred vstavimo veljavne vrednosti, v druge pa neveljavne vrednosti.

2.5.1.2 Mejne vrednosti

Analiza mejnih vrednost služi za odkrivanje napak pri mejnih vrednostih vhodnih podatkov. Napake se preverja v bližini praga vrednosti, s čimer naj bi se preverilo, če so omejitve na pragu pravilno obravnavane. Pri tem uporabimo ekstreme vhodnih podatkov za obravnavano domeno (npr. minimum, maksimum, malenkost pod/nad minimumom, malenkost pod/nad maksimumom).

2.5.1.3 Ugibanje napak

Tak pristop je primeren za izkušene testerje, oziroma razvijalce, ki so že imeli izkušnje s podobno programsko opremo v smislu podobne problemske domene, načrtovanja, zahtevnosti in podobno.

Tester oziroma razvijalec lahko na temelju predhodnih izkušenj ugiba, kakšne napake bi se lahko nahajale v programu. Zato napiše tak testni primer, da bi lahko tako napako odkril. Tipičen primer je recimo preizkus deljivosti s številom nič. Ugibanje napak velja za ad hoc pristop pri načrtovanju testa.

2.5.2 Metoda bele skrinjice ali strukturna analiza

Za princip bele skrinjice se odločimo predvsem takrat, ko testiramo posamezne module. Pri tem izvajamo teste, ki temeljijo na poznavanju notranje strukture programske kode. Med metode bele skrinjice spadajo testiranje zank, testiranje glavnih poti ter testiranje pokritosti.

2.5.2.1 Testiranje zank

Skoraj vsak algoritem vsebuje zanke in te so si običajno zelo podobne. Zato obstaja velika verjetnost, da lahko pri pisanju zank naredimo drobne, težko opazne napake. Pri sestavi testnih primerov je zato pomembno, da vključimo tudi testiranje zank, še posebej v okolici mejne vrednosti ponovitev. Testni primeri naj pokrivajo tako preproste kot vgnezdene zanke.

2.5.2.2 Testiranje glavnih poti

Pri tej metodi je bistveno, da sestavimo take testne primere, s katerimi bi izvedli vsak ukaz vsaj enkrat. Pri tem si lahko pomagamo kar z diagramom poteka. Test mora pokriti vse možne poti na diagramu poteka. Velja opozoriti, da ne pokrivamo vseh kombinacij poti, ampak samo veljavne poti v diagramu.

2.5.2.3 Test pokritosti

Pri testu pokritosti želimo izvesti vsak programski ukaz v programski kodi ter pokriti vsako možno pot v programu. Ker velikost tega števila lahko sega celo v neskončnost, je cilj pri tej metodi najti najboljšo možno pokritost programske kode glede na cenovne ter časovne omejitve.

2.5.3 Kombinacija metode črne in bele skrinjice

Obe metodi testiranja se uporabljata za iskanje istih stvari, vendar z različnih vidikov. Vsaka po svoje sta pomembni pri testiranju programske opreme, vendar bi neodvisna izvedba obeh hkrati pomenila tratenje virov.

Pri tej metodi tako velja, da za testiranje najnižjih nivojev kode uporabimo tehniko ročnega pregledovanja kode, za tem pa lahko uporabimo metodo črne skrinjice. Po končanem pregledu se lotimo notranje strukture kode z metodo bele skrinjice, saj tega metoda črne skrinjice ni pokrila. V stroki se za tovrstno metodo uporablja izraz siva skrinjica.

V primeru, da od naročnika prejmemo le izvršljivo kodo brez programske kode, metode bele skrinjice ne moremo uporabiti. V takem primeru uporabimo le metodo črne skrinjice, a smo pri testiranju zelo pozorni ter previdni. V praksi je seveda najbolje uporabljati oboje.

2.5.4. Statična in dinamična delitev

Z vidika druge delitve lahko posebej opredelimo ročne metode testiranja.

2.5.4.1 Ročne metode testiranja

Branje kode je najbolj tradicionalna metoda preverjanja delovanja programa. Branje kode opravlja tipično nekdo drug in ne avtor kode, saj se s tem dosežejo boljši rezultati pri iskanju pomanjkljivosti in napak. S tem dobimo tudi bolj neodvisno oceno, še zlasti, če uporabimo anonimno recenzijo.

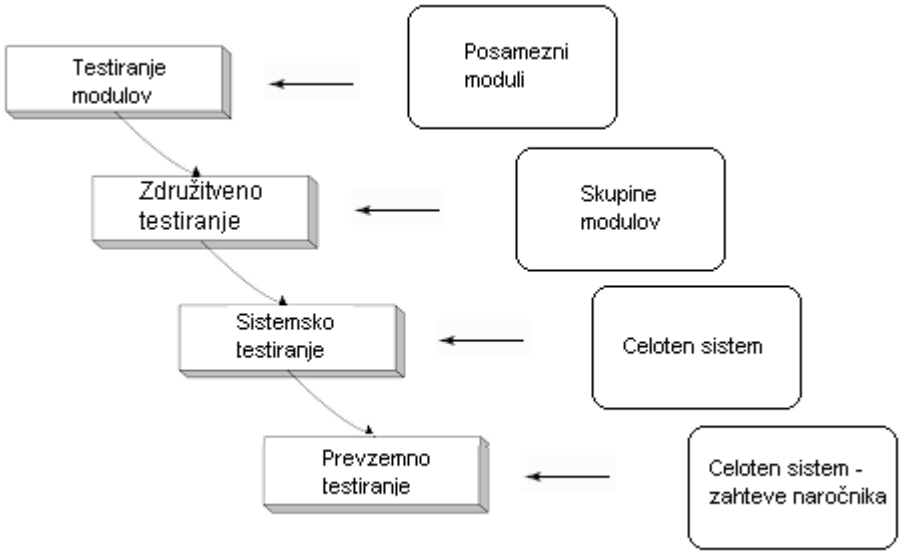
Sledenje in preverjanje kode izvajamo na formalnih recenzijskih sestankih, če je rezultatov več. Pri tem se kodo prebira, hkrati razlaga njen pomen ter tako pripomore k odkitju napak.

Sledenje opravimo na programski kodi s pomočjo testnih podatkov. Testni primeri ne smejo biti kompleksni in dolgi, temveč kratki in jedernati, da ne bi bilo sledenje prezahtevno.

Pri metodi postopne abstrakcije poskušamo iz programske kode v nekaj korakih razbrati funkcijo, ki jo ta koda implementira. To funkcijo nato primerjamo s funkcijo, ki je opisana v specifikaciji. Za uspešno prestani test morata funkciji biti skladni.

2.6 Nivoji testiranja

Preden se programsko opremo preda naročniku, mora iti skozi štiri nivoje testiranja: testiranje modulov (ang. unit test), združitveno ali integracijsko testiranje (ang. integration test), sistemsko testiranje (ang. system test) ter prevzemno testiranje (ang. acceptance test). Prve tri teste izvedejo razvijalci, zadnjega pa običajno izvede naročnik, skupaj z razvijalci. Štirje omenjeni nivoji so lepo razvidni na sliki 10, kjer so prikazani v obliki V-modela.

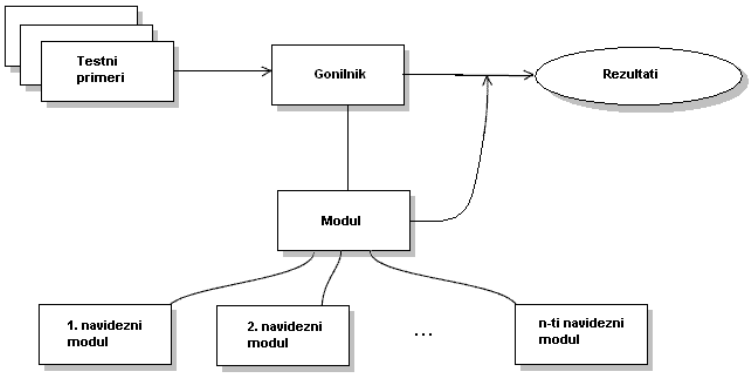


Slika 10: Predstavitev nivojev testiranja - V-model.

V nadaljevanju bomo predstavili vsak nivo testiranja posebej [4].

2.6.1 Testiranje modulov

Pri testiranju modulov se razvijalci osredotočijo na test posameznih modulov oziroma enot, kot so procedure, funkcije, metode ali razredi. Po potrebi testiramo tudi zanke, vmesnike ter druge elemente. V ta namen postavimo posebno testno konfiguracijo, ki nadomesti sistem, da bomo lahko vanj vgradili modul. Potreben je gonilnik (ang. driver), ki kliče testirani modul, in navidezni modul (ang. stub), ki nadomešča podrejene module (slika 11). Bolj, ko je modul notranje enoten in manj, ko je soodvisen, lažje ga je testirati [2].



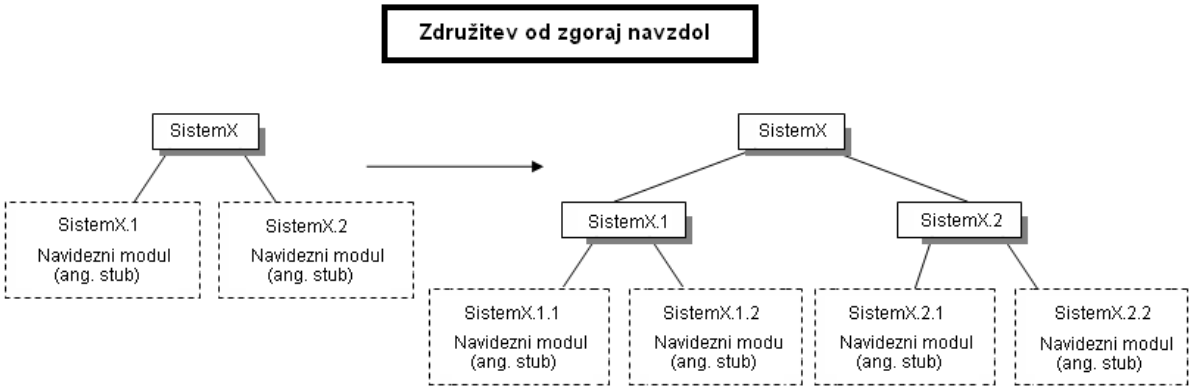
Slika 11: Testiranje modulov.

2.6.2 Združitveno testiranje

Postopek združitve poteka postopno, zato testiranje poteka po metodi črne skrinjice. Ob vsaki razširitvi sistema ali dodajanju novega modula je potrebno ponoviti testiranje. S postopno vpeljavo novih modulov in ponovnim testiranjem lažje poiščemo vzroke za napake.

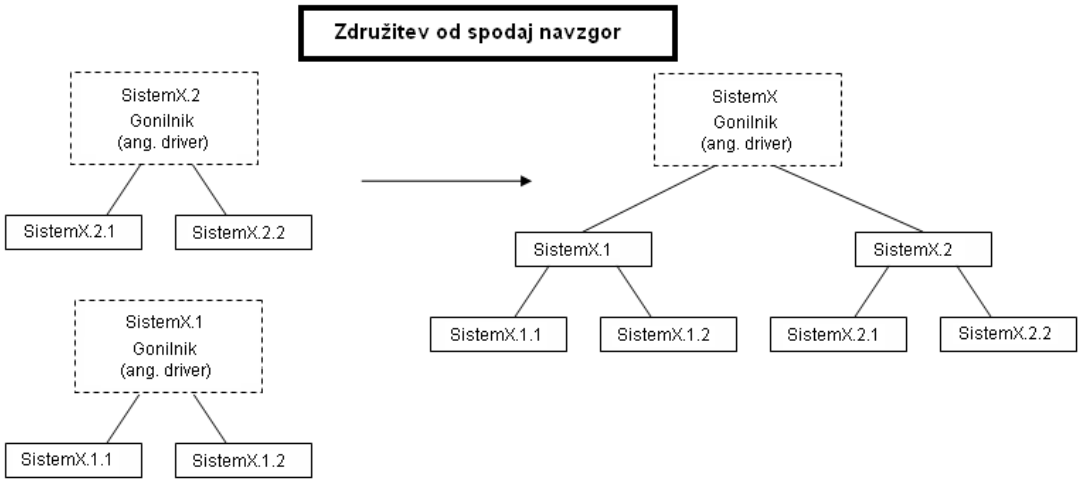
Sama združitev lahko poteka na več načinov:
- združitev od zgoraj navzdol,
- združitev od spodaj navzgor in
- kombinacija zgornjih dveh.

Izbira načina je odvisna od težavnosti testiranja.



Slika 12: Združitveno testiranje od zgoraj navzdol.

Združitev od zgoraj navzdol (slika 12). Začnemo z glavnim ali kontrolnim modulom ter postopno dodajamo podrejene module v globino ali v širino. Če združujemo najprej v globino, lahko prej preverimo določene funkcije sistema, saj so potrebni le podrejeni moduli. Problemi nastopijo le, če je simuliranje procesiranja v nižje ležečih modulih težavno.



Slika 13: Združitveno testiranje od spodaj navzgor.

Združitev od spodaj navzgor (slika 13) poteka tako, da module na nižjem nivoju združimo v skupine. Napisati moramo gonilnike, ki koordinirajo testiranje skupin modulov. Nato postopoma odstranjujemo gonilnike, jih nadomeščamo s pravimi moduli ter skupine združujemo.

Zadnji pristop sestavlja združitev od zgoraj navzdol ter od spodaj navzgor. S pristopom od spodaj navzgor združujemo in testiramo posamezne module na nižjem nivoju, ki jih nato združimo v skupine. S pristopom od zgoraj navzdol se medtem lotimo testiranja glavnega ali kontrolnega modula. Na ta način se z združevanjem zmanjša število potrebnih navideznih modulov. Proces dodaja podskupine, ki so nastale pri združitvi od spodaj navzgor, ter jih postopoma priključuje glavnemu modulu. Ta pristop se obnese, če razvijalci dobro poznajo izdelek, ki ga razvijajo, pristop z združitvijo navzdol ali navzgor pa bi bil zaradi časovnih omejitev nemogoč [2].

2.6.3 Sistemsko testiranje

Sistemsko testiranje je kritični del v razvoju programske opreme. Ko je programska oprema nameščena v svojem delovnem okolju, je treba opraviti validacijo sistema. To pomeni preveriti najprej skladnost s predpisano dokumentacijo, nato pa še testirati celoten sistem. Zanimajo nas tudi nefunkcionalne zahteve, kot so: hitrost, zmožnost okrevanja sistema po izpadu, maksimalna obremenitev sistema, varnost in občutljivost sistema. Zanima nas tudi, kaj se zgodi ob vnašanju napačnih podatkov ali ukazov in podobno. Vrste sistemskega testiranja so:

- *Test okrevanja* sistema izvajamo tako, da povzročimo izpad, nato pa opazujemo, če se sistem dovolj hitro zopet postavi, pravilno deluje in odziva na naše zahteve.

- *S testom varnosti* sistema je potrebno ugotoviti, če lahko sistem prepreči neavtoriziran dostop ter tako ohrani integriteto. Obenem preverimo odziv sistema na viruse, ustrezno upoštevanje poteka veljavnosti gesel, ter uporabo enkripcije/dekripcije in podobno.

- *Test obremenitve* sistema preveri, če se sistem v primeru velikih obremenitev (npr.pri večjem številu uporabnikov, pri večji porabi spomina itd.) lahko normalno postavi in deluje.

- Pri *testu občutljivosti* preverjamo, če lahko različne kombinacije pravilnih vhodnih podatkov privedejo do nestabilnosti sistema ali napačnega procesiranja le-teh.

2.6.4 Alfa in beta testiranje

Pred predajo programske opreme naročnikom lahko izdelek predamo v poskusno uporabo zainteresiranim posameznikom ali skupinam. Če se to izvaja znotraj razvijalske hiše, to poimenujemo alfa testiranje, če pa se izda v uporabo izven razvijalske hiše (npr. naročniku), se to imenuje beta testiranje. Po uspešnem alfa ali beta testiranju sledi prevzem programske opreme s prevzemnim testiranjem.

2.6.5 Prevzemno testiranje

Po uspešnem sistemskem testiranju sledi prevzemno testiranje. Tukaj gre za pazljivo načrtovan testni postopek z realnimi vhodnimi podatki stranke. Programska oprema mora biti nameščena na strojni opremi, ki jo uporablja naročnik v resničnem svetu. Optimalno bi bilo, da se sistem pred prevzemom v realnem okolju testira vsaj 25 ur. Testni primeri so ustvarjeni po zahtevah uporabnikov.

Prevzemno testiranje predstavlja pomemben mejnik za razvijalce, zato se je treba nanj pripraviti in ga izvesti na profesionalen način. Pri lepo utečenih projektih daje prevzemno testiranje videz demonstracije programske opreme.

Uspešno zaključeno prevzemno testiranje pomeni, da bo stranka uspešno pričela z uporabo programske rešitve, s tem pa je potrdila njeno koristnost in vrednost.

2.7 Regresijsko testiranje

Regresijsko testiranje (ang. regression testing) ne velja za nivo testiranja, temveč predstavlja vnovično testiranje programske opreme, ki se običajno izvaja na različnih nivojih testiranja. V kolikor uporabnik ali tester najde napako, je treba posodobiti ali ustvariti nove testne primere, da s tem zagotovimo, da se podobna napaka ne bo več ponovila v prihodnjih verzijah [3].

2.8 Razhroščevanje

Razhroščevanje (ang. debugging) ali popravljanje napak se ne sme zamenjevati s testiranjem. Pri razhroščevanju iščemo vzroke za napake ter jih odpravljamo, medtem ko testiranje išče napake.

3. Odprtokodna izvedba okolja za celovito izvajanje testiranja

3.1 Opis problemske domene

Testiranje se nahaja v vsaki fazi razvoja, zato je zapleteno slediti vsem pridobljenim informacijam in podatkom, načrtom, specifikacijam in znanju. Kaj hitro se lahko namreč zgodi, da se določene informacije izgubijo v morju sporočil elektronske pošte, nepreglednih imenikov repozitorija ali pa celo le dopišejo v opombo pri neki prijavljeni napaki, ki jo po rešitvi napake spregledamo. Na začetku projekta je morda še nekaj časa lahko slediti spremembam in najdenim napakam v sistemu, ko pa sistem narašča, se kaj hitro lahko zgodi, da spremembam ni več mogoče slediti.

Zato smo se v tej nalogi odločili, da prikažemo kako se lahko iz posameznih odprtokodnih rešitev z malo truda in časa pride do celovitega okolja za izvajanje testiranja. S tem bo olajšano upravljanje za vse člane na vseh nivojih, ki so povezani pri samem procesu testiranja na nekem projektu, kot tudi za upravljalce, ki bi radi preverili stanje projekta.

Rešitev tako predstavlja združitev več spletnih aplikacij, s katerimi lahko zagotovimo enostaven in celovit sistem.

3.2 Opis uporabljenih orodij

Pri izbiri orodij smo se osredotočili na že uveljavljene odprtokodne sledilce napak ter orodja za upravljanje s testnimi primeri. Pri tem smo upoštevali, da v postopku združevanja ne bi prihajalo do neujemanj pri programskih in strojnih zahtevah. Tako smo izbrali sledilec napak z imenom Mantis, ki se ga lahko lepo poveže z orodjem za upravljanje s testnimi primeri TestLink. Za sistem za zbiranje znanja ter vsebin (ang. knowledge and content management system) smo izbrali orodje MediaWiki. S tem smo določili prostor za razne članke, napotke, opise tehnik ter procedur in podobno v našem okolju. Vključili smo tudi forum, kjer si lahko razvijalci, testerji, upravljalci in ostali izmenjujejo mnenja. Vse skupaj smo sestavili v celoto, kjer kot center služi sistem CMS[5], Joomla!.

[5] CMS (ang. Content Management System) – Sistem za upravljanje vsebin.

3.2.1 MantisBT

MantisBT [11] je odprtokodna izvedba spletne aplikacije sledilca napak. Napisan je v programskem jeziku PHP, ob podpori dobrega sistema za upravljanje podatkovnih baz MySQL. Namestitev se lahko izvaja na različnih platformah (Windows, Linux, Mac OS itd.). Deluje pa pod pogoji uporabe GNU GPL[6]. Poleg tega je še zlasti pomembno, da se zelo dobro sestavi z orodjem za upravljanje s testnimi primeri TestLink.

3.2.2 TestLink

Tudi TestLink [12] je ravno tako odprtokodna izvedba spletne aplikacije, ki predstavlja orodje za upravljanje s testnimi primeri. Z njim je mogoče ustvariti in upravljati testne primere kot tudi organizirati le-te v testnih načrtih. S testnimi načrti se omogoča testnim članom, da izvajajo testne primere ter sledijo rezultatom z dinamično predstavitvijo testnih poročil, izvajanjem testnih poročil, sledenjem programskim zahtevam, dodeljevanjem opravil članom in podobnim. Tudi to orodje temelji na programskem jeziku PHP in sistemu za upravljanje s podatkovno bazo MySQL.

3.2.3 MediaWiki

MediaWiki [15] je priljubljena odprtokodna spletna aplikacija, ki se velikokrat uporablja za upravljanje z bazo znanj v marsikaterem podjetju [9]. Tudi MediaWiki temelji na programskem jeziku PHP, kar daje slutiti, da ne bo težav ob združitvi s sistemom za upravljanje podatkovnih baz MySQL.

3.2.4 Joomla!

Joomla! [13] je odprtokodna izvedba spletne aplikacije za upravljanje s spletnimi vsebinami. Je zelo priljubljeno orodje za ustvarjanje in upravljanje spletnih vsebin. Z velikim naborom vtičnikov (ang. plugin) ter razširitev (ang. extension) lahko dodatno razširimo funkcionalnosti orodja. Kot vsi sistemi tudi ta podpira programski jezik PHP ter sistem za upravljanje s podatkovnimi bazami MySQL. Joomla! je bil izbran kot centralna točka našega okolja za celovito izvajanje testiranja.

3.2.5 Kunena Forums

Kunena forum [10] je brezplačna razširitev za sistem Joomla!. Omogoča hitro dodajanje znotraj samega sistema ter lahko namestitev. Za potrebe tega projekta je zmogljivost več kot zadovoljiva. Vsebuje tudi veliko nastavitev, s katerimi je možno forum na hiter in enostaven način prilagoditi našim potrebam.

---

[6] GPL - General Public License

3.3 Namestitev

V nadaljevanju bomo skušali prikazati, kako se postavi okolje za celovito izvajanje testiranja. Postopek smo opravili na operacijskem sistemu Windows Server 2003, vendar ga je enostavno izvesti tudi na kakšnem drugem operacijskem sistemu npr. Ubuntu Linux.

3.3.1 Predpogoji za namestitev sistema

Celotna namestitev sistema se izvaja preko spletnega brskalnika. Zato je pred namestitvijo sledilca napak, orodja za upravljanje s testnimi primeri in ostalimi orodji, potrebno na sistem namestiti spletni strežnik, sistem za upravljanje s podatkovno bazo, ter zagotoviti podporo programskemu jeziku PHP.

Temu najlažje in najhitreje zadostimo, če namestimo programski paket odprtokodnih orodij XAMPP[7], ki ga dobimo na naslovu [16].

XAMPP je programski paket odprtokodnih orodij za hitro namestitev spletnega strežnika Apache, sistema za upravljanje s podatkovnimi bazami MySQL ter podporo programskemu jeziku PHP.

Programski paket XAMPP vsebuje še nekaj dodatnih orodij, ki pa presegajo obseg te diplomske naloge. Nekatera med njimi so: strežnik elektronske poste Mercury, FTP strežnik FileZilla, OpenSSL itd.

Dobra lastnost XAMPP programskega paketa je, da je na voljo za različne platforme (Windows, Linux, Mac).

3.3.2 Namestitev okolja

Ker smo uporabili programski paket XAMPP za postavitev spletnega strežnika, se bomo v nalogi sklicevali na privzeto relativno pot do imenika namestitve z uporabo oznake `<xampp_dir>`. Tako izgled sklica na spletni strežnik označuje pot `<xampp_dir>\htdocs`.

3.3.2.1 Predpriprava na namestitev orodij

Preden se lotimo same namestitve spletnih aplikacij, je potrebno preveriti, ali lahko na spletnem strežniku dostopamo do sistema za upravljanje podatkovnih baz MySQL. S tem lahko tudi sočasno preverimo, če spletni strežnik podpira izvajanje skript v programskem jeziku PHP. Za to opravilo se lahko uporabi kar primer programske kode, ki je prikazan na sliki 14. S skripto se povežemo na podatkovno bazo, če je povezava uspela, ta izpiše pridobljene podatke. V nasprotnem primeru nas obvesti, da povezava ni uspela.

[7] XAMPP – X(cross-platform)/Apache/MySQL/PHP/Perl

```
1: <html>
2: <head><title>PHP and SQL Integration Test</title></head>
3: <body>
4: <?php
5: // Connecting, selecting database
6: $link = mysql_connect("<IME PODATKOVNEGA STREZNIKA ali IP NASLOV>",
          "<UPORABNISKO IME SISTEMA>", "<UPORABNISKO IME SISTEMA>")
           or die("Could not connect");//(localhost,root,'<blank>')
7: print "Connected successfully";
8:
9: // Performing SQL query
10: $query = "SELECT now()";
11: $result = mysql_query($query)
12:     or die("Query failed");
13:
14: // Printing results in HTML
15: print "<table>\n";
16: while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
17:     print "\t<tr>\n";
18:     foreach ($line as $col_value) {
19:         print "\t\t<td>$col_value</td>\n";
20:     }
21:     print "\t</tr>\n";
22: }
23: print "</table>\n";
24: // Closing connection
25: mysql_close($link);
26: ?>
27: </body>
28: </html>
```

Slika 14: Vsebina datoteke `helloSQL.php`.

Skripto shranimo v imenik spletnega strežnika, v našem primeru je to `<xampp_dir>\htdocs`. Zatem v naslovno vrstico v spletnega brskalnika vnesemo naslov, oziroma pot do skripte `http://localhost/helloSQL.php`. Če je povezava uspela, se nam v spletnem brskalniku prikaže izpis "`Connection successful`". V nasprotnem primeru smo nekje naredili napako, ki jo je potrebno odkriti in odpraviti, še preden nadaljujemo z namestitvijo orodij.

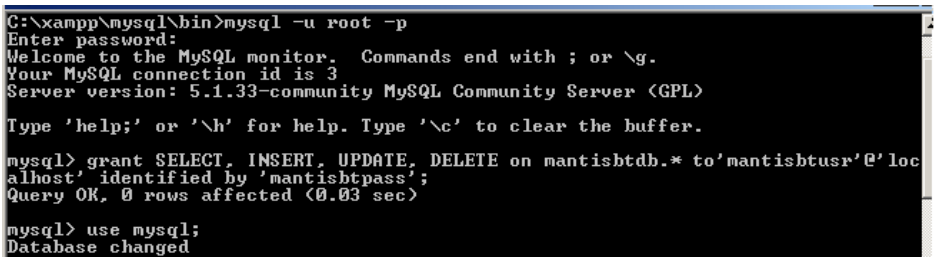
3.3.2.2 Dodajanje uporabnikov v sistem MySQL

Preden začnemo nameščati posamezne komponente našega sistema, je dobro, da si v podatkovni bazi že vnaprej pripravimo vsa potrebna uporabniška imena ter jim dodelimo potrebne pravice za dostop in upravljanje podatkovnih baz v sistemu za upravljanje s podatkovnimi bazami. Dodeljene uporabniške pravice naj bi obsegale izvajanje operacij, kot so: `SELECT`, `INSERT`, `UPDATE` in `DELETE`, za samo namestitev sistemov pa potrebujemo status administrativnega uporabnika, ki ima poleg zgoraj naštetih operacij tudi pravice za izvajanje operacij, kot so: `INDEX`, `CREATE`, `ALTER` in `DROP`. [5, 7].

V ukazni vrstici (ali lupini) se premaknemo v imenik `<xampp_dir>\mysql\bin`. Izvedemo ukaz `mysql -u root -p`. Tako se znajdemo znotraj MySQL opravilne vrstice, kjer vpišemo in izvršimo zaporedje ukazov (slika 15 in 16).

```
1. grant SELECT, INSERT, UPDATE, DELETE on mantisdb.* to
   'mantis_user_name'@'localhost' identified by
   'mantis_user_password';
2. grant SELECT, INSERT, UPDATE, DELETE on testlinkdb.* to
   'testlink_user_name'@'localhost' identified by
   'testlink_user_password';
3. grant INDEX, CREATE, SELECT, INSERT, UPDATE, DELETE,
   ALTER, LOCK TABLES on mediawikidb.* to
   'mediawiki_user_name'@'localhost' identified by
   'mediawiki_user_password';
4. use mysql;
```

Slika 15: Zaporedje ukazov za dodajanje in dodeljevanje pravic uporabnikom.



Slika 16: Dodajanje uporabniškega računa znotraj ukazne vrstice MySQL.

Z izvršitvijo ukazov, ki so prikazani na sliki 17, lahko preverimo, če se pravice uporabnikov ujemajo z dodeljenimi pravicami iz prejšnjega koraka (slika 18).

```
1. show grants for mantis_user_name@localhost;
2. show grants for testlink_user_name@localhost;
3. show grants for mediawiki_user_name@localhost;
```

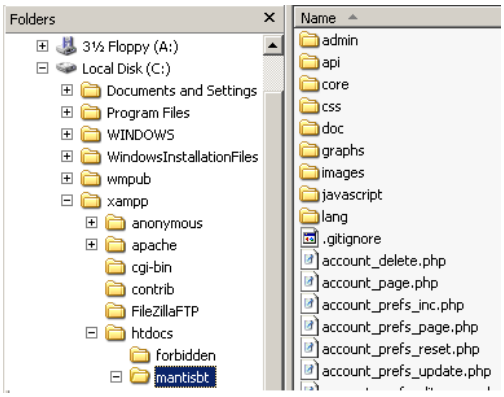
Slika 17: Zaporedje ukazov za preverjanje uporabniških pravic.

Slika 18: Rezultat poizvedbe znotraj ukazne vrstice MySQL.

Na ta način smo ustvarili tri uporabnike ter jim določili pravice. Vnos uporabniških računov ter dodeljevanje pravic se lahko izvede tudi z orodjem phpMyAdmin, ki je del programskega paketa XAMPP. PhpMyAdmin je odprtokodna spletna aplikacija, ki vsebuje le nekaj skriptnih datotek PHP in nam omogoča upravljanje podatkovnih baz MySQL kar preko spletnega brskalnika.

3.3.2.3 Namestitev orodja MantisBT

Arhivsko datoteko razširimo, po potrebi preimenujemo in premaknemo v imenik spletnega strežnika, npr. `<xampp_dir>\htdocs` kot je prikazano na sliki 19.

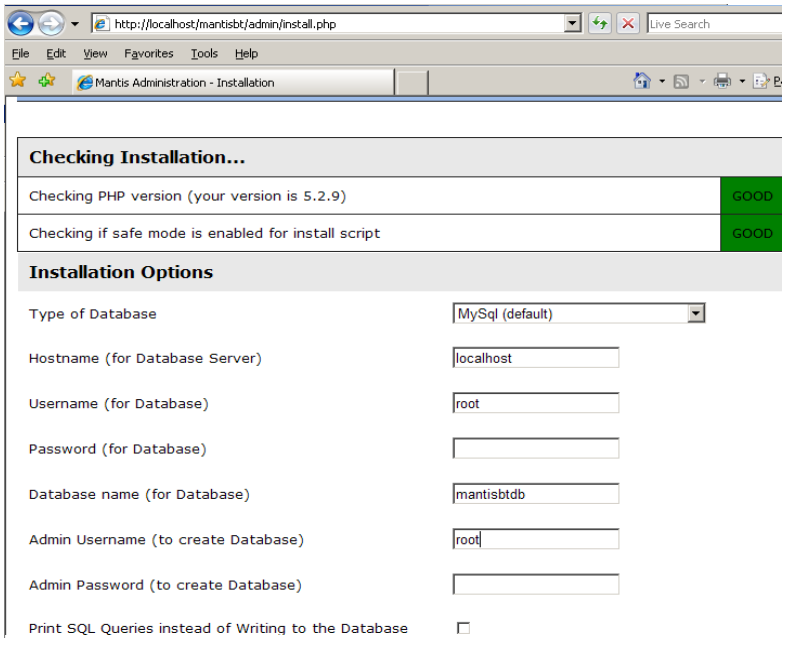


Slika 19: Imenik `<xampp_dir>\htdocs` ter vsebina imenika `mantisbt`.

V spletni brskalnik vnesemo pot do namestitvene strani ter pričnemo s postopkom namestitve (slika 20). V vnosna polja vnesemo potrebne podatke, vključno z uporabniškim računom ter imenom podatkovne baze, ki smo ju ustvarili že v pripravljalnem postopku `(mantis_user/mantis_pass/mantisdb)`.

Pregled stanja namestitve lahko preverimo tako, da v spletni brskalnik vnesemo pot do strani za preverjanje nastavitev `http://localhost/mantis/admin/check.php`.

Po namestitvi je pomembno, da izbrišemo `admin` imenik, ki se nahaja pod `mantisbt` korenskim imenikom, ter se tako izognemo možni zlorabi.



Slika 20: Postopek namestitve sistema MantisBT.

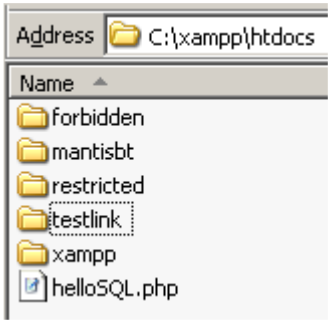
Konfiguracija aplikacije se večinoma izvaja s prilagajanjem datoteke `config_inc.php` (slika 21). V konfiguracijsko datoteko sedaj dodamo predhodno ustvarjen uporabniški račun, ki ima bolj omejene pravice za delo s podatkovno bazo [7, 8].

```
<? php
...
        // Configure the database
        $g_hostname = 'localhost';
        $g_db_type = 'mysql';
        $g_database_name = 'mantisbtdb';
        $g_db_username = 'root'; // <- mantis_user
        $g_db_password = '';  // <- mantis_pass

        //Customizing Mantis
        $g_window_title = 'Mantis QA';

        //File upload settings
        $g_allow_file_upload = ON;

        //Reset password confirmation e-mail
        $g_send_reset_password = OFF;
...
?>
```

Slika 21: Izsek vsebine konfiguracijske datoteke `config_inc.php`.

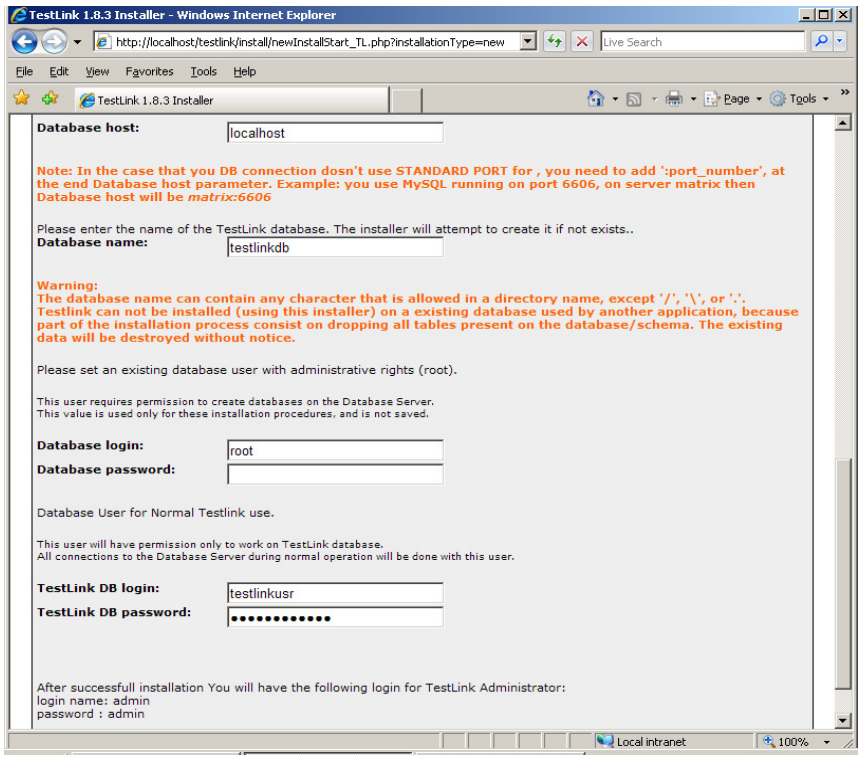
3.3.2.4 Namestitev orodja TestLink

Tako kot pri namestitvi aplikacije MantisBT, tudi tokrat razširimo arhivsko datoteko, jo po potrebi preimenujemo in premaknemo v imenik spletnega strežnika `<xampp_dir>\htdocs`. Drevesna struktura spletnega strežnika je prikazana na sliki 22.



Slika 22: Podoba imenika spletnega strežnika `<xampp_dir>\htdocs`.

V spletnem brskalniku odpremo namestitveno stran aplikacije. Med postopkom namestitve v vnosna polja vnesemo potrebne podatke, vključno z uporabniškim računom ter imenom podatkovne baze, ki smo ju ustvarili že v pripravljalnem postopku ; `(testlink_user/testlink_pass/testlinkdb)` (slika 23).



Slika 23: Postopek namestitve sistema TestLink.

Po namestitvi izbrišemo imenik `install`, ki se nahaja v korenskem imeniku `testlink`, da se tako izognemo možni zlorabi.

Pregled konfiguracijskih datotek orodja TestLink [5]:

- `config.inc.php` je konfiguracijska datoteka glavnega menija ter služi kot ovojna datoteka (ang. wrapper) za ostale konfiguracijske datoteke. Datoteka vsebuje privzete vrednosti parametrov sistema. Sklic nanjo se nahaja skoraj v vsaki strani sistema.
- `custom_config.inc.php` je nastavitvena datoteka (slika 24). V njej nastavljamo različne parametre, ki so posebej prirejeni za naš sistem, ne da bi posegali v konfiguracijsko datoteko.
- `/cfg/<bug_tracker>.cfg.php` je datoteka, v kateri nastavimo dostopne podatke za dostop do podatkovne baze sledilca napak.

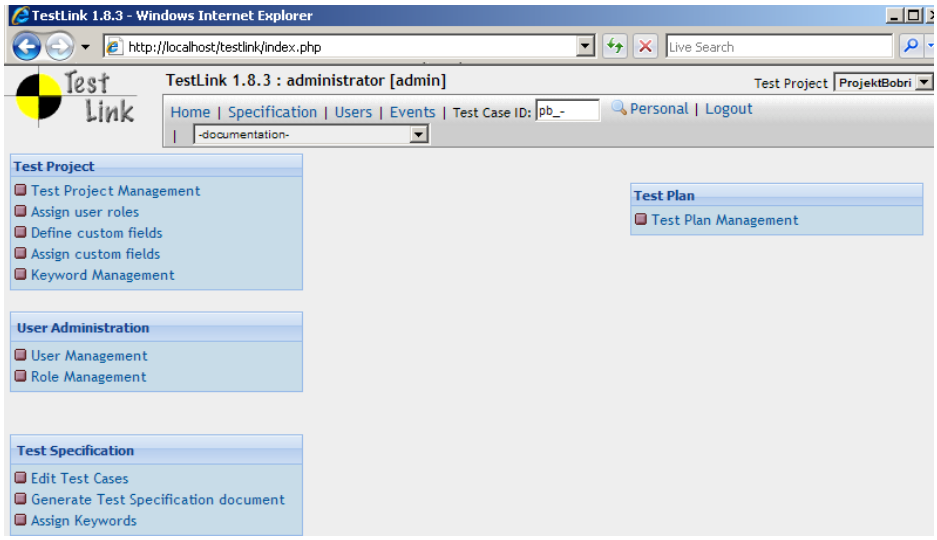
Konfiguracijska datoteka `config_inc.php` se nahaja v korenskem imeniku aplikacije. Priporočljivo je, da se za prilagajanje aplikacije to datoteko čimmanj spreminja. V ta namen raje uporabimo datoteko `custom_config_inc.php`.

```
<? php
...
      # SMTP server Configuration
      $g_smtp_host   = 'smtp.mailserver.com'; #SMTP server

      # Configure using custom_config.inc.php
      # for problem/error notification
```

```
        $g_tl_admin_email = 'tl_admin@mailserver.com';

        # email sender
        $g_from_email = 'testlink@mailserver.com';
        $g_return_path_email  = 'ales.vetrih@mailserver.com';

        # Urgent = 1, Not Urgent = 5, Disable = 0
        $g_mail_priority = 5;
...

?>
```

Slika 24: Izsek nastavitvene datoteke `custom_config_inc.php`.

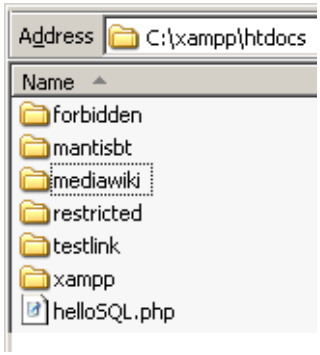
Na sliki 25 je predstavljen vmesnik spletne aplikacije za upravljanje s testnimi primeri TestLink. Kot je že iz vmesnika razvidno, ne omogoča zgolj upravljanja s testnimi primeri, ampak ponuja tudi možnosti dela s testnimi načrti, dodeljevanje uporabniških vlog, generiranje testnih specifikacij, statistik ter reportov in drugo.



Slika 25: Vmesnik spletne aplikacije za upravljanje s testnimi primeri -TestLink.

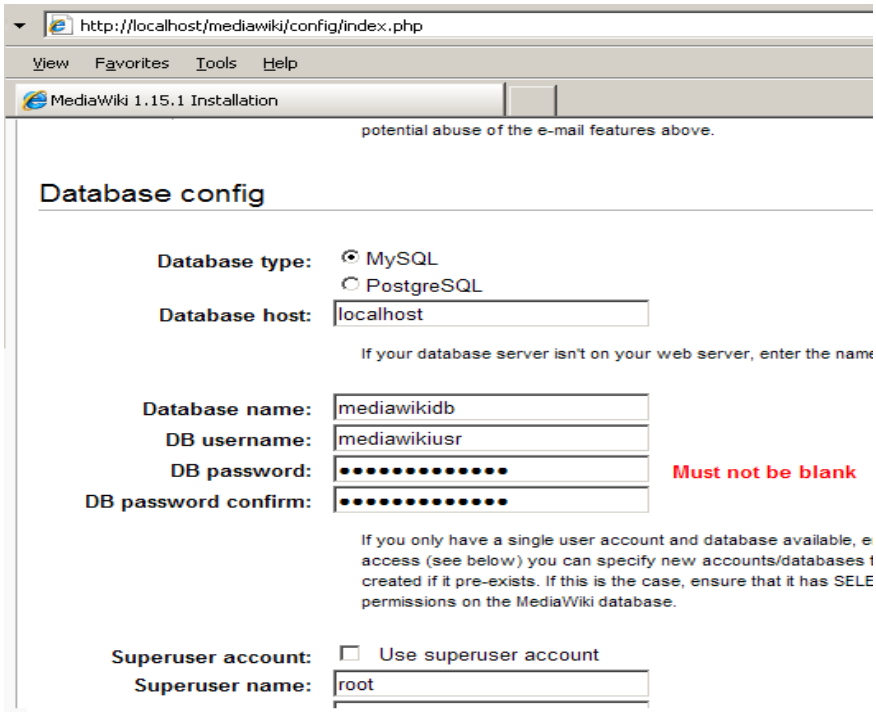
3.3.2.5 Namestitev orodja MediaWiki

MediaWiki arhivsko datoteko razširimo, po potrebi preimenujemo in premaknemo v imenik spletnega strežnika `<xampp_dir>\htdocs`. Slika 26 prikazuje drevesno strukturo spletnega strežnika.



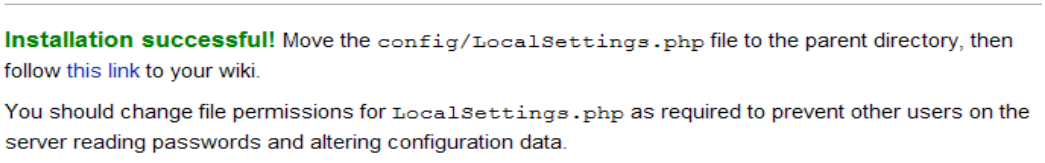
Slika 26: Podoba imenika spletnega strežnika `<xampp_dir>\htdocs`.

Preko spletnega brskalnika izvedemo postopek namestitve aplikacije. Med postopkom namestitve v vnosna polja vnesemo potrebne podatke, vključno z uporabniškim računom ter imenom podatkovne baze, ki smo ju ustvarili že v pripravljalnem postopku (slika 27).



Slika 27: Postopek namestitve sistema MediaWiki.

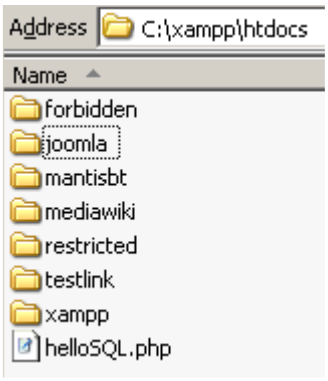
Po postopku namestitve je potrebno premakniti datoteko `LocalSettings.php` iz imenika `<mediawiki_dir>\config\` en nivo višje v `<mediawiki_dir>\` korenski imenik (slika 28).



Slika 28: Obvestilo o potrebnem premiku konfiguracijske datoteke.

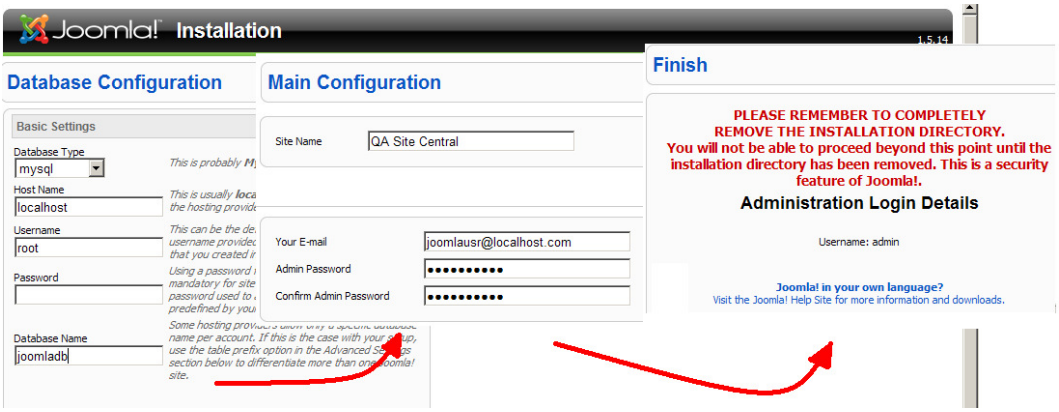
3.3.2.6 Namestitev orodja Joomla!

Joomla! arhivsko datoteko razširimo, po potrebi preimenujemo in premaknemo v imenik spletnega strežnika `<xampp_dir>\htdocs`. Slika 29 prikazuje končno drevesno strukturo spletnega strežnika.



Slika 29: Podoba imenika spletnega strežnika `<xampp_dir>\htdocs`.

Med namestitvijo določimo ime podatkovne baze ter opredelimo uporabniške račune za dostop le-teh, kot tudi za prijavo v sistem Joomla! (slika 30). V skladu s prejšnjimi poimenovanji podatkovih baz se odločimo za naziv `joomladb`.



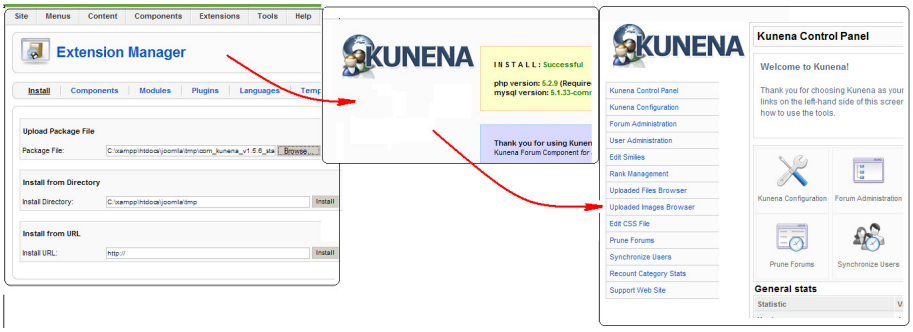
Slika 30: Namestitveni koraki sistema Joomla!.

Po končani namestitvi je potrebno še pred prvo prijavo v sistem odstraniti imenik `<joomla_dir>\installation\` iz korenskega imenika `<joomla_dir>\`. Za prvo prijavo uporabimo kar uporabniški račun ter geslo `admin/admin`..

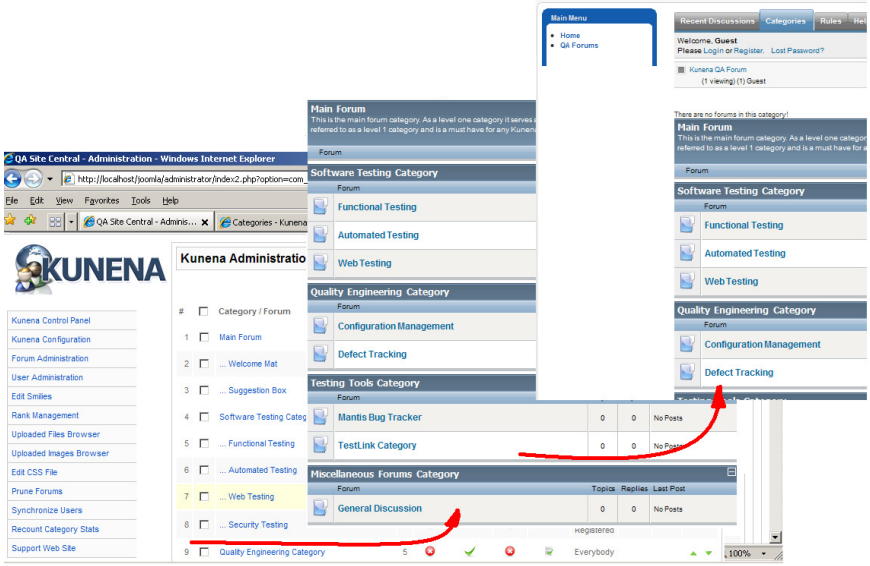
3.3.2.7 Namestitev foruma Kunena

Kunena forum pride v obliki razširitve (ang. extension) [14], ki se namesti znotraj sistema Joomla! (slika 31). Razširitev namestimo preko administratorskega vmesnika. Rezultat je lepo

povezana rešitev, za katero ne potrebujemo dodatnih vtičnikov (ang. plugins), da bi dosegli združljivost s sistemom Joomla! (slika 32).



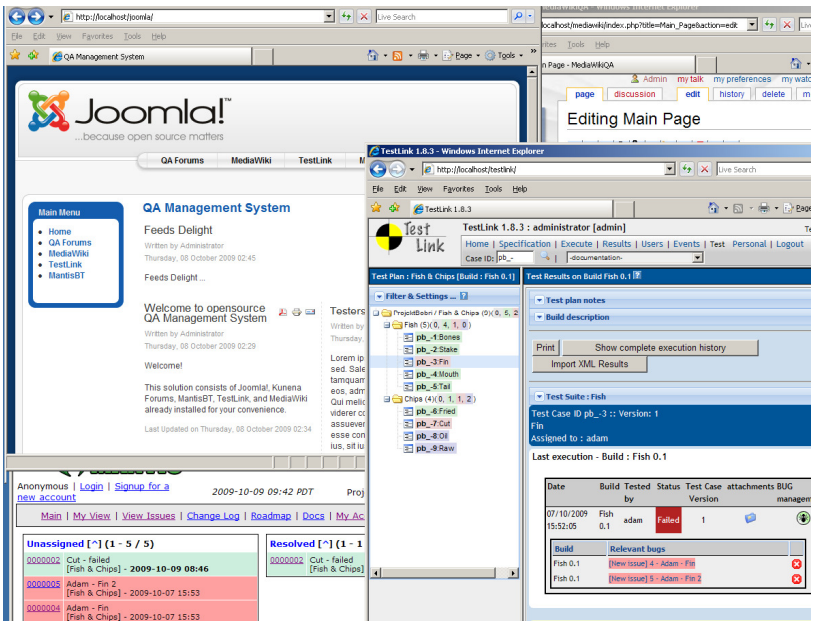
Slika 31: Postopek amestitve razširitve Kunena v sistemu Joomla!.



Slika 32: Združitev foruma Kunena s sistemom Joomla!.

3.3.3 Podoba okolja pred postopkom združitve

Okolje je v tem koraku pripravljeno za opravljanje dela. Toda tukaj je končana šele prva faza postavitve našega okolja. Trenutno se orodja vedejo samostojno, vsako zase, torej brez neke posebne povezave. S tem, ko imamo odprtih več oken, nam sistem ne nudi neke urejenosti in preglednosti (slika 33).



Slika 33: Prikaz nameščenih med seboj nepovezanih samostojnih orodij.

Zato je zagotovo eden bolj smiselnih naslednjih korakov, da vse sisteme združimo v neko celoto.

Slika 34: Podoba foruma po namestitvi znotraj sistema Joomla!.

3.4 Postopek združitve posameznih orodij

Sistemi kot taki so nameščeni ter pripravljeni za uporabo. Vendar, ker si želimo ustvariti centralno mesto, od koder bomo lahko dostopali do vseh orodij z enega mesta (slika 34), je potrebno opraviti še nekaj sprememb v posameznih orodjih. Za centralno mesto, ki bo združevalo vsa naša orodja, smo si izbrali sistem Joomla!.

3.4.1 Združitev orodij MantisBT in TestLink

Za uspešno združitev sledilca napak MantisBT in orodja za upravljanje s testnimi primeri TestLink moramo izvesti naslednje tri korake [6].

Korak 1. Nastavitev sledilca napak MantisBT

Za uspešno združitev orodij je potrebno v sledilcu napak MantisBT dodati novega uporabnika z vlogo gledalca (ang. viewer) – `anonymous_mantis_user, nato pa` mu omogočiti anonimno prijavo v orodje MantisBT.

Da bi omogočili anonimno prijavo v orodje MantisBT, je potrebno izvesti spremembe na dveh parametrih v konfiguracijski datoteki `<mantis_dir>/config_inc.php`.
Parameter z imenom `$g_allow_anonymous_login` je potrebno nastaviti na vrednost `'ON'`, parameter z imenom `$anonymous_account` pa nastavimo na uporabniško ime, ki smo ga ravnokar ustvarili za anonimni dostop do aplikacije. Izvedene spremembe so prikazane v sliki 35.

```
# --- anonymous login ---------
# Allow anonymous login
$g_allow_anonymous_login = ON;
$g_anonymous_account = 'anonymous_mantis_user';
```

Slika 35: Prikaz potrebnih nastavitev znotraj datoteke `config_inc.php`.

Korak 2. Nastavitev orodja za upravljanje s testnimi primeri TestLink

V imeniku aplikacije TestLink je potrebno nastaviti spremembe v vmesniku `mantis.cfg.php`, ki se nahaja v imeniku `<testlink_dir>/cfg/`.

V datoteki je potrebno nastaviti naslednje konstante:

- `BUG_TRACK_DB_HOST` – naziv podatkovnega strežnika, kjer se nahaja podatkovna baza MantisBT,
- `BUG_TRACK_DB_NAME` – naziv podatkovne baze,
- `BUG_TRACK_DB_TYPE` – naziv tipa podatkovne baze,
- `BUG_TRACK_DB_USER` – uporabniško ime za dostop do podatkovne baze,
- `BUG_TRACK_DB_PASS` – uporabniško geslo za dostop do podatkovne baze,
- `BUG_TRACK_HREF` – povezava URL za pregled napak v sistemu MantisBT in

- `BUG_TRACK_ENTER_BUG_HREF` – povezava URL za vnos novih napak v orodje MantisBT.

Podoba nastavljene datoteke je prikazan na sliki 36.

```
/** The DB host to use when connecting to the mantis db */
define('BUG_TRACK_DB_HOST', 'myserver');
/** The name of the database that contains the mantis tables */
define('BUG_TRACK_DB_NAME', 'mantisbtdb');
/** The DB type being used by mantis values: mysql,mssql,postgres */
define('BUG_TRACK_DB_TYPE', 'mysql');
/** The DB user to use for connecting to the mantis db */
define('BUG_TRACK_DB_USER', 'mantisbtusr');
/**The DB password to use for connecting to the mantisdb */
define('BUG_TRACK_DB_PASS','mantisbtpass');
/** link to the bugtracking system, for viewing bugs */
define('BUG_TRACK_HREF', "http://myserver/mantisbt/view.php?id=");
/** link to the bugtracking system, for entering new bugs */
define('BUG_TRACK_ENTER_BUG_HREF',"http://myserver/mantisbt/");
```

Slika 36: Prikaz potrebnih sprememb znotraj datoteke `mantis.cfg.php`.

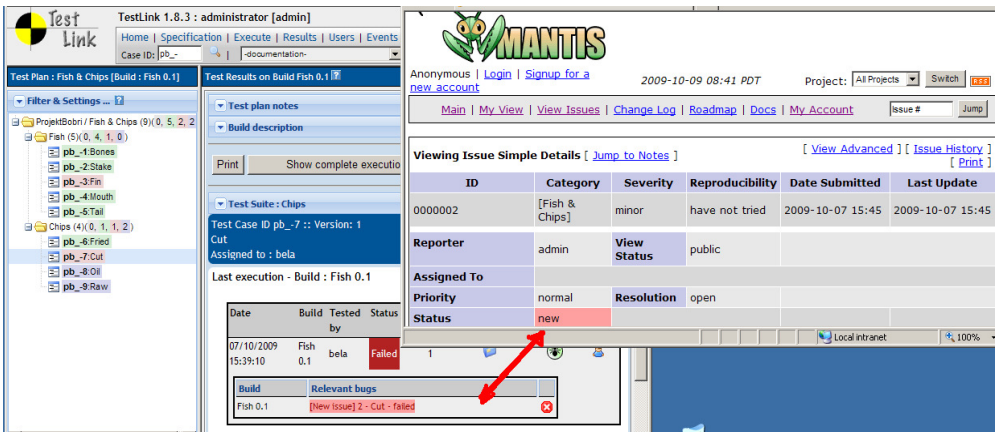
Korak 3. Omogočanje združitve s sledilcem napak MantisBT

V zadnjem koraku vnesemo, oziroma spremenimo vrednost spremenljivke `$g_interface_bugs` iz privzete vrednosti `'NO'` na vrednost `'MANTIS'`, v nastavitveni datoteki `<testlink_dir>/costum_config.inc.php`, orodja TestLink (slika 37). Končni rezultat je prikazan na sliki 38; in s tem je tudi postopek združevanja zaključen.

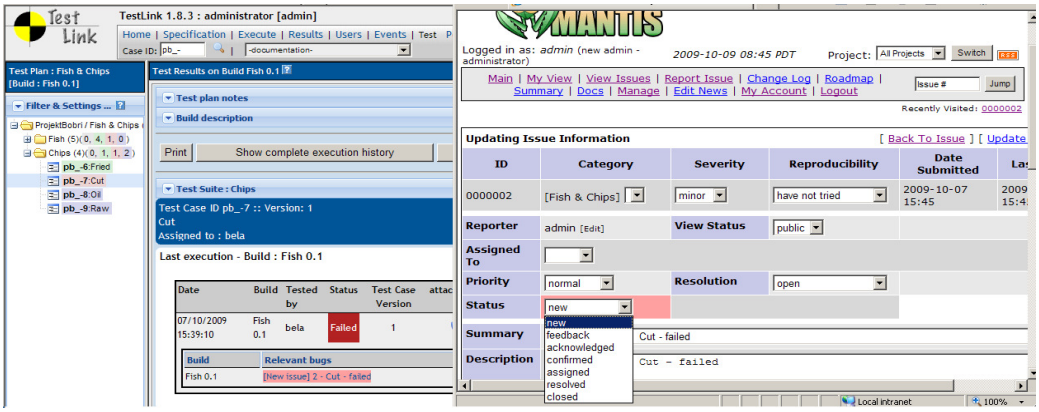
```
$g_interface_bugs = 'MANTIS';
```

Slika 37: Prikaz potrebnih sprememb znotraj datoteke `costum_config.inc.php`.

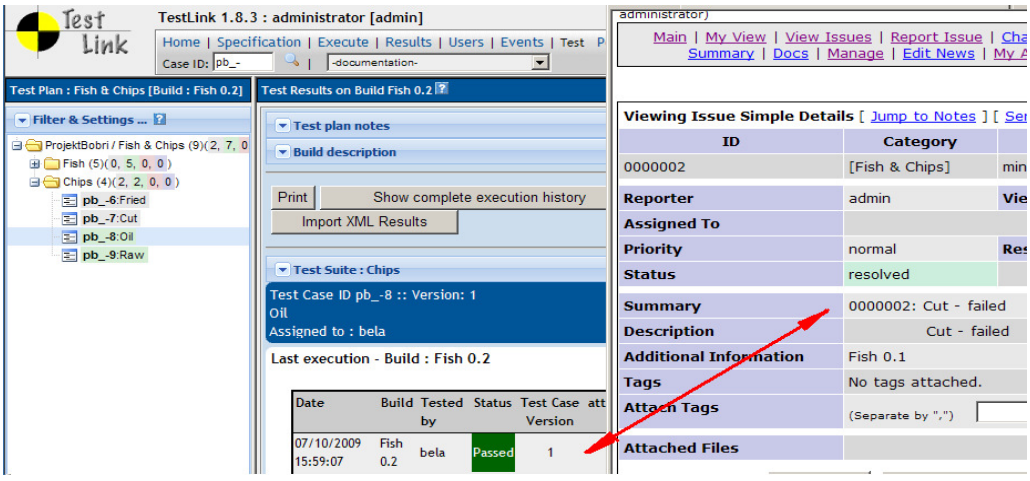
Slika 38 prikazuje uspešno povezavo med sledilcem napak in orodjem za upravljanje s testnimi primeri, ki se odraža s poenoteno predstavitvijo statusa prijavljene napake. Tako smo poskrbeli, da bo vsaka nadaljnja sprememba stanja napake v sledilcu napak (slika 39) razvidna tudi v orodju za upravljanje s testnimi primeri (slika 40).



Slika 38: Uspešna povezava med TestLink in MantisBT.



Slika 39: Sprememba stanja napake znotraj MantisBT.



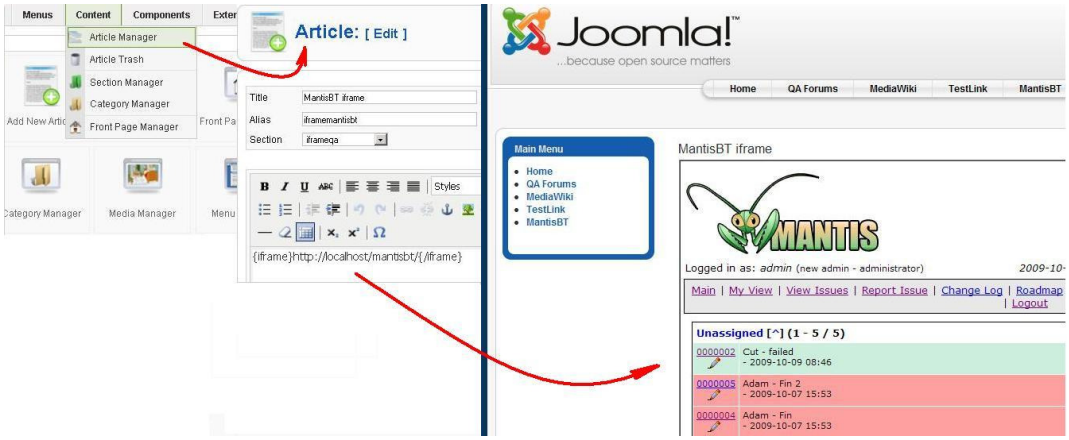
Slika 40: Odraz na spremembo stanja napake znotraj TestLink.

Vzpostavljena povezava med orodji pripomore k lažjemu sledenju napak ter ugotavljanja kdaj, oziroma v kateri verziji in v katerem testnem ciklu so se te pojavile.

3.4.2 Združitev s sistemom Joomla!

Ko smo uspešno opravili združitev med orodjem za upravljanje s testnimi primeri ter sledilcem napak, se lahko lotimo združitve preostalih orodij. Kot smo že omenili pri izbiri orodij, je Joomla! zelo prilagodljiv sistem, saj zanj obstaja mnogo razširitev, vtičnikov in modulov. Za izvedbo združitve smo uporabili vtičnik `mosiframe` [18], ki omogoča, da se na strani, oziroma v članku sistema Joomla! lahko vstavi ter uporablja značke `<iframe>`.

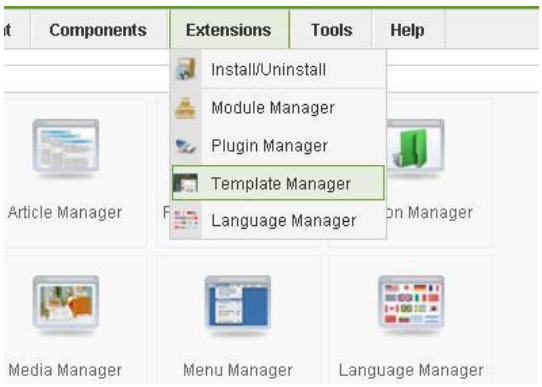
Značke `<iframe>` nam omogočajo prikaz vsebine drugih spletnih strani znotraj naše strani. V članek vstavimo sklic na vtičnik (opcijsko lahko nastavimo tudi atributa za višino in širino notranjega okna) ter podamo željeno povezavo, na primer do sledilca napak MantisBT je sledeča: `{iframe}http://localhost/mantisbt{/iframe}` (slika 41).



Slika 41: Potek namestitve vtičnika *mosiframe*.

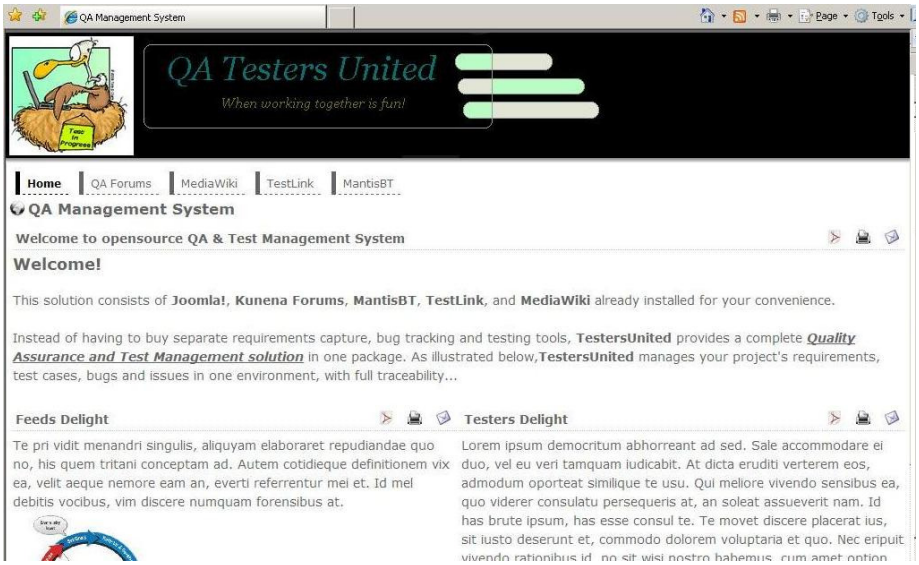
3.4.3 Oblikovanje spletne strani

Oblikovanje spletne strani v sistemu Joomla! zahteva poseg v predhodno nameščeno stilsko predlogo posamezne teme (ang. template). Stilsko datoteko namestimo preko vmesnika pod opcijo `Extensions`→`Install/Uninstall` (slika 41). V našem primeru smo uporabili arhivsko datoteko predloge `August` [18].



Slika 42: Opcije `Install/Uninstall` in `Template Manager`.

Po namestitvi se preko vmesnika pod opcijo `Extensions`→`Template Manager` opravi samo urejanje stilskih predlog CSS [17]. Ko smo zadovoljni s predlogo, potrdimo spremembe ter odpremo našo začetno stran in preverimo rezultat urejanja (slika 43).



Slika 43: Glavna spletna stran po izvedbi stilskih sprememb v stilski predlogi `August`.

Slika 44: Podoba prijavne strani z uporabo `mosiframe` vtičnika.

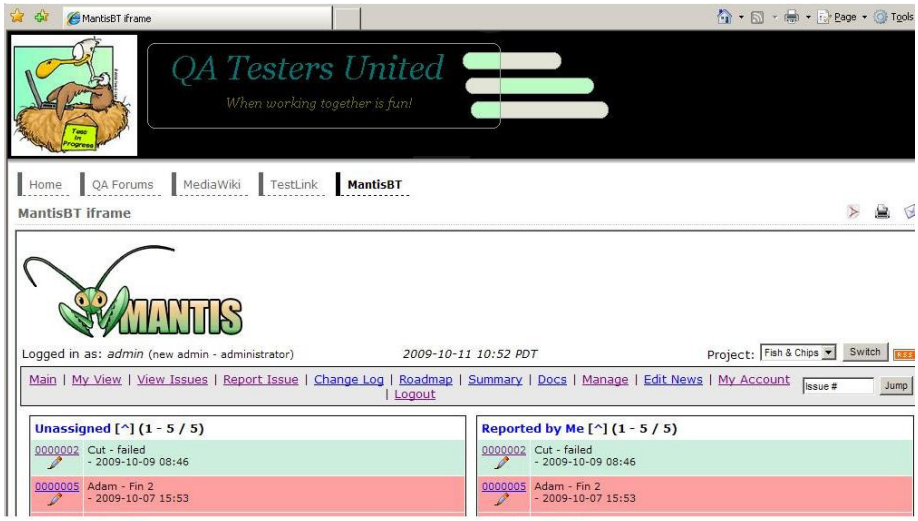
4. Predstavitev končne rešitve

Cilj diplomskega dela je bil prikazati način, kako se lahko z uporabo različnih odprtokodnih rešitev sestavi prilagojeno celovito okolje za izvedbo testiranja programske opreme.

4.1 Kratka predstavitev sistema

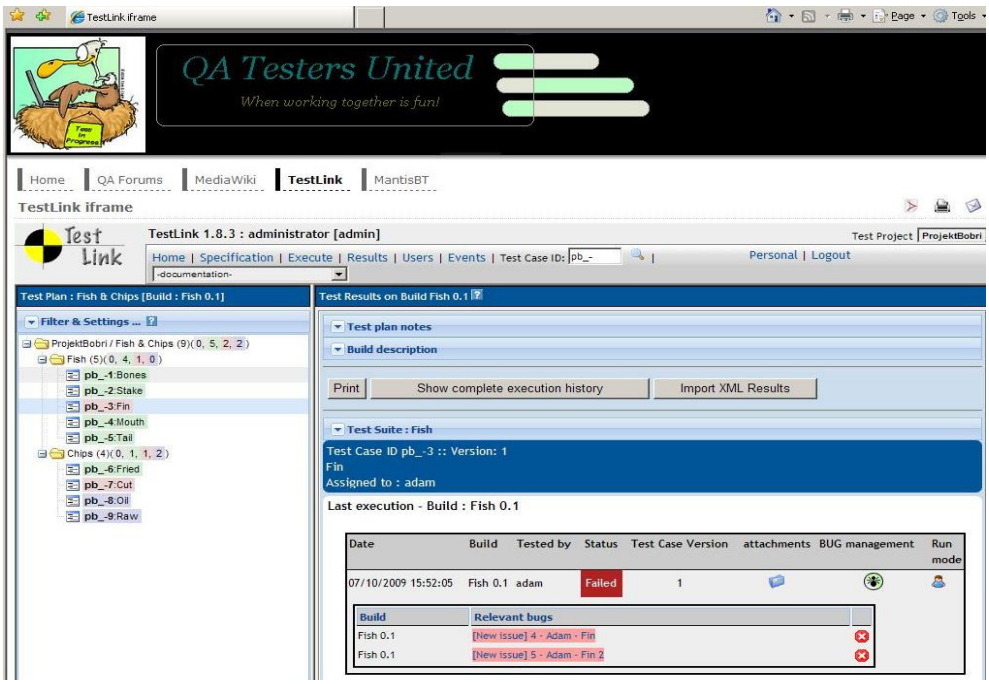
Ko je okolje nameščeno, lahko pričnemo z njegovo uporabo. Odpravimo se na domačo stran sistema Joomla!, kjer se odpre *osnovna stran našega sistema* (slika 43). Poleg povezav do orodij, ki so dostopna preko opcij zgornjega menija (povezave smo dodali med postopkom združitve orodij), se na osnovni strani lahko nahajajo tudi različne vsebine, povezane s testiranji in projekti, kot so: novice, novosti, časovni roki za določene projekte, uspehi, opravila, zahteve aplikacij in drugo, kar je trenutno v testiranju.

Z izbiro opcije `MantisBT`, ki jo izberemo v zgornjem meniju, dostopamo do sledilca napak MantisBT, ki se nam prikaže v glavnem oknu z izpisom trenutnega stanja najdenih napak (slika 45). V njem se lahko vnašajo nove napake, kot tudi spreminjajo, posodabljajo ali brišejo stare napake.



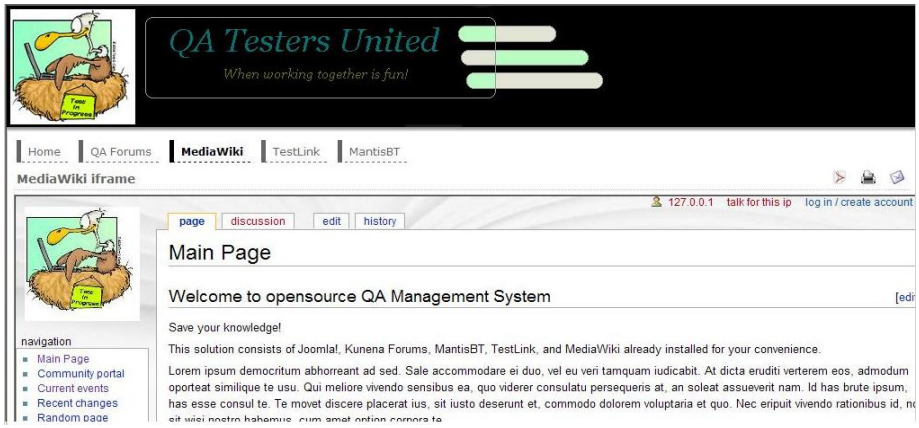
Slika 45: Okolje MantisBT znotraj sistema Joomla!.

Z izbiro opcije `TestLink` dobimo dostop do orodja za upravljanje s testnimi primeri, kjer lahko dodajamo testne načrte, testne primere, dodeljujemo stanje odkritih napak, izdelavo testnih poročil in podobno. Ker smo opravili združitev z orodjem MantisBT, smo na tak način zagotovili, da se stanje napake v MantisBT posodobi tudi pri testnem primeru, ki je povezan z isto identifikacijsko številko in se nahaja v orodju TestLink (slika 46).



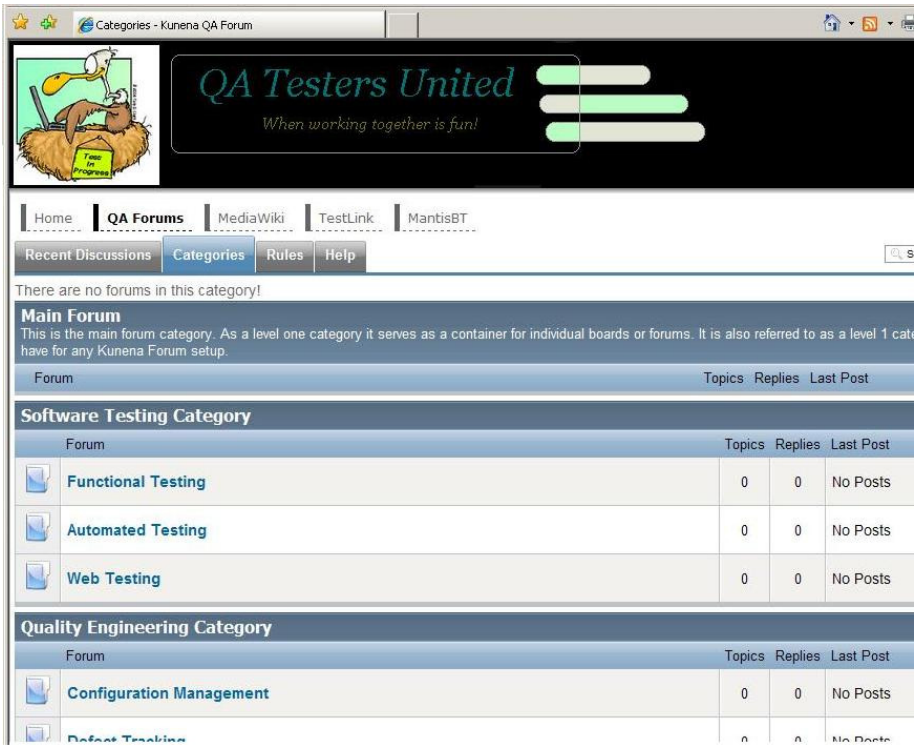
Slika 46: Okolje TestLink znotraj sistema Joomla!.

Izbira opcije `MediaWiki` nam prikaže MediaWiki, ki predstavlja naš sistem za upravljanje z znanjem (ang. knowledge managment) (slika 47). V njej se lahko hrani članke s področja testiranja, opis postopkov testiranja, opis testnih okolij, zahteve za izvajanje programske in strojne opreme v testiranju, opis namestitve sistemov v testiranju, navodila za nove člane v skupini in podobno.



Slika 47: Pozdravna stran sistema MediaWiki znotraj sistema Joomla!

Pod opcijo `QA Forums` pa je dosegljiv forum, kjer si lahko člani izmenjujejo mnenja, vprašanja in odgovore, posredujejo zamisli in podobno (slika 48).



Slika 48: Prikaz foruma Kunena znotraj sistema Joomla!

4.2 Namen

Namen sistema je zagotoviti takojšnjo pripravljenost delovnega okolja takoj zatem, ko je le-ta uspešno nameščen. To je tudi eden poglavitnih razlogov za uporabo spletnih aplikaciji pri namestitvi okolja. S tem omogočimo testnim članom ter upraviteljem, da ga takoj pričnejo uporabljati.

V praksi se dogaja, da se na začetku projekta izgubi veliko časa s postavitvijo in prilagajanjem delovnega okolja: torej z nameščanjem potrebnih namiznih aplikacij, nastavitvijo povezav do sistema za upravljanje s podatkovnimi bazami, sinhronizacijo namiznih aplikacij, dostopanjem do baze znanja, skladišča zahtev, dokumentacije in podobnega.

Okolje omogoča smotrno izrabo časa pri vpeljavi novih članov v skupino, saj se z uporabo spletnih rešitev izognemo izgubi dneva ali dveh, ki jih drugače potrebuje novi član, da si namesti delovno okolje. Ta za svoje delo sedaj potrebuje pravzaprav samo spletni brskalnik.

Forum in sistem wiki, kot posrednika znanj, postaneta dostopna vsakomur, ki želi sodelovati pri diskusiji (izmenjava mnenj, stališč in misli). Vsakdo lahko poda svoje predloge, članke, prispevke, pripombe in na tak način prispeva k izboljšanju kakovosti in storilnosti dela v skupini, oziroma na projektu. To vodi tudi do povečanja sodelovanja med člani, saj se tako razvija boljši duh v skupini (ang. team spirit).

4.3 Prednosti

Ena izmed poglavitnejših prednosti tega sistema je njegova **prenosljivost**, saj se z relativno malo truda prenese celoten sistem na drugo platformo. Ker je rešitev sestavljena iz nabora več spletnih aplikacij, je tudi **dostopnost** do nje, in s tem tudi uporaba ter izvajanje, neodvisna od uporabnikovega operacijskega sistema. Vse, kar potrebujemo za dostop, je spletni brskalnik.

S **časovnega vidika**, oziroma pri uvajanju novih članov omogoča okolje takojšnjo vpeljavo v projekt, brez potrebnih prilagajanj ter postavitve delovnega okolja. Ko je okolje enkrat v obratovanju, se nove testne skupine izognejo izgubi časa, ki je potreben za namestitev delovnega okolja. To je še posebej koristno v večjih podjetjih, ki izvajajo več vzporednih testnih projektov za različne naročnike.

Z **vrednostnega vidika** je najbolj očitno to, da je rešitev prosto dostopna vsakomur, ki bi jo potreboval pri svojem delu. Še zlasti pride v poštev uporabnikom, ki nimajo potrebnih sredstev za nakup komercialne različice, ali pa bi sredstva raje vložili v kak drug del projekta.

Ker je celotna rešitev sestavljena iz odprtokodnih sistemov, se lahko njihove izboljšave bolj **enostavno nadgrajujejo in razširjajo**. Pogosto se dogaja, da zamišljena nadgradnja ali razširitev že obstaja ali je v procesu izdelave. Seveda lahko pri tem tudi sami sodelujemo ter tako prispevamo k odprtokodni skupnosti. Te nadgradnje in razširitve lahko nato vključimo v sistem preko raznih modulov, razširitev ter vtičnikov.

S povezavo sledilca napak MantisBT in sistemom za upravljanje s testnimi primeri TestLink se izognemo tudi možni izgubi že odkritih okvar, katerih sledenje na bolj obsežnih projektih lahko postane dokaj težavno. Lahko se namreč pripeti, da med ročnim prenosom identifikacijske številke za odkrito napako iz ene aplikacije pride do napačnega vnosa v drugi aplikaciji ter tako celo izgubimo sled za prvotno odkrito napako.

Spletna aplikacija se nahaja na samo eni lokaciji, kar pripomore k zmanjšanju stroškov vzdrževanja, odpravljanja napak ter hrambe podatkov.

4.4 Slabosti

Slabosti pri našem sistemu so gotovo razširitve in prilagoditve okolja. To velja v primeru, ko ne najdemo nadgradnje v obliki vtičnikov, razširitev ali modulov, s katerimi bi lahko zadovoljili naše potrebe. V takih primerih potrebujemo za nadgradnjo našega sistema napredno poznavanje programskega jezika PHP kot tudi poznavanje notranjega delovanja sistema, ki bi ga radi razširili ali prilagodili. Čeprav je recimo Joomla! uporabniku prijazna, se je treba za resnejše spremembe najprej poslužiti ustrezne literature.To pa seveda ni delo testnih skupin, ker imajo drugo delo.

Urejanje spletnih predlog za prilagajanje podobe spletne strani v sistemu Joomla! zahteva osnovno znanje označevalnega jezika HTML, kot tudi znanje za izdelavo in uporabo kaskadnih slogovnih predlog CSS.

5. Sklepne ugotovitve

Cilj diplomskega dela je bil opisati postopek postavitve odprtokodnega okolja za celovito izvedbo testiranja programske opreme. Prikazana rešitev je lahko dober nadomestek za nekatera draga komercialna orodja, ki so danes na tržišču. Tukaj bi lahko omenili samo sistem SpiraTest [20], izdelek podjetja Inflectra, ki je nekoliko podoben našemu okolju.

Namen diplomske naloge je prikazati odprtokodno rešitev, s katero se lahko izognemo marsikateri preglavici med procesom testiranja in posredno zagotavlja kakovost pri testiranju programske opreme. Z združitvijo vseh orodij smo želeli prikazati, kako si lahko na enostaven način zagotovimo, da so vse informacije, spremembe, dogovori, obvestila, napake, okvare, zahteve in podobno, ki so potrebne za uspešno in kakovostno delo na področju testiranja in čez celoten razvoj programske opreme na razpolago na enem centralnem mestu. Tako se zagotovi, da vsi člani vidijo najnovejše spremembe, obvestila, novosti na projektu, saj so te prikazane na prvi strani sistema.

Možnih izboljšav celotnega sistema je veliko in bi lahko rekli, da so omejene zgolj s potrebami na posameznem projektu. Nekaj možnih izboljšav, oziroma razširitev je recimo:

- enkratna prijava v celotni sistem za vse podsisteme,
- dinamičen prikaz novo prispele vsebine e-sporočila, pridobljenega po elektronski pošti,
- prikaz nazadnje vnešenih napak v sledilcu napak na začetni strani,
- prikaz najnovejših člankov, vnešenih v Wiki sistem,
- dodajanje in urejanje Wiki vsebin preko vmesnika WYSIWYG za urejanje vsebin sistema Joomla! in,
- vpeljava skupinskega pogovora (ang. chat) s pomočjo programa za neposredni pogovor med prijavljenimi.

Dodatek A

Seznam slik

6. Literatura

[1] D.Phillips, *The Software Project Manager's Handbook: Principles That Work at Work 2nd Edition*, John Wiley & Sons, 2007.

[2] F.Solina, *Projektno vodenje razvoja programske opreme*, Založba FE in FRI, 1997.

[3] I.Burnstein, *Practical Software Testing*, New York: Springer-Verlag, 2003.

[4] (2009) EMRIS - Enotna metodologija razvoja informacijskih sistemov. Dostopno na: http://www2.gov.si/mju/emris.nsf

[5] (2009) TestLink. Installation & Configuration manual. Dostopno na: http://www.teamst.org/_tldoc/1.8/installation_manual.pdf

[6] (2009) TestLink. How to setup Bug tracking system in TL 1.6. Dostopno na: http://www.teamst.org/_tldoc/1.7/tl-bts-howto.pdf

[7] (2009) Installing MantisBT. Dostopno na: http://forums.techarena.in/tips-tweaks/1154097.htm

[8] (2009) MantisBT: Administration guide. Dostopno na: http://docs.mantisbt.org/master/en/administration_guide.html

[9] (2009) Sites using MediaWiki/Corporate.Dostopno na: http://www.mediawiki.org/wiki/Sites_using_MediaWiki/corporate

[10] (2009) Kunena Forum. Dostopno na: http://extensions.joomla.org/extensions/communication/forum/7256

[11] (2009) MantisBT. Dostopno na: http://www.mantisbt.org/

[12] (2009) TestLink. Dostopno na:http://www.teamst.org/

[13] (2009) Joomla. Dostopno na: http://www.joomla.org/

[14] (2009) Joomla! Extensions directory. Dostopno na: http://extensions.joomla.org/

[15] (2009) MediaWiki. Dostopno na: http://www.mediawiki.org/wiki/MediaWiki

[16] (2009) XAMPP. Dostopno na: http://www.apachefriends.org/en/xampp.html

[17] (2009) Webgau.de - Template August. Dostopno na: http://www.webgau.de/joomla-templates/20-template-august.html

[18] (2009) Mosiframe. Dostopno na: http://extensions.joomla.org/extensions/style-a-design/popups-a-iframes/4144
[

19] (2009) Projektni management pri razvoju programskih rešitev. Dostopno na: http://www.cek.ef.uni-lj.si/magister/kampus60.pdf

[20] (2009) Inflectra – SpiraTest. Dostopno na: https://www.inflectra.com/SpiraTest/Default.aspx