

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Jerič

OGRODJA ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Ljubljana, 2012

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Jerič

OGRODJA ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2012



Št. naloge: 00200/2012

Datum: 02.03.2012

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

Kandidat: **MATEJ JERIČ**

Naslov: **OGRODJA ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI**
FRAMEWORK FOR AUTOMATIC EXECUTION OF TESTING
ACTIVITIES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Programska oprema se nahaja povsod okoli nas. Zaradi tako pomembne vloge v naših življenjih, se pojavljajo zahteve po čim krajših razvojnih ciklih in visoki kakovosti izdelane programske opreme. Za doseg te ciljev je med razvojem nujna izvedba ustreznega postopka testiranja. Poleg ročnega testiranja se vedno bolj uveljavlja avtomatsko testiranje in uporaba različnih orodij, ki to omogočajo. V okviru diplomskega dela predstavite različne tehnike testiranja, kjer se še posebej osredotočite na avtomatsko izvajanje testnih scenarijev. Predstavite funkcionalnosti in delovanje ogrodja za avtomatsko izvajanje testnih aktivnosti, ki ga uporabljate v podjetju ter predlagajte ustrezne izboljšave, ki izhajajo iz strateških usmeritev podjetja.

Mentor:

viš. pred. dr. Aljaž Zrnec

Dekan:

prof. dr. Nikolaj Zimic



IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani **Matej Jerič**

z vpisno številko **63060438**

sem avtor diplomskega dela z naslovom:

Ogrodja za avtomatsko izvajanje testnih aktivnosti.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Aljaža Zrnca,
- so elektronska oblika diplomskega dela, naslov (naslov v slovenščini in angleščini), povzetek (naslov povzetka v slovenščini in angleščini) ter ključne besede (slovenske in angleške) identični s tiskano obliko diplomskega dela in da
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne _____ Podpis avtorja: _____

ZAHVALA

Zahvaljujem se mentorju, viš. pred. dr. Aljažu Zrncu, za konkretne napotke, dobro usmerjanje in veliko potrpljenja pri izdelavi diplomske naloge.

Zahvaljujem se ženi Barbari za pomoč in podporo, sinu Lovru za popestritev trenutkov med pisanjem diplomske naloge in prijatelju Urbanu za nasvete in pomoč.

KAZALO

1. UVOD	3
2. TESTIRANJE NA SPLOŠNO	6
2.1 Metode testiranja	6
2.2 Nivoji testiranja.....	7
2.3 Življenjski cikel testiranja.....	9
3. AVTOMATSKO TESTIRANJE.....	13
3.1 Pojav avtomatskega testiranja	13
3.2. Kdaj, kako in koliko avtomatizirati.....	14
3.3 Orodja za funkcionalno testiranje	16
3.4 Zakaj avtomatsko testiranje	17
3.5 Cilji avtomatskega testiranja	18
3.6 ATLM (Automated Test Lifecycle Methodology).....	19
3.7 Primerjava ročnega in avtomatskega testiranja.....	23
4. IZZIVI NA PODROČJU TESTIRANJA	25
4.1 Goljufije pri avtomatizaciji testiranja	25
4.2 Odpravimo klasični argument za avtomatizacijo testiranja	25
4.3 Smotrni pristop k avtomatizaciji.....	30
5. ORODJA ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI	31
5.1 Microsoft Test Manager 2010	31
5.2 Software Testing Automation Framework (STAF).....	32
6. INTERNO OGRODJE ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI	34
6.1 Arhitektura in namestitve.....	34
6.2 Opis poteka procesa testiranja.....	39
6.3 Predlog izboljšave	40
7. SKLEPNE UGOTOVITVE	42
8. SEZNAM UPORABLJENIH VIROV	44

SEZNAM UPORABLJENIH PREGLEDNIC

Preglednica 1: Najpogostejše pregledani vidiki kakovosti programske opreme	4
Preglednica 2: Prikaz zmanjšanja porabe časa pri uvedbi avtomatskega testiranja.....	5
Preglednica 3: Podpora razvojnim ciklom z določenim tipom orodij	17

SEZNAM UPORABLJENIH SLIK

Slika 1: Faze razvojnega procesa, kot ga določa EMRIS	9
Slika 2: Življenjski cikel testiranja skozi faze razvojnega procesa IS	10
Slika 3: Prikaz metodologije življenjskega cikla testiranja (ATLM)	20
Slika 4: Arhitektura OAT	34
Slika 5: Drevesna struktura imenikov OAT	35
Slika 6: Primer sistemske nastavitvene datoteke	37
Slika 7: Primer datoteke s testnim naborom	38
Slika 8: Procesni model poteka testiranja	40
Slika 9: Procesni model poteka testiranja z upoštevanjem predlogov izboljšav.....	41

SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

ALM	Application Lifecycle Management
ATLM	Automated Test Lifecycle Methodology
EMRIS	Enotna metodologija razvoja informacijskih sistemov
IS	Informacijski sistem
OAT	Ogrodje za avtomatsko testiranje

POVZETEK

V diplomskem delu sem opisal ključne lastnosti različnih vrst testiranja ter primerjal ročno in avtomatsko testiranje. Predstavil sem značilnosti ogrodja, ki sem ga uporabljal pri svojem delu, in zanj predlagal izboljšave.

Uvod predstavlja najbolj bistveno spoznanje IT podjetij: da je testiranje ključnega pomena pri razvoju njihovega izdelka in da mu pri načrtovanju ne gre posvečati premajhne pozornosti.

V drugem poglavju sem se osredotočil na lastnosti testiranja ter predstavil različne metode in nivoje testiranja, kot tudi življenjski cikel testiranja.

V tretjem poglavju sem se osredotočil na avtomatsko testiranje, njegovo razširjenost in postopek sprejemanja odločitve za uvedbo avtomatskega testiranja, razloge in cilje tega ter na koncu tudi na konkretnjšo primerjavo z ročnim testiranjem.

V četrtem poglavju sem predstavil izzive, težave in ovire, ki jih testiranje prinaša.

V petem poglavju se natančneje ukvarjam z orodji za avtomatsko testiranje ter predstavim Microsoftovo orodje Test Manager in IBM-ovo odprtokodno ogrodje STAF.

V šestem poglavju sem predstavil ogrodje za avtomatsko izvajanje testnih aktivnosti, ki smo ga uporabljali v podjetju, kjer sem bil zaposlen. Vključil sem tudi predloge za izboljšavo delovanja ogrodja.

Če zelo strnem misli iz diplomske naloge, je njeno ključno sporočilo, da morajo podjetja testiranju namenjati ustrezno pozornost, primerljivo s tisto, ki je je deležen sam razvoj programske opreme. Avtomatskega testiranja ne gre jemati za samoumevnega, kot tudi ni smotno vseh upov polagati v avtomatizacijo, temveč je treba v čim več primerih uporabiti kombinacijo z ročnim testiranjem.

Ključne besede: programska oprema, avtomatizacija testiranja, testni scenarij, metode testiranja, testiranje programske opreme, testno orodje

ABSTRACT

The thesis describes the key features of different kinds of testing and the comparison of the manual with the automated testing. It also depicts the characteristics of the framework I was using at work and some improvements for it.

The introduction begins with the most important observation of the IT companies: testing has become of significant importance in the development of their product and considerable attention should be paid to it during the planning of the development.

The second chapter focuses on the testing characteristics and presents different methods and levels of testing, as well as the lifecycle of testing.

The third chapter focuses on the automated testing, its widely spread use and the process of making a decision for implementing the automated testing. It also presents its reasons and aims and concludes with the comparison with the manual testing.

In the fourth chapter, the challenges, difficulties and obstacles in testing are described.

In the fifth chapter, I thoroughly dealt with the frameworks for the automatic testing activities and described Microsoft's tool Test Manager and IBM's open-source STAF.

In the sixth chapter, the framework for the automatic execution of testing activities we were using in the company of my former employment is described. Some suggestions for the improvement of the framework are also there.

If I were to summarize the main ideas and thoughts from the thesis, I would say its main message is that companies should pay adequate attention to testing, comparable to that devoted to the sole software development. Automated testing should not be taken for granted as well as not considered to be the silver bullet. Instead, in as many cases as possible, it should be used in combination with manual testing.

Keywords: software, testing automation, test case, testing methods, software testing, test tool

1. UVOD

Testiranje programske opreme je ena izmed najdražjih, vendar obenem najpomembnejših aktivnosti, ki jih izvajamo v fazi razvoja programske opreme. Učinkovito testiranje lahko zmanjša skupne stroške razvoja, ekipi pa omogoči bolj objektiven pogled na kakovost programske opreme (bolje pozna njegove prednosti in slabosti), ki ga razvija.

Napačno bi bilo testiranje razumeti zgolj kot razhroščevanje, saj je njegovih ciljev več. Glavni so:

- zagotavljanje kakovosti,
- dokazovanje in potrjevanje pričakovanega delovanja,
- preverjanje delovanja v skladu z zahtevami in
- ocenjevanje zanesljivosti.

Zagotavljanje kakovosti je v svetu široke uporabe računalnikov izjemnega pomena, saj so posledice napak lahko zelo hude. Povzročijo lahko enormne izgube, letalske nesreče, ponesrečene vesoljske polete, zastoje na borzah in še kaj. Brez pretiravanja lahko rečemo, da lahko napake v nekaterih primerih odločajo med življenjem in smrtjo. Seveda pa o kakovosti ne govorimo zgolj z vidika preživetja, temveč predvsem, da programska oprema deluje v skladu s potrebami naročnika. Nerealno bi bilo od razvijalca pričakovati, da bo že v prvem poskusu sestavil brezhiben program. Tudi on je le človek, ki pa mu testiranje in posledično odpravljanje napak pomagata izboljšati končni izdelek.

Pri dokazovanju in potrjevanju pričakovanega načina delovanja programske opreme pri testiranju dejansko uporabljamo merjenje. Na podlagi testnih rezultatov lahko testni inženirji pridejo do določenih sklepov, kakovost programske opreme pa primerjajo na podlagi testirane različne programske opreme z enakimi specifikacijami. Pogoji za to primerjavo je, da so bile testirane z enakimi testi.

Glede na to, da kakovost ni merljiva kot taka, za njeno preverjanje uporabljamo kriterije: funkcionalnost, inženirstvo in prilagodljivost.

O teh kriterijih lahko razmišljamo tudi kot o dimenzijah v programskem svetu kakovosti, vsako dimenzijo pa lahko razčlenimo na njene sestavne dele. Spodnja tabela prikazuje nekatere najpogostejše pregledane vidike kakovosti.

Funkcionalnost (zunanja kakovost)	Inženirstvo (notranja kakovost)	Prilagodljivost (kakovost v prihodnje)
Pravilnost	Učinkovitost	Fleksibilnost
Zanesljivost	Preverljivost	Ponovna uporabnost
Uporabnost	Dokumentacija	Vzdrževanje
Integriteta oz. neokrnjenost	Struktura	

Preglednica 1: Najpogostejše pregledani vidiki kakovosti programske opreme

V primeru dobrega testiranja pridobimo meritve za vse pomembne dejavnike, uteži posameznih dejavnikov pa so odvisne od vrste aplikacije. Pri vsakem sistemu, kjer gre za človeška življenja, mora biti izjemen poudarek na zanesljivosti in integriteti, v običajnem poslovnem sistemu sta najpomembnejši uporabnost in vzdrževanje, medtem ko za enkratni znanstveni program morda nobena izmed teh ne bo igrala ključne vloge. Testiranje mora torej biti prilagojeno preizkušanju ključnih dejavnikov posamezne aplikacije, saj bo s tem njena kakovost postala vidna in neoporečna.

Testi z namenom potrjevanja pravilnega delovanja programske opreme se imenujejo **pozitivni testi**. Njihova težava je, da lahko potrdijo delovanje programske opreme le v določenih testnih primerih. Končno število testov namreč ne more dati zagotovila, da oprema deluje v vseh situacijah. Prav nasprotno: en sam negativni test je dovolj za potrditev, da programska oprema ne deluje. **Negativni testi** so tisti, ki stremijo k zlomu programske opreme oziroma skušajo dokazati, da programska oprema ne deluje. Kljub temu pa mora programska oprema imeti takšne lastnosti, da uspešno prestande tudi ustrezno število teh testov. Ker je testiranje zahtevno ter zahteva veliko časa in sredstev, je snovanje testiranja prav tako pomembno v procesu razvijanja programske opreme.

Testiranje programske opreme predstavlja velik kompromis med proračunom, časom in kakovostjo. Dejstvo je, da napake, odkrite v zgodnjih fazah razvoja, povzročijo le zanemarljive stroške, in da je največ napak v programski opremi povezanih z arhitekturo (in torej ne z napakami, ki nastanejo med pisanjem kode). Napake bodo skoraj vedno obstajale v zapleteni programski opremi, vendar pa niso posledica nepazljivosti ali neodgovornosti razvijalcev, temveč zapletenosti programske opreme. Prav iz tega razloga ne moremo nikoli povsem izključiti arhitekturnih napak oziroma pomanjkljivosti in lahko sklepamo, da so napake, ki povzročajo najvišje stroške, povezane z arhitekturo.

Omejtvam navkljub testiranje predstavlja ključni del v razvoju programske opreme. Po podatkih ankete, ki jo je izvedlo podjetje Innovative Defence Technologies, približno polovica anketiranih podjetij za testiranje nameni od 50 do 75 % vsega časa, namenjenega razvoju nove programske opreme. [2] V istem viru zasledimo tudi podatek, da je testiranju običajno namenjenih kar 50 % sredstev proračuna.

Na podlagi druge raziskave, ki jo je izvedel Quality and Assurance Institute, pa so raziskovalci prišli do zaključka, da je razvoj orodij za avtomatsko testiranje v začetni fazi sicer zahteval znatne naložbe, vendar pa je v končni fazi poraba časa za testiranje do 75 % manjša. Podrobneje so rezultati njihove raziskave razvidni iz spodnje tabele. [2]

	Ročno testiranje v urah	Avtomatsko testiranje v urah	Izboljšanje v odstotkih
Razvoj testnega načrta	32	40	-25
Razvoj testnih primerov	262	117	55
Izvedba testiranja	466	23	95
Analiza rezultatov testa	117	58	50
Sledenje napakam	117	23	80
Ustvarjanje poročil	96	16	83
Celotne ure	1090	277	75

Preglednica 2: Prikaz zmanjšanja porabe časa pri uvedbi avtomatskega testiranja

Testiranje programske opreme še ni dokončno razvito. Danes še vedno uporabljamo enake testne metode kot pred 30 leti, čeprav nekatere izmed njih sploh niso dobre. Programsko testiranje je lahko drago, vendar je zanemarjanje testiranja, predvsem tam, kjer so v igri človeška življenja, lahko bistveno dražje.

2. TESTIRANJE NA SPLOŠNO

2.1 Metode testiranja

Poznamo naslednje metode testiranja:

- metoda bele skrinjice,
- metoda črne skrinjice,
- metoda sive skrinjice.

Metodi bele in črne skrinjice se dopolnjujeta, zato je za popolno testiranje programske opreme priporočljivo uporabiti obe.

2.1.1 Metoda bele skrinjice

Pri tej metodi imamo na voljo izvorno kodo, na osnovi katere načrtujemo testne scenarije. S to metodo testiramo posamezne module programske opreme. Osredotoča se na notranjo strukturo izvorne kode. Metoda se lahko uporabi v fazi testiranja modulov, integracije ali systemskega testiranja, vendar je običajno uporabljena v prvi fazi, t.j. testiranja modulov. Metoda bele skrinjice lahko pripomore k odkritju velikega števila napak, ne more pa odkriti neuvedenih delov specifikacij ali manjkajočih zahtev.

2.1.2 Metoda črne skrinjice

Z metodo črne skrinjice testiramo funkcionalnost programske opreme. Poznavanje notranjih struktur ali izvorne kode v splošnem ni potrebno. Pri testiranju s to metodo podajamo določene vhodne podatke, izhodne podatke pa nato primerjamo s pričakovanimi rezultati. Pri tem ne vemo nič o tem, kako je do rezultata prišlo.

Prednosti metode:

- testi so nepristranski, saj jih izvaja preizkuševalec,
- testiranje poteka z uporabnikovega vidika,
- znanje programskega jezika ni potrebno in
- testne scenarije lahko oblikujemo, takoj ko so dokončane specifikacije.

Slabosti metode:

- testi so lahko nepotrebni, celo odvečni, če jih je predhodno že izvedel razvijalec,
- izdelava testnih scenarijev je zahtevna in
- testiranje vseh možnih vhodnih podatkov je nerealno, saj bi to zahtevalo ogromno časa.

2.1.3 Metoda sive skrinjice

Metoda sive skrinjice je kombinacija metod bele in črne skrinjice. Notranja struktura kode je deloma znana, kar pomeni, da imamo dostop do notranje strukture in algoritmov za namen izdelave testnih scenarijev. Kljub temu testiranje izvajamo z uporabnikovega vidika. Metoda sive skrinjice se lahko uporablja v vseh stopnjah testiranja, vendar je običajna uporaba v fazi integracijskega testiranja.

2.2 Nivoji testiranja

Testiranje se dejansko prične že v fazi načrtovanja, kjer izdelamo načrte testnih scenarijev. Testirati začnemo na nivoju posameznih modulov in postopoma napredujemo k integraciji celotnega sistema. Kot referenca za testiranje med integracijo modulov služi načrt programske opreme. Za tem validacija na osnovi zahtev, zapisanih v specifikaciji, ugotavlja, ali jih sistem izpolnjuje. Končno sistemsko testiranje v simuliranih ali realnih razmerah preverja, ali razvita programska oprema pravilno deluje skupaj z drugimi sistemskimi sklopi. (Regresijsko testiranje je podrobneje opisano v podpoglavju 2.2.5.) Za testiranje posameznega modula je v prvi vrsti zadolžen razvijalec modula, pri velikih sistemih pa poznejše testiranje izvaja posebna neodvisna skupina.

2.2.1 Testiranje modulov

Testiranje modulov je že uveljavljen koncept pri razvoju programske opreme. S pojavom novih tehnologij in orodij, ki omogočajo avtomatizacijo, pa je ta vrsta testiranja stopila v ospredje.

Modul je najmanjša enota programske opreme, ki jo lahko testiramo. Posamezne module testiramo po načelu bele skrinjice. Kot pomoč pri testiranju modulov so uporabljena testna ogrodja modulov, gonilniki, navidezni moduli in lažni objekti. Testiranje običajno izvaja razvijalec.

Glavne prednosti pri uporabi te metode so:

- povečevanje zaupanja v spremembe in vzdrževanje programske kode,
- programsko kodo je enostavneje ponovno uporabiti,
- razvoj poteka hitreje,
- strošek odpravljanja programske napake v fazi testiranja modulov je veliko manjši v primerjavi s tistimi, odkritimi v poznejših fazah,
- razhroščevanje je enostavno in
- programska koda je veliko bolj zanesljiva.

2.2.2 Testiranje integracije

Testiranje integracije se izvaja po fazi testiranja modulov in pred sistemskim testiranjem. Posamezne module združujemo in jih testiramo kot nove komponente. Namen je odkriti napake pri komunikaciji med integriranimi moduli. Uporabimo lahko katero koli izmed metod testiranja.

Poznamo štiri glavne pristope k testiranju integracije:

- Big-bang,
- integracija od spodaj navzgor,
- integracija od zgoraj navzdol in
- kombinirani pristop.

2.2.3 Sistemsko testiranje

Sistemsko testiranje je del razvojnega cikla, ki ga izvajamo pred prevzemnim testiranjem. Poteka na dokončanem integriranem sistemu, pri katerem ugotavljamo skladnost programske opreme s specifikacijami – to je t.i. validacija sistema. Testiranje običajno poteka z uporabo črne skrinjice.

V tej fazi preverjamo funkcionalne in sistemske zahteve, kar smatramo za preiskovalno testno fazo. Ta vključuje destruktivno testiranje zasnove, obnašanje in pričakovanja naročnika. S tem preverjamo delovanje in komunikacijo integriranih delov sistema, ki morajo zadostiti funkcionalnim zahtevam.

V primerjavi z integracijskim testiranjem je tu število sistemskih testov manjše, kar je posledica odkritja napak v prejšnjih fazah testiranja. V skladu s tem so stroški odprave napak, ki jih odkrijemo šele v fazi sistema testiranja, bistveno višji.

K sistemskemu testiranju uvrščamo tudi testiranje okrevanja sistema, testiranje varnosti, testiranje obremenitve in testiranje občutljivosti.

“Pri testiranju okrevanja sistema programsko opremo na različne načine prisilimo, da izpade. Nato opazujemo, če sistem pravilno in hitro okreva, tako kot od njega zahtevamo, bodisi avtomatično bodisi z operaterjevim posredovanjem. Pri testiranju varnosti moramo predvsem ugotoviti, če sistem preprečuje neavtoriziran dostop. Pri testiranju obremenitve sistem izpostavimo nenormalnim obremenitvam (število uporabnikov, število vhodnih podatkov, poraba spomina itd.) in opazujemo, kdaj sistem izpade ali postane neuporaben. Pri testiranju občutljivosti skušamo odkriti, ali lahko določene kombinacije vhodnih podatkov, ki so posamezno obravnavani sicer pravilni, povzročijo nestabilnost sistema oziroma napačno procesiranje. Programsko opremo, namenjeno širšemu krogu uporabnikov, pred prodajo pogosto damo v poskusno uporabo zainteresiranim posameznikom ali skupinam v obliki tako imenovanih verzij α in β .” [1]

2.2.4 Prevzemno testiranje

Prevzemno testiranje preverja skladnost delovanja aplikacije z zahtevami naročnika. Izvaja se v testnem okolju, ki simulira produkcijsko okolje, kjer bo aplikacija delovala. Poteka z uporabniškega vidika, izvaja pa ga končni uporabnik oziroma naročnik. Je zadnja faza testiranja, preden je aplikacija predana v produkcijsko delovanje. Testiranje pa se s tem korakom še ne konča. Ko aplikacijo predamo v uporabo končnim uporabnikom, ti odkrijejo napake, ki smo jih v fazah testiranja spregledali oziroma jih nismo predvideli. Poleg tega pa

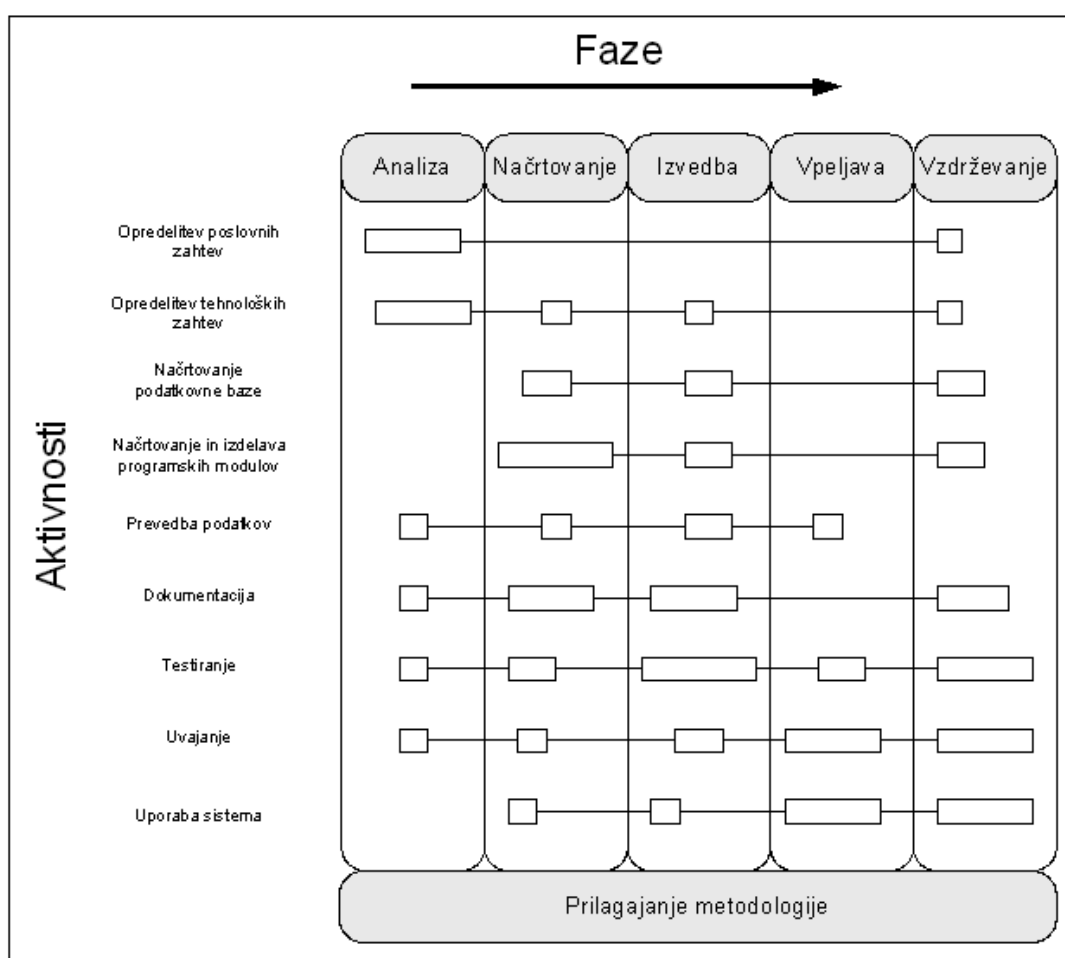
prihaja tudi do novih zahtev, ki zahtevajo uvedbo novih funkcionalnosti. Pri tem vedno obstaja možnost nastanka novih napak, ki jih lahko odkrijemo z regresijskim testiranjem.

2.2.5 Regresijsko testiranje

To je vrsta testiranja, ki zagotavlja, da spremembe, bodisi popravilo napak ali dodane nove funkcionalnosti, niso povzročile odpovedi delovanja aplikacije. Izvaja se lahko v vseh fazah testiranja, običajno pa v fazi systemskega testiranja. Prednosti avtomatizacije testiranja pridejo do izraza prav v tej fazi. Gre za pogosto izvajanje množice testnih scenarijev, ki jih lahko izvajamo kar čez noč. S tem procesom zagotavljamo, da spremembe na izdelku ohranjajo isto akovost, kot jo je ta imel pred uvedbo sprememb.

2.3 Življenjski cikel testiranja

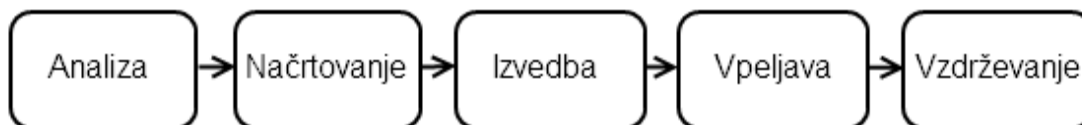
Enotna metodologija razvoja informacijskih sistemov (v nadaljevanju EMRIS) opredeljuje faze, aktivnosti in opravila v razvojnem procesu informacijskega sistema. Življenjski cikel testiranja je zelo povezan z razvojnim procesom IS. Spodnja slika prikazuje faze razvojnega procesa, kot ga določa EMRIS:



Slika 1: Faze razvojnega procesa, kot ga določa EMRIS

Vsaka faza procesa opredeljuje aktivnost in opravila testiranja.

Naslednja slika pa prikazuje življenjski cikel testiranja skozi faze razvojnega procesa IS:



Slika 2: Življenjski cikel testiranja skozi faze razvojnega procesa IS

2.3.1 Prva faza: analiza

V fazi analize določimo strategijo testiranja. Izdelamo dokument o strategiji testiranja, v katerem opredelimo testno okolje, kjer se bo testiranje izvajalo, testna orodja in testne podatke, ki se bodo pri testiranju uporabljali. Testna orodja in testni podatki obsegajo tri kategorije: testiranje izvajanja modulov, sklopov modulov in aplikacij, testiranje modulov, sklopov modulov in modulov, ki obsegajo povezovanje z drugimi IS, ter testiranje celotnega aplikativnega sistema in izvedbo potrditvenega testa.

2.3.2 Druga faza: načrtovanje

V fazi načrtovanja izdelamo model in načrt testiranja. Model testiranja vsebuje seznam testnih scenarijev. Testni scenarij mora zagotoviti pregled nad tem, kdo, kdaj, kaj, kje, kako in zakaj je testiral. Scenarij predstavlja spremljanje nekega postopka od začetka do konca, v okviru katerega za izbrane podatke spremljamo njihovo uporabo in spreminjanje, s čimer potrdimo pravilnost delovanja testnega objekta. Model testiranja vsebuje scenarije za testiranje modulov, sklopov modulov, scenarije za testiranje modulov, ki obsegajo povezovanja z drugimi informacijskimi sistemi in scenarije za testiranje celotnega aplikativnega sistema ter kriterije za uspešno prestani potrditveni test aplikativnega sistema.

Načrt testiranja izdelamo v skladu s testnimi scenariji, ki so določeni v modelu testiranja. Sestavljen je iz načrta testiranja posameznih modulov, načrta testiranja sklopov modulov, načrta testiranja modulov, ki obsegajo povezovanje z drugimi informacijskimi sistemi, načrta testiranja delovanja aplikativnega sistema kot celote in načrta izvedbe potrditvenega testa. V načrtu testiranja je treba za vse v modelu testiranja opredeljene scenarije določiti njihove časovne zahteve, termine izvajanja posameznih scenarijev in opredeliti izvajalce testiranja. Cilj testiranja aplikativnega sistema je preveriti njegovo pravilno delovanje v drugem (testnem) okolju glede na v modelu testiranja zamišljene scenarije. Potrditveni test je po uporabljenih testnih scenarijih enak testiranju aplikativnega sistema, razlika je le v okolju, v katerem se testiranje izvaja.

2.3.3 Tretja faza: izvedba

V tej fazi izvedemo testiranja na različnih nivojih in pripravimo testno okolje. Testiranje modulov in sklopov modulov izvajamo vzporedno z njihovo izdelavo. Module testiramo sproti, in sicer takoj, ko so izdelani – ne čakamo, da so izdelani vsi. Osnovno testiranje izvede razvijalec, nato pa delo prevzame ekipa za testiranje in module testira. Po odkritih napakah

se te odpravijo in moduli ponovno testirajo. Testiramo tako posamezne module kot sklope modulov ter module, ki obsegajo povezovanje z drugimi informacijskimi sistemi.

V testnem okolju, ki ga moramo pripraviti, bo izveden test aplikacije, ki pa ga je treba izvesti izven razvojnega okolja. Priprava testnega okolja vključuje: pripravo strežnikov in odjemalcev, testne podatkovne zbirke ter prevedbo podatkov in uskladitev s testnimi okolji v drugih organizacijskih sistemih, s katerimi se bo obravnavani IS povezoval.

Test aplikacije izvedemo v pripravljenem testnem okolju glede na načrt testiranja aplikacije. Običajno ga izvedemo v več iteracijah, v katerih preverimo vse scenarije, ki so bili zabeleženi v modelu testiranja. Ko se rezultati testiranja po več iteracijah ujemajo s pričakovanimi rezultati, testiranje zaključimo.

2.3.4 Četrta faza: vpeljava

V fazi vpeljave je testiranje zaključeno, pripravimo okolje za potrditveni test in tega tudi izvedemo. Pred začetkom potrditvenega testa mora biti dokončana vsa dokumentacija in pripravljena podatkovna zbirka s testnimi podatki za potrditveni test. Nameščena mora biti vsa potrebna sistemska in aplikativna programska oprema.

Namen potrditvenega testa je ugotoviti in potrditi, da aplikacija vsebuje vso zahtevano funkcionalnost (kriteriji za to se nahajajo v modelu testiranja kot kriteriji za potrditveni test), kot je bila določena v fazi analize, oziroma opisati morebitne pomanjkljivosti ali napake in oceniti njihovo kritičnost ali vpliv na delovanje aplikacije.

2.3.5 Peta faza: vzdrževanje

V tej fazi se opravi testiranja na različnih nivojih. Odločitev o načinu in potrebah testiranja določita metodolog in vodja razvojne skupine. Treba je pripraviti nove testne scenarije za vse module, ki so bili nadgrajeni ali posodobljeni oziroma na novo razviti. Za to lahko kot izhodišče uporabimo že obstoječe scenarije, ki jih glede na potrebe testiranja posodobimo, za nove module pa je v vsakem primeru treba izdelati nove testne scenarije.

Treba je izdelati načrt, ki opredeljuje časovne termine in druge vire, ki so potrebni za izvedbo testnih scenarijev. Pri testiranju funkcionalnosti posodobljenih in nadgrajenih modulov so lahko zraven novih testnih scenarijev uporabljeni tudi že obstoječi – stari testni scenariji, s katerimi je zagotovljeno, da je obstoječa funkcionalnost sistema ohranjena. Načrt testiranja tudi natančno opredeli, da bodo uporabljeni stari testni scenariji.

Izvede se testiranje modulov in sklopov modulov, ki so bili posodobljeni ali nadgrajeni.

Treba je posodobiti dokumentacijo ostalih ključnih izdelkov aktivnosti testiranja, in sicer dokument o strategiji testiranja, dokumentacijo o pripravi testnega okolja in dokumentacijo o pripravi okolja za potrditveni test.

Izvede se test celotne aplikacije.

Če je obseg vzdrževalnih del velik, se lahko izvede ponoven potrditveni test funkcionalnosti aplikacije. V primeru manjših posodobitev in nadgradenj pa izvedba potrditvenega testa ni potrebna.

3. AVTOMATSKO TESTIRANJE

3.1 Pojav avtomatskega testiranja

V želji po doseganju čim boljše kakovosti programske opreme se vse več podjetij obrača k avtomatizaciji upravljanja v razvoju – prav nič drugače ni pri testiranju.

Nekatera orodja, ki se ukvarjajo s področji nadzora razvoja programske opreme, so v razvojnih okoljih postala nepogrešljiva. Tako imamo danes na trgu veliko orodij, ki omogočajo avtomatsko testiranje.

Testiranje lahko izvajamo ročno (brez uporabe orodij) ali avtomatsko (s pomočjo orodij). Avtomatsko testiranje je pogosto cenovno najbolj ugodno, vendar pa kombinacija ročnega in avtomatskega testiranja ostaja najboljša rešitev. Kljub temu velja omeniti, da ročno testiranje ostaja primarni način testiranja v približno 80 odstotkih. Običajno je tako zaradi ozkih grl v ciklu razvoja programske opreme. Vsekakor pa drži, da je ročno testiranje časovno zamudno, težavno in predstavlja velik strošek.

Avtomatizacija testiranja dejansko pomeni pisanje računalniškega programa, ki bi izvajal sicer ročno opravljeno delo. Za avtomatizacijo testiranja se zato odločimo takrat, ko so nam pomembni hitrost, učinkovitost, točnost in natančnost ter vztrajnost (orodje za testiranje lahko teste opravlja znova in znova).

Pri hitrosti je ključna možnost obdelave večje količine podatkov v bistveno krajšem času, pri učinkovitosti govorimo o tem, da smo med potekom avtomatskega testiranja prosti oziroma na razpolago za druge naloge, pri točnosti in natančnosti izključujemo možnost človeške napake, ki se, denimo, pojavi ob padcu koncentracije, pri vztrajnosti pa cenimo dejstvo, da lahko testno orodje teste ponavlja v nedogled. [3]

Avtomatsko testiranje delimo na testiranje modulov in na avtomatske funkcionalne sistemske teste. Medtem ko je prvo skoraj vedno avtomatizirano, pa drugo temelji na posnemanju človekovega testiranja s testnim orodjem. Lahko si predstavljamo, da so tovrstna orodja zelo draga, zato je dobro pretehtati med stroški in koristmi, ki nam jih prinašajo.

Avtomatizacija testiranja je bila vedno zanimiva alternativa dragemu, časovno zahtevnemu in nekonsistentnemu ročnemu testiranju. Najbolj običajna vprašanja, na katera si moramo odgovoriti, ko razmišljamo o avtomatizaciji testiranja, so:

- kdaj investirati v avtomatizacijo testiranja,
- kako uvesti oziroma nadaljevati s programom avtomatizacije testiranja in
- kolikšen del ročnega testiranja bi moral preiti na avtomatsko testiranje.

Odgovori na vprašanja, kdaj, kako in koliko, so vedno odvisni od konteksta celotnega programa. Kaj je cilj avtomatizacije? Ključni programski dejavniki vključujejo: razvoj koncepta, cilje kakovosti in hitrost razvoja. Kdaj, kako in koliko avtomatizacije testiranja

vnesti v program je odvisno od teh dejavnikov – povratak naložbe se mora poravnati z njimi, sicer bo dolgoročna posledica avtomatizacije testiranja veliko tveganje in skoraj zagotovo neuspeh.

Najbolj običajni ključni dejavniki so:

- razvojni koncept (agilen, neagilen, opravljen, neopravljen),
- cilji kakovosti (hitrost rešitve pred napako) in
- ciljna hitrost uvajanja (obseg nove funkcionalnosti na izid).

Zavedati se moramo, da avtomatizacija testiranja ni srebrna krogla (angl. silver bullet) – če jo obravnavamo kot tabletko, ki pozdravi vse, bomo zagotovo razočarani. Avtomatizacija testiranja je le pomočnik, kar pomeni, da bo ustrezni program za avtomatsko testiranje zagotovo prinesel vidne rezultate za vaše podjetje, če sicer že imate učinkoviti testni proces, ki pa je preobremenjen.

3.2. Kdaj, kako in koliko avtomatizirati

3.2.1 Kdaj avtomatizirati

Preden se odločimo za avtomatizacijo testiranja, moramo pretehtati nekatere ključne kazalce. Ti nam pokažejo, ali bi avtomatizacija testiranja v našem primeru izboljšala rezultate dela. Ključni kazalci v cilju programa bi tako bili:

- ne dosegamo ciljev kakovosti,
- hitrost reševanja napak v proizvodnji nikakor ni sprejemljiva,
- ciljna hitrost uvajanja ni ustrezna, saj testiranje deluje kot ozko grlo, in
- testiranje ni zaključeno znotraj predvidenega časovnega okvirja.

Vse to projektna ekipa običajno izkusi istočasno – dejansko gre za to, da je ekipa dosegla svoje maksimalne testne zmoglosti. Ko se to zgodi, ima vodstvo dve možnosti: ali vzpostavitev testne ekipe za ročno testiranje ali pa zagon avtomatskega testiranja za potrditev koncepta (angl. proof-of-concept), ki bi pokazalo, ali se lahko avtomatsko testiranje spopade s testnim primanjkljajem.

Opozoriti moramo, da testiranje samo pogosto ni ozko grlo, vendar dokler ga kot takega dojemamo, se ne ukvarjamo z drugimi težavami v kakovosti sistema. Avtomatizacija testiranja je en način, kako se oddaljiti od zaznavanja testiranja kot ozkega grla s tem, da testna ekipa pridobi na moči, medtem ko se program lahko osredotoči na druge težave kakovosti v sistemu – na primer začetna kakovost kode.

3.2.2 Kako uvesti načrt avtomatskega testiranja

Ljudje bodo potrebovali šolanje in posledično znanja, s katerimi bodo lahko vpeljali avtomatizacijo.

Pri obravnavi procesa bosta morala testna skupina in program kot celota opredeliti, kako se bo avtomatizacija testiranja vklopila v proces razvoja sistema – na začetku bi se morali osredotočiti na visoko donosne avtomatske aktivnosti, ki bodo sprostile ročne. Posledično, ko imamo dovolj razpoložljivih virov, se lahko spet osredotočimo na najbolj tvegana področja vpeljave, ki ne dosegajo ciljne kakovosti. Proces bo moral:

- vključiti avtomatizacijo testiranja v celotni razvojni proces,
- se osredotočiti na visokodonosne avtomatske aktivnosti,
- testirati ustvarjanje podatkov,
- prestatii dimni in regresijski test.

Vsak program avtomatskega testiranja bi se moral začeti s poštenim vrednotenjem ljudi, procesa in razpoložljivih tehnologij. Na podlagi te ocene, in če je upravičeno tudi na podlagi odzivov na zahtevo in predlogov (angl. request and proposal R&P), bi morali zagnati eno ali več iniciativ potrditve koncepta, ki bi ugotovile, kateri proces in tehnologija najbolj ustrezata našim kratkoročnim in dolgoročnim zahtevam.

Ekipa torej mora:

- oceniti ljudi, proces, tehnologijo in uspešnost trenutnega testnega procesa,
- prepoznati sistemske izvirne težave, ki ovirajo uspeh testiranja, ter
- oceniti razpoložljive procese in tehnologije.

3.2.3 Koliko avtomatizirati

Kolikšen delež celotnega testiranja bi moral biti avtomatiziran, je odvisno od ključnih programskih dejavnikov, ki so navedeni zgoraj. Obstajajo pa določene splošne smernice, ki bi jih morali vedno upoštevati:

- nikoli ne avtomatiziramo celotnega testnega procesa – izurjeno raziskovalno testiranje bo vedno prinašalo napake, ki jih avtomatsko testiranje ne bo zaznalo,
- avtomatsko testiranje je orodje, pripomoček in ne rešitev – če ga ne uporabimo pametno in z znanjem, se bomo hitro soočili s posledico neučinkovitega testiranja,
- avtomatizacija testiranja bi vedno morala povrniti naložbo v razmerju vsaj 4 proti 1 in
- teža tekočega vzdrževanja se bo s časom zmanjšala, razen če aplikacija postane statična.[25]

Po tem osnovnem premisleku pa je delež celotnega testiranja odvisen od tega, kateri ključni programski dejavniki so pomembni za posamezno organizacijo. Če je cilj očitno zvišanje hitrosti uvajanja ali zagotovitev doseganja ciljev kakovosti, potem bo dobro avtomatizirati pomemben del obstoječe funkcionalnosti. Po drugi strani pa, če je cilj znižanje truda s testiranjem celotne testne skupine, bi se morali osredotočiti na visoko donosne avtomatske aktivnosti:

- testiranje ustvarjanja podatkov,
- dimni test in
- regresijski test.

3.3 Orodja za funkcionalno testiranje

Najbolj razširjena orodja za avtomatsko funkcionalno testiranje so orodja proizvajalcev HP, IBM, Borland, Compuware in Oracle.

Ker je funkcionalno testiranje dejansko način posnemanja uporabe določene aplikacije, imajo navadno vsa orodja zanj funkcijo posnemi in predvajaj. To pomeni, da se snema vsak naš korak (dejansko sprehod skozi aplikacijo), ki pa predstavlja en testni scenarij. Skripte, ki jih posnamemo s takšnimi orodji, imajo nestrukturirano kodo z nespremenljivimi podatki. Občutljive so na najmanjše spremembe obnašanja testirane aplikacije (npr. zamik gumba v eno smer, dodano vnosno polje itd.). Glavna prednost takih orodij je hitrost izdelave testa. Tovrstna orodja so namenjena vsem preizkuševalcem, tudi tistim brez znanja programiranja.

Novejša orodja za funkcionalno testiranje so objektno usmerjena, kar pomeni, da z njimi posnamemo testni scenarij v obliki testne skripte. Na voljo ima vnaprej pripravljeno podatkovno zbirko, iz katere črpa testne podatke, ob snemanju skripte pa orodje posname vsako akcijo uporabnika, da lahko med predvajanjem testa natančno ponovi vsako dejanje. Tem orodjem so skupne lastnosti:

- razpoznavanje objektov (med izvajanjem testa orodje prepozna elemente uporabniškega vmesnika, npr. gumbi, okna, ...),
- sinhronizacija testiranja (starejša orodja so se pogosto znašla v težavah pri sinhronizaciji zaradi prehitevanja skripte – recimo prezgoden klik na gumb, ko ga sploh še ni na mestu),
- branje podatkov iz vnaprej določene podatkovne zbirke (pri nekaterih orodjih lahko testne podatke črpamo zaporedno ali naključno iz podatkovne zbirke, kar omogoči hitro izvajanje testnih primerov z različnimi testnimi podatki),
- preverjanje rezultatov (preverjamo uspešnost testa, ki nam je običajno prikazana v posebnem oknu s poročilom o testu; različna testna orodja na različne načine prikazujejo rezultate, zaželeno pa je, da poročilo vsebuje podatke o številu izbranih testov, številu testov, ki so bili dejansko izvedeni, številu pozitivnih in negativnih testov, številu odkritih napak, številu nedokončanih testov, podrobne informacije o negativnih in nedokončanih testih, omejitve orodja in časovne parametre, tj. čas trajanja posameznega testa. Testno orodje z dobro oblikovanim grafičnim vmesnikom, ki osvežuje podatke že med testiranjem, nam omogoča, da rezultate spremljamo sproti in v primeru velikega števila neželenih odstopanj test prekinemo ter rezultate posredujemo razvijalcem – nekatera orodja celo omogočajo shranjevanje slik oken, kjer se napake zgodijo) in
- vizualizacija kode (testno skripto lahko pregledamo v drevesnem pregledu ali pregledu kode. V primeru, da poznamo programski jezik, ki ga orodje podpira, imamo v slednjem primeru možnost spreminjanja oziroma dopolnjevanja kode).

3.3.1 Tipi orodij v razvojnem ciklu testiranja programske opreme

V vsaki fazi testiranja je razvojni cikel testiranja podprt z določenim tipom orodij. Spodnja tabela ponuja njihov pregled po posameznih fazah. [2]

Faza življenjskega cikla	Tip orodja	Opis orodja
Faza analize	Poslovno modeliranje	Posnemanje opredelitve uporabniških zahtev, hitra avtomatska konstrukcija
	Sledenje napakam	Upravljanje napak, ki se pojavijo v življenjskem ciklu
	Generatorji dokumentacije	Avtomatizacija generiranja dokumentov
	Upravljanje zahtev	Upravljanje in organizacija zahteve, načrtovanje testnih procedur in poročanje o napredku testiranja
Faza opredelitve zahtev	Verifikatorji zahtev	Verifikacija sintakse, semantike, možnosti testiranja
	Generatorji primerov uporabe	Ustvarjanje primerov uporabe
	Načrtovanje podatkovne zbirke	Razvoj sistemov odjemalec-strežnik druge generacije
Analiza in načrtovanje	Strukturni diagram, diagram podatkovnih tokov, diagram zaporedja	Modeliranje podatkov in procesov
	Generatorji testnih procedur	Generiranje testne procedure iz zahtev, načrta ali podatkov in objektnih modelov
	Preverjanje sintakse / razhroščevalniki	Preverjanje pravilnosti sintakse, sposobnosti razhroščevanja – navadno jih dobimo s prevajalnikom kode.
Kodiranje	Puščanje pomnilnika ter napake med izvajanjem	Zaznavanje napak med izvajanjem in zaznavanje puščanja pomnilnika
	Testiranje izvorne kode	Verifikacija vzdrževanja, prenosljivosti, kompleksnosti in skladnosti s standardi
	Statični in dinamični analizatorji	Prikaz kakovosti in strukture kode

Preglednica 3: Podpora razvojnim ciklom z določenim tipom orodij

3.4 Zakaj avtomatsko testiranje

Ugotovili smo že, da je ogromno pozornosti namenjene testiranju, saj bi vsako podjetje želelo izdati preverjeni izdelek. Navedel sem že podatek, da veliko podjetij navadno testiranje stane vsaj 50 % celotnega razvojnega proračuna.

Inštitut Quality and Assurance Institute je leta 1995 izvedel raziskavo, v kateri so ugotovili, da so testna orodja v začetni fazi sicer potrebovala dobršen del naložb, da pa so na koncu podjetjem omogočila pomemben prihranek časa, ki je sicer namenjen testiranju. V raziskavi so torej prišli do spoznanja, da avtomatsko testiranje omogoča kar 75-odstotni prihranek časa!

3.5 Cilji avtomatskega testiranja

Če želimo doseči kakovostno raven avtomatskega testiranja, nikoli ne smemo zanemariti pomena človeškega dejavnika – v dobrem in slabem smislu. Po eni strani želimo z avtomatizacijo povečati učinkovitost testne ekipe in čim bolj znižati možnost človeške napake, po drugi strani pa celotnega postopka testiranja ne želimo prepustiti avtomatskim orodjem, ki se sama po sebi ne bodo znala prilagoditi spremembam, novim situacijam in se domisliti novih poti. Pri načrtovanju razvoja avtomatskih orodij je tako ključnega pomena ustrezna dvosmerna komunikacija med razvojno in testno ekipo ter človeškega znanja.

Za boljši pristop k sistematski in produktivni avtomatizaciji Bach predlaga upoštevanje naslednjih načel:

- avtomatizacija testiranja naj ne poveča obsega dela testnega inženirja, temveč naj mu pomaga doseči boljšo učinkovitost. Inženir naj si z avtomatskimi orodji pomaga prihraniti čas, orodja pa naj bodo deležna njegovega znanja pri prilagajanju na nove situacije,
- avtomatizacija ni zgolj zagon testov, medtem ko inženir počiva. Podobno kot pri prvi točki poudarja, da človeško znanje ostaja nenadomestljivo, zlasti na področju generiranja testov (avtomatsko orodje lahko generira in posreduje naključne podatke in različne kombinacije parametrov, s katerimi želimo preveriti pravilno delovanje sistema), konfiguracije sistema (orodje lahko spreminja parametre na testiranem sistemu in opravi enake teste ob različnih delovanjih sistema), simulacije (orodje lahko simulira podsystem ali okolje pri robnih pogojih delovanja), zagona testov (orodje lahko samo nadzoruje upravljanje testiranega sistema, prav tako lahko vnaša spremembe v grafični vmesnik testiranega sistema, kakor da bi jih vnašal človek), zajema in analize rezultatov (orodja nenehno spremljajo potek testiranja in beležijo rezultate, obveščajo nas o izvedbi in uspešnosti testov).[24]

Pomembno je predvidevanje velikih sprememb v okolju, ki bodo vplivale tudi na naš izdelek. Spremembe na njem bodo namreč za seboj potegnile spremembe pri testiranju, zato moramo paziti, da način testiranja (kar se pri avtomatskem rado zgodi) ne zastara. Za preprečevanje tega imamo dve možnosti: ali nakup robustnega, neobčutljivega orodja ali pa nakup poceni testnega orodja, ki ga lažje spreminjamo, vendar je časovno potratno.

Pri nakupu testnih orodij navadno pride do izraza tudi naša spretnost pri iskanju testnih orodij. Na trgu jih je namreč mogoče najti pod tem imenom, kar pa ne zagotavlja nujno kakovosti. Nasprotno pa skoraj zagotovo pomeni visoko ceno. Spretna vodstva podjetij znajo kot testna orodja uporabiti tudi druga orodja, ki so izdelana za druge namene in so pogosto celo brezplačna.

Pretirano zanašanje na avtomatsko testiranje nas precej hitro lahko stane tudi poslovnega uspeha. Vedno znova je treba poudariti, da sta aktivno človekovo znanje in njegova zmožnost hitrega prilagajanja novim situacijam nenadomestljiva. To preprosto pomeni, da še tako dobra testna orodja in še tako veliko število njih ne morejo nadomestiti inženirja.

Kot pri vsaki stvari, je tudi pred začetkom testiranja treba natančno opredeliti prednostne naloge testiranja. Želimo namreč testirati tiste lastnosti, ki bodo zagotovile odkrivanje napak, ki vplivajo na stabilnost in pravilno delovanje sistema. Pri opredeljevanju teh se je dobro postaviti v položaj kupca in izdelek opazovati z njegovega stališča.

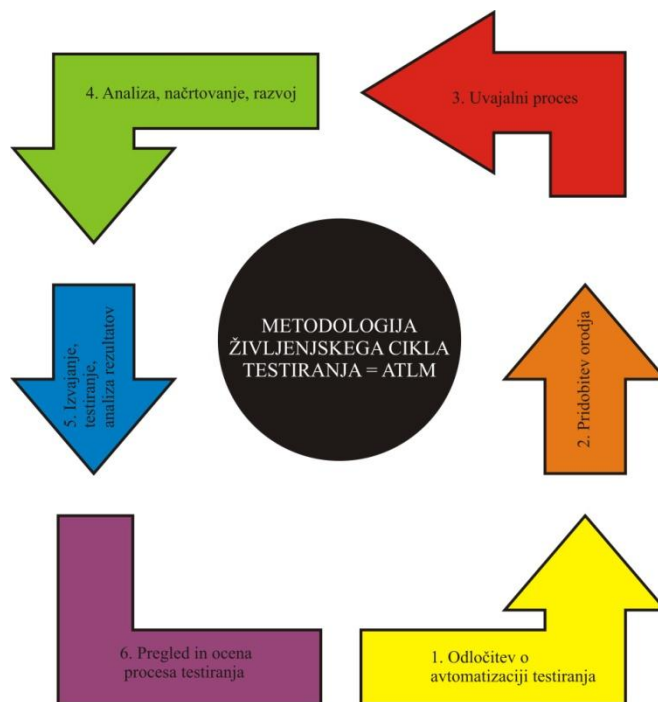
3.6 ATLM (Automated Test Lifecycle Methodology)

Omenil sem že, da je uvedba avtomatskega testiranja zahteven dogodek, saj se ne bo zgodila sama po sebi zgolj z nakupom testnih orodij, temveč se mora novemu načinu dela prilagoditi celotna testna skupina. Tudi k tej novosti je zato treba pristopiti kar se da organizirano. Za ta prehod lahko uporabimo metodologijo ATLM. Metodologija ATLM poudarja predvsem dokumentiranje in formalizacijo procesa testiranja. To je edina metodologija, ki obravnava celovit pristop pri uvedbi avtomatizacije testiranja. Glede na to, da je testiranje modulov skoraj vedno avtomatizirano, pa daje ATLM poudarek predvsem avtomatizaciji funkcionalnega testiranja.

ATLM je namenjena procesu testiranja, ki se odvija istočasno z razvojnim življenjskim ciklom programske opreme. Vključuje tako testiranje modulov kot sistemsko testiranje. Posebej poudarja tveganja in koristi avtomatizacije ter obravnava načrtovanje, analizo, izvajanje in upravljanje procesa testiranja.

ATLM je neodvisna od razvojne faze programske opreme in jo je mogoče uporabiti v kateri koli razvojni fazi, saj se tudi ne nanaša na konkretno testno orodje in ne priporoča specifične uporabe orodja.

Prehod z ročnega na avtomatsko testiranje po tej metodologiji poteka skozi šest faz in množico aktivnosti. [3]



Slika 3: Prikaz metodologije življenjskega cikla testiranja (ATLM)

3.6.1 Prva faza: odločitev o avtomatizaciji testiranja

Uvedba avtomatskega testiranja je dolgoročna naložba, zato moramo v začetni fazi pričakovati predvsem stroške in šele kasneje koristi. Vodstvo in testna ekipa morata dobro pretehtati razloge za in proti uvedbi avtomatskega testiranja. Oceniti moramo, kaj naj nam novost prinese, predvsem pa dobro izbrati testno orodje, ki mora ustrezati izkušnjam preizkuševalcev in programski opremljenosti. Pri kalkulaciji ne smejo pozabiti na izobraževanje zaposlenih ali celo zaposlovanje novih inženirjev.

Vodstvo mora svoja pričakovanja čim bolj prilagoditi realnosti in se zavedati, da eno orodje ne bo zmoglo podpreti celotnega procesa testiranja, da vsi testi avtomatski, da sprva ne bo časovnega in finančnega prihranka pri zaposlenih, da se avtomatski testi v času uvajanja še ne bodo izvajali izven delovnega časa in da v splošnem mnogih koristi še nekaj časa ne bo možno videti.

3.6.2 Druga faza: izbor orodij

Ko vodstvo in testna ekipa sprejmeta odločitev o avtomatizaciji testiranja, je čas za izbiro testnih orodij. Najbolj pomembno je, da je orodje prilagojeno razvojnemu orodju, ki ga razvijalci že uporabljajo. Dobro je, da pripravimo nabor obstoječih razvojnih orodij in oblikujemo merila, na podlagi katerih bomo izvajali selekcijo. Med merila, ki morajo biti jasno opredeljena, vključimo, npr. ceno, združljivost, uporabnost na različnih projektih, zahtevnost uporabe itd.

Verjetno si marsikateri testni inženir v tej fazi želi, da bi posamezno orodje lahko preizkusil, vendar tega ne omogoča prav veliko njih, če pa že, je preizkusni čas zelo kratek. Z malo sreče ob koncu te faze izberemo najbolj ustrezno orodje.

3.6.3 Tretja faza: uvajalni proces

Po uspešno zaključeni prejšnji fazi (izbrali smo ustrezno testno orodje) nastopi čas za vpeljavo v podjetju. Pomembno je, da na tej točki vsi udeleženi razumejo, da vpeljujemo novo orodje.

Če se v podjetju že izvaja kakšno testiranje, ga najprej analiziramo, kar moramo storiti tudi, če obstoječi proces ni dobro opredeljen. Opredeliti moramo strategijo, namene in cilje testiranja. Če testiranje doslej še ni bilo izvedeno, preprosto začnemo na novo z novim procesom.

Zdaj je tudi čas, da preverimo ustreznost testnih orodij, kar storimo tako, da ocenimo, ali je orodje primerno glede na izkušnje preizkuševalcev, ali se prilagaja drugim razvojnim orodjem, ali imamo na voljo dovolj časa za njegovo vpeljavo, ali bomo potrebovali dodaten kader ipd.

Ta preizkus vedno izvajamo na že obstoječi testni aplikaciji in s tem preverimo njuno združljivost ter opazimo morebitne težave. Če se nam zgodi, da v tej fazi ugotovimo, da izbrano orodje ni primerno, nadaljujemo z iskanjem novega. V najslabšem primeru, tj. da ustreznega testnega orodja ne najdemo, zamisel o avtomatizaciji začasno opustimo in nadaljujemo z ročnim testiranjem.

3.6.4 Četrta faza: analiza, načrtovanje in razvoj testov

Še preden načrtujemo testne primere, moramo analizirati zahteve testiranja. Analiza vključuje testne scenarije, teste združljivosti (testiramo aplikacijo na strojni opremi, na kateri bo tekla) in izpostavitve programske opreme nepredvidljivemu obnašanju uporabnika.

Ko smo opredelili zahteve testiranja, se lahko lotimo načrtovanja. Testni načrt je dobro začeti oblikovati že v prejšnjih fazah in ga sproti le dopolnjevati, vključuje pa obravnavanje vseh aktivnosti, potrebnih za izvajanje testiranja, in verifikacijo, organiziranost in učinkovito uporabo procesov, osebja, orodja in opreme, vloge ter odgovornosti udeležencev projekta, doseg testnega procesa glede na omejitve v času in pri zaposlenih, zahteve za testno okolje, uporabljene metode testiranja, časovni načrt za razvoj in izvajanje testnih primerov itd. Prav tako moramo za vse testne primere predvideti testne scenarije. Pri tem moramo za vsak test opredeliti, kakšne vrste je (ročni ali avtomatski), sisteme kodiranja, pristop pri izdelavi programske kode testov, pa tudi povsem osnovne podatke, kot so ime, avtor, vhodni in izhodni podatki, potrebna dejanja itd.

Po standardu ANSI/IEEE 829-1983 osnutek splošnega testnega načrta je videti tako:

Identifikator testnega načrta → uvod → opis izdelka → lastnosti, ki bodo testirane → karakteristike, ki ne bodo testirane → uporabljene metode testiranja in terminalna kriterijska funkcija → odpovedna kriterijska funkcija → prekinitev in nadaljevanje testiranja → identifikacija izročljivih dokumentov → potrebne aktivnosti za izvedbo testiranja → opis potrebnega testnega okolja → identifikacija odgovornosti → človeški viri in potrebno šolanje → roki → tveganja in posledice → potrditev testnega načrta → dodatki.

Tu obravnavamo vse aktivnosti, ki so potrebne za izvajanje testiranja in verifikacijo, poleg tega pa tudi organiziranost in učinkovito uporabo procesov, osebja, orodja in opreme.

3.6.5 Peta faza: izvajanje testiranja in analiza rezultatov

V tej fazi izvedemo predhodno pripravljene testne scenarije, rezultate zbiramo in jih analiziramo, testiranje izvajamo na vseh ravneh (od testiranja modulov do sistemskega testiranja), po izvedenem testiranju pa zabeležimo odkrite napake. Rezultati testov so lahko:

- pozitivni (test je bil izveden v skladu s pričakovanji, rezultati so pričakovani, kaže, da testirana komponenta oziroma programska oprema deluje pravilno),
- negativni (test ni bil uspešen, rezultati niso skladni s pričakovanji oziroma z zahtevami, testirana komponenta oziroma programska oprema ne deluje pravilno),
- lažni pozitivni (test je bil uspešen, vendar komponenta ne deluje pravilno; razlog se lahko skriva v napačnem razpoznavanju objektov, napačnem branju rezultatov, ...; vsekakor moramo test ponoviti, morda tudi dodelati kodo testnih primerov) in
- lažni negativni (test ni bil uspešen, vendar komponenta deluje pravilno; dobljeni rezultat primerjamo s pričakovanim, saj je razlog za ta rezultat lahko v napakah v testnih primerih).

Dejansko obnašanje testnega orodja se bo seveda pokazalo, šele ko ga bomo začeli uporabljati v različnih situacijah. Ključno je, da prva izvajanja neposredno spremljamo in da na kritičnih točkah težave odpravljamo.

Za analizo rezultatov uporabljamo različne metrike, ki pa jih izberemo tako, da nam dajo čim več informacij. Primeri metrik so: pokrivanje testiranja (primerjava števila izvedenih testnih primerov s številom testnih scenarijev, ki so zapisani v specifikaciji), gostota napak (število vseh najdenih napak v primerjavi s številom izvedenih testnih primerov) in trenutna stopnja kakovosti (primerjava števila uspešnih s številom izvedenih testnih primerov).

3.6.6 Šesta faza: pregled in ocena procesa testiranja

Šesta, zadnja faza metodologije ATLM se izvaja skozi celotni življenjski cikel testiranja, ker je cilj nenehno iskanje izboljšav procesa. Med procesom in po izvajanju testnih primerov moramo upoštevati izbrane metrike in analizirati rezultate. Metodologija ATLM predlaga naslednje korake:

- po izvajanju testiranja testna skupina spremlja učinkovitost testnega orodja in prouči možnost sprememb za morebitne izboljšave,
- na podlagi rezultatov testiranja vidi, ali je aplikacija zrela za produkcijo,
- skupina dokumentira vse zbrane izkušnje,
- dokumentacija celotnega življenjskega cikla naj bo zbrana v sistemu za verzioniranje in
- izračuna stroške avtomatizacije (podatke zbiramo že med procesom testiranja).

3.7 Primerjava ročnega in avtomatskega testiranja

Za ročno testiranje se odločamo, zlasti ko je razvoj programske opreme še v začetni fazi, ko nam primanjkuje časa za avtomatizacijo, ko nimamo kadra, ki bi avtomatizacijo lahko izvedel, ko avtomatizacija ni smiselna zaradi visokih stroškov in neponovljivosti, ko testiramo uporabniški vmesnik ali ko poskušamo napako ponoviti, da bi lahko opredelili njen izvor. [3]

Dobro pripravljena avtomatizacija testiranja ima številne, v podpoglavjih 3.1, 3.2 in 3.4 opisane prednosti, kljub temu pa zahteva pazljivo načrtovanje in je lahko časovno zamudna. Veliko pozornosti in s tem časa bomo namenili pripravi testnih primerov, ki jih moramo spreminjati in preverjati ob vsaki spremembi, ki jo razvijalci naredijo v programu.

Mnoga podjetja se kljub navedenim prednostim testnih orodij branijo njihove uporabe. Običajno je razlog v pomanjkanju časa, ki bi ga testna ekipa morala nameniti učenju, mnogo orodij doslej pa tudi ni pokazalo takih rezultatov, kot so jih obljubljala.

Trditi, da je ena vrsta testiranja boljša od drugega, je nemogoče. Dejstvo je, da samo z enim načinom testiranja zagotovo ne bomo odkrili vseh napak, zato je najbolje uporabljati kombinacije ročnega in avtomatskega testiranja. To bi bilo idealno, vendar pa seveda pogosto ni izvedljivo, zato se moramo odločiti za eno možnost. Katera bo prevladala, je običajno odvisno od razvojne faze testirane programske opreme, testnega okolja in drugih dejavnikov.

3.7.1 Pregled avtomatizacije testiranja

Skripte za avtomatsko testiranje so dragocen način izvajanja regresijskega testiranja programske opreme. Preprečujejo, da bi z uvedbo novih funkcionalnosti povzročili nedelovanje obstoječe funkcionalnosti. Običajno dopolnjujejo ročno in raziskovalno testiranje, ki je bolj učinkovito za testiranje novih funkcionalnosti in zmogljivosti. Za popolno pokritost testiranja je treba izvesti iste avtomatske testne scenarije na celotnem spektru programskih okolij, spletnih brskalnikov, naprav in okolij.

Funkcionalno testiranje zagotavlja, da aplikacija deluje tako, kot mora – da deluje tako, kot uporabnik pričakuje, da deluje. Pove nam, ali dokončana aplikacija deluje pravilno in ali zagotavlja pravilno funkcionalnost. Funkcionalni testni scenariji obsegajo zahteve uporabnikov in posredujejo informacijo o tem, ali aplikacija ustreza tem zahtevam.

Pritisk na IT podjetja se neprestano stopnjuje. Zahteve managerjev silijo podjetja v iskanje novih načinov razvoja programske opreme in zagotavljanja konkurenčnih izdelkov. V času razvoja se podjetja tako srečujejo s krajšimi roki dobave in z enakimi ali s še manjšimi finančnimi sredstvi. Istočasno se managerji pričenjajo zavedati povezave med kakovostno programsko opremo in dobičkom.

Avtomatizacija je ključ do pospešitve, večje natančnosti in prilagodljivosti testiranja programske opreme, zraven tega pa omogoča hitrejše odkrivanje in popravljanje napak v programski opremi.

3.7.2 Funkcionalno avtomatsko testiranje v primerjavi z ročnim testiranjem

3.7.2.1 Prednosti funkcionalnega avtomatskega testiranja

Funkcionalno avtomatsko testiranje z zagotavljanjem celovitejše pokritosti testiranja:

- zmanjša tveganje za odkrivanje napak v produkcijski fazi ter poveča vračanje vlaganj v avtomatizacijo,
- omogoča hitrejšo izvedbo (računalniki so bistveno hitrejši v izvajanju funkcionalnih testnih skript kot ljudje, zato lahko v krajšem času izvedejo več testov, več aplikacij lahko testirajo v določenem časovnem intervalu in več projektov je lahko zaključenih pravočasno. V nasprotju z ljudmi lahko računalniki delujejo 24 ur na dan, ponoči ter čez vikende in praznike, ne dolgočasijo se in se ne utrudijo ter ne predpostavljajo, kaj deluje in kaj ne),
- omogoča večjo pokritost vseh testnih možnosti – da so vsi možni testi pokriti: (orodja za avtomatsko testiranje omogočajo testiranje programske opreme in izvajanje testnih skript na vseh priljubljenih spletnih brskalnikih, operacijskih sistemih itd. Z uporabo avtomatskih orodij je regresijsko testiranje aplikacije, ki se stalno spreminja, veliko lažje v primerjavi z ročnim testiranjem. Poleg tega omogočajo hitrejša in pogostejša iteracija testnih načrtov ter hitro dodajanje novih testnih skript za večjo pokritost celotnega testiranja),
- zagotavlja natančnejše testiranje in odkrivanje napak v zgodnejših fazah razvoja (orodja za avtomatizacijo omogočajo enostavnejše ponovitve in dokumentiranje najdenih napak, pospešijo razvojni proces pri dokazovanju pravilnosti delovanja funkcionalnosti na vseh delovnih okoljih, podatkovnih kompletih in poslovnih procesih),
- zagotavlja formalni proces (uvajanje avtomatizacije testiranja sili testno ekipo v formalizacijo delovnega procesa, kar se odraža v večji usklajenosti in boljšem dokumentiranju celotnega delovnega procesa) ter
- olajša ponovno uporabo testnih skript (ko so testne skripte enkrat napisane, jih lahko uporabljamo, ponovno uporabljamo, nadgrajujemo in popravljamo, če se aplikacija spremeni. Nobene potrebe ni po ponovnem pisanju nove testne skripte za isto funkcionalnost, če se aplikacija deloma spremeni).

3.7.2.2 Slabosti ročnega funkcionalnega testiranja

Ročno testiranje je časovno potratno. V našem primeru, denimo, imamo testni scenarij, po katerem je treba generirati veliko količino podatkov in preveriti delovanje aplikacije z velikim številom datotek. Testne scenarije, ki se ponavljajo ali so si podobni, je treba večkrat ponoviti. Slabo so pokriti koda, programska okolja itd.

4. IZZIVI NA PODROČJU TESTIRANJA

4.1 Goljufije pri avtomatizaciji testiranja

Velikokrat slišimo pritoževanje, da je večina projektov razvoja programske opreme neuspešnih. To nas ne bi smelo presenetiti – navzven je programska oprema videti nadvse preprosta, vendar pa je res tudi, da se "hudič skriva v podrobnostih". Izkušeni razvijalci programske opreme to vedo, zato k vsakemu novemu projektu pristopijo skeptično in z odprtimi očmi.

Najpomembnejše spoznanje, ki ga je James Bach pridobil v času devetletnega vodenja ekip za testiranje v IT industriji in pri delu z avtomatizacijo testiranja (v nekaterih najpremožnejših podjetjih v IT industriji), je, da so projekti testiranja prav tako dovzetni za napake kot ostali projekti razvoja programske opreme. Trdi celo, da po njegovih izkušnjah večkrat propadejo, največkrat zato, ker podjetja ne posvetijo dovolj pozornosti testnemu okolju oziroma mu je namenijo bistveno manj, kot je namenijo razvoju programske opreme.

Ne čudi torej dejstvo, da skoraj vsi samooklicani strokovnjaki, aktivni testni preizkuševalci, managerji testiranja ter seveda podjetja, ki prodajajo testna orodja, z velikim navdušenjem priporočajo avtomatizacijo testiranja. Posledično niti ne preseneča dejstvo o slabi obveščenosti javnosti in analizah o avtomatskem testiranju – vse to je namreč posledica nezrelosti na tem področju. Kot skupnost je področje testiranja v fazi občudovanja zamisli o avtomatizaciji testiranja, ki še ni prešla v fazo zaznavanja pasti in morebitnih težav.

Avtomatizacija testiranja se res zdi zelo dobra zamisel, lahko bi rekli tudi priljubljena. Večina polno zaposlenih preizkuševalcev verjetno sanja o tem, da bi lahko pognali teste, sami pa bi se ukvarjali s čim prijetnejšim. Opozoriti pa moram, da avtomatskega testiranja nikakor ne gre razumeti na tak način in ga jemati zlahka.

4.2 Odpravimo klasični argument za avtomatizacijo testiranja

»Avtomatski testi izvajajo številne operacije brez intervencije človeka, kar zmanjša človeške napake in privede do hitrejših rezultatov. Večina produktov zahteva večkratno izvajanje testnih scenarijev. Posledica tega je, da avtomatizacija testiranja v določenem časovnem obdobju občutno zmanjša stroške dela. Običajno se podjetju taka investicija povrne že po dveh ali treh ciklih izvajanja avtomatskega testiranja.« [24]

Ta navedba izvira iz tehnične dokumentacije o avtomatizaciji testiranja, ki jo je objavilo eno izmed vodilnih podjetij v prodaji testnih orodij. Podobne izjave lahko najdemo v oglasih in dokumentaciji za večino plačljivih orodij za testiranje, včasih pa vsebujejo še impresivne grafe. Bistvo vsega pa je eno: računalniki so hitrejši, cenejši in bolj zanesljivi kot ljudje, zato avtomatizirajte. Ta trditev temelji na številnih nepremišljenih predpostavkah. V nadaljevanju si bomo ogledali nekaj izmed njih.

4.2.1 Prva predpostavka: testiranje je zaporedje operacij

Pravilnejše razmišljanje o testiranju je, da gre za zaporedje interakcij, ki se prepletajo z ocenjevanjem oziroma vrednotenjem. Nekatere interakcije so enostavne in predvidljive, druge kompleksne, dvoumne in ohlapne. Pogosto je priporočljivo ustvariti koncept zaporedja interakcij, ki sestavljajo določen testni scenarij, vendar pa zmanjšanje testiranja na raven rutiniranih akcij privede do slabih rezultatov testiranja.

Ročno testiranje, po drugi strani, pa je proces, ki je sposoben hitrega prilagajanja spremembam in uspešnega kosa s kompleksnimi težavami. Ljudje so sposobni na prvi pogled zaznati stotine težavnih vzorcev in prepoznati neškodljive anomalije. Morda se ljudje niti ne zavedajo vseh operacij, ki jih nezavedno počnejo, v zaporedju operacij pa morajo biti vsa vrednotenja skrbno načrtovana. Testiranje je lahko videti samo kot zaporedje operacij, vendar je dobro testiranje interaktivni kognitivni proces. Prav iz tega razloga je avtomatizacija najbolj uporabna v ozkem spektru testiranja in ne v večini procesov testiranja.

Če želite avtomatizirati celotni spekter testiranja, boste bržkone za to porabili veliko denarja in časa. Rezultat tega bodo relativno slabi testni scenariji, ki spregledajo veliko pomembnih napak, odkrijejo pa veliko takšnih, pri katerih gre zgolj za nepričakovano pravilno vedenje.

4.2.2 Druga predpostavka: testiranje pomeni izvajanje istih operacij znova in znova

Ko določen testni scenarij enkrat izvedemo in ne najdemo nobene napake, obstaja majhna verjetnost, da bomo s tem scenarijem odkrili kakšno napako, razen v primeru, da se pojavi nova. Pri ročnem izvajanju testnih scenarijev običajno prihaja do manjših sprememb v testnih scenarijih, kar povečuje verjetnost odkritja napak, tako novih kot starih.

Spremenljivost je velika prednost ročnega testiranja v primerjavi z avtomatskim testiranjem. V enem izmed podjetij, kjer je Bach delal, so vodili statistiko, ali so napake najdene z avtomatskim testiranjem ali ročnim. Rezultati so pokazali, da čez 80 % napak odkrijejo s pomočjo ročnega testiranja, kljub večletnim vlaganjem v avtomatizacijo testiranja. To so si razlagali tako, da so ročni testi bolj spremenljivi in bolj usmerjeni v nove funkcionalnosti in specifična področja sprememb, kjer se običajno skriva največ napak in obstaja večja verjetnost njihovega odkritja.

Pogosto ponavljanje testiranja lahko zmanjša možnosti odkrivanja pomembnejših napak – iz istega razloga stopanje po stopinjah nekoga drugega zmanjšuje možnost, da stopimo na mino.

4.2.3 Tretja predpostavka: testne akcije lahko avtomatiziramo

Nekatere naloge, ki so enostavne za ljudi, so zahtevne za računalnike. Verjetno najzahtevnejši del avtomatizacije je interpretacija rezultatov izvedenih testnih scenarijev. Pri grafičnih vmesnikih je z avtomatizacijo zelo težko zaznati pomembne napake, obenem pa ignorirati nepomembne.

Velika stopnja negotovosti in sprememb otežuje avtomatizacijo v tipičnem inovativnem projektu razvoja programske opreme. V tržno usmerjenih projektih razvoja programske opreme je običajno uporabljen agilen razvojni pristop, ki kar kliče po spremembah na programski opremi vse do pozne razvojne faze. To dejstvo, skupaj z značilno odsotnostjo celovite in natančne specifikacije programske opreme, je za uvajanje avtomatizacije kot vožnja po brezpotju z družinskim avtomobilom: to lahko storite, vendar morate voziti počasi, veliko bo popraviljanja smeri in morda se boste tudi zataknili.

Četudi imamo specifično zaporedje operacij, ki bi ga načeloma lahko avtomatizirali, lahko to naredimo le, če imamo primerno orodje. Dobre informacije o orodjih je težko najti, zraven tega pa je skoraj nemogoče oceniti najbolj kritične vidike orodij za regresijsko testiranje, če ne ustvarimo ali pregledamo industrijskega testnega načrta in ga preizkusimo na orodju. V nadaljevanju si bomo ogledali nekaj najpomembnejših dejavnikov, na katere moramo biti pozorni pri izbiri testnega orodja. Dobro je biti pozoren na to, koliko orodij nikoli ne bi mogli oceniti samo na podlagi podrobnega pregleda uporabniških navodil ali gledanja predstavitvenega posnetka.

Pomembni dejavniki pri izbiri testnega orodja:

- funkcionalnost: ali ima orodje vse kritične funkcionalnosti, ki jih potrebujemo, še posebej na področju rezultatov in upravljanja testnih scenarijev in načrtov,
- zanesljivost: ali lahko orodje v daljših obdobjih deluje brez težav ali je polno napak – veliko orodij za testiranje je razvitih v majhnih podjetjih, ki slabo preizkusijo lasten izdelek,
- kapaciteta: ali orodje dobro deluje tudi v proizvodnem okolju, ali lahko obvladuje veliko število testnih scenarijev, ki se izvajajo več ur ali več dni in vključujejo veliko skript,
- učljivost: ali se lahko upravljanja orodja naučimo v kratkem času, ali so na voljo usposabljanja ali knjige, ki pripomorejo k uvajanju,
- upravljanje: ali je uporaba orodja enostavna za uporabo in ali je nagnjeno k napakam uporabnika,
- zmogljivost: ali je orodje dovolj hitro, da bo omogočalo znatne prihranke pri razvoju testnih scenarijev in času izvajanja v primerjavi z ročnim testiranjem,
- združljivost: ali orodje deluje z določeno tehnologijo, ki jo moramo testirati, in
- nevsiljivost: kako dobro orodje simulira dejanskega uporabnika, ali je obnašanje aplikacije, ki jo testiramo, enako pri uporabi avtomatizacije in brez nje.

4.2.4 Četrta predpostavka: avtomatizirani testni scenarij je hitrejši, saj ne potrebuje človeškega posredovanja

Vsa orodja zahtevajo človeško posredovanje, zlasti za analizo rezultatov in odpravljanje napak v testnih scenarijih. Prav tako je lahko presenetljivo težko izvesti zahtevni testni scenarij brez težav. Skupni imenovalci težav so spremembe v programski opremi, ki jo testiramo, težave s spominom, težave z datotečnim sistemom, omrežjem ter napakami v samem testnem orodju.

4.2.5 Peta predpostavka: avtomatizacija zmanjša človeške napake

Drži, nekaj napak je zmanjšanih, vendar predvsem tistih, ki jih ljudje delajo, ko jim je naloženo opravljanje dolgega seznama preprostih miselnih nalog. Istočasno se okrepijo ostale napake. Vsaka napaka, ki jo spregledamo med ustvarjanjem glavnih primerjalnih datotek, bo ostala neopažena pri vseh naslednjih izvajanjih. Prav tako bi lahko med nadzorom ponesreči deaktivirali stotine testov.

Nemalo podjetij pride do ugotovitve, da bi tovrstne težave najbolj učinkovito zmanjšali z rednim vzdrževanjem in s testiranjem avtomatskega okolja oziroma z uporabo ročnega testiranja. Pri slednjem jih je namreč veliko lažje odkriti in tudi manevrirati z osnovnimi dokumenti za upravljanje testiranja, s poročili in z razvojnimi standardi.

4.2.6 Šesta predpostavka: primerjava stroškov in koristi avtomatskega in ročnega testiranja

Resnica je, da sta ročno in avtomatsko testiranje dejansko dva različna procesa, ne le dva različna načina za izvajanje istega procesa. Njuni dinamiki se razlikujeta in prav tako tudi napake, ki naj bi jih odkrili. Posledično to pomeni, da je neposredna primerjava v ceni ali številu odkritih napak nesmiselna. Poleg tega pa je v pravo primerjavo vključenih toliko posebnosti in skritih dejavnikov, da je k primerjavi najbolje pristopiti skozi serijo realnih projektov. Zato je priporočljivo, da se avtomatizacijo testiranja izvaja kot del večdelnega poskusa testne strategije, ne pa kot aktivnost, ki dominira procesu ali je celo samozadostna.

4.2.7 Sedma predpostavka: avtomatizacija omogoča bistvene prihranke pri delovni sili

Velikokrat podjetja pričakujejo, da bodo stroški delovne sile bistveno upadli že po dveh, treh izvedbah avtomatskega testiranja. Od kod jim ta podatek, ni prav jasno, vsekakor pa ne drži.

Strošek avtomatskega testiranja je sestavljen iz več delov:

- strošek razvoja avtomatizacije,
- strošek upravljanja avtomatskih testov,
- strošek vzdrževanja avtomatizacije ob spremembi programske opreme in
- strošek vseh dodatnih novih nalog, ki jih zahteva avtomatizacija.

Te stroške je treba primerjati s stroški dela pri ročnem testiranju, kjer bo verjetno prevladalo slednje. Pravzaprav je težko najti primer avtomatizacije testiranja, pri katerem bi ročni preizkuševalci posledično imeli manj dela.

Kako se ti stroški izidejo, je odvisno od mnogih dejavnikov, vključno s tehnologijo, ki jo testiramo, z orodji, ki jih uporabljamo, z znanji razvojnih inženirjev in s kakovostjo nabora testnih scenarijev.

Pisanje ene same testne skripte ne predstavlja nujno veliko dela, ustvarjanje koristnega testa pa lahko traja tedne ali celo mesece. Enako velja za proces odločanja, katero orodje kupiti, katere teste avtomatizirati, kako avtomatizacijo spremljati skozi celoten proces in seveda za

proces učenja uporabe orodja in dejanskega pisanja testnih skript. Previden pristop k temu procesu (torej tak, da se bo odrazil v uporabni programski opremi) pogosto zahteva mesece trdega dela ali več, če je razvijalec avtomatizacije neizkušen na področju avtomatizacije testiranja ali pri posebnostih orodja in tehnologije.

Kaj pa stalni stroški vzdrževanja? Večina analiz stroškov avtomatizacije testiranja popolnoma ignorira posebne nove naloge, ki jih je treba izvajati zaradi avtomatizacije:

- testni scenariji morajo biti natančno dokumentirani,
- avtomatizacija sama mora biti testirana in dokumentirana,
- ob vsakem izvajanju nabora testnih scenarijev mora nekdo pregledati rezultate, da bi ločil resnične napake od lažnih,
- radikalne spremembe, ki naj bi jih uvedli na testirani programski opremi, je treba pregledati in oceniti njihov vpliv na nabor testnih scenarijev, morda je treba tudi napisati novo testno kodo, ki bo prilagojena spremembam,
- če je nabor testnih scenarijev dan na vpogled tudi drugim inženirjem, je treba izvesti sestanke za usklajevanje razvoja, vzdrževanja in operiranja tega, ter nenazadnje,
- prestatati je treba tudi marsikakšen glavobol zaradi prenašanja testov, če je testirana programska oprema prenesena v novo programsko okolje ali pa celo v novo različico istega programskega okolja.

Te nove naloge predstavljajo pomembne izzive pri preizkuševalčevem delu. V večini podjetij, kjer je Bach delal, so na neki točki želeli, da preizkuševalci polovico svojega delovnega časa namenijo avtomatizaciji, na koncu pa so to namero vsi opustili in prišli do zaključka, da je bolje imeti polno zaposlene inženirje ali skupino inženirjev, ki bodo skrbeli za avtomatizacijo. Pisanje testne kode in izvajanje interaktivnega ročnega testiranja sta namreč tako različni aktivnosti, da se bo posameznik, ki naj bi počel oboje, slej ko prej osredotočil na eno in zanemaril drugo. Razvoj avtomatizacije je dejansko razvoj programske opreme, kar zahteva določeno znanje oziroma talent za programiranje. Nekateri inženirji preprosto ne zmorejo obojega. Kakor koli že, podjetja, ki imajo z avtomatizacijo testiranja resne načrte, bodo na koncu imela popolno testno ekipo, ki bo dobro upravljala tudi avtomatsko testiranje, in na ta strošek je treba računati.

4.2.8 Osma predpostavka: avtomatizacija ne bo imela vpliva na testni projekt

To je najbolj pereča težava, s katero se soočimo v procesu poskusa uvajanja avtomatizacije. Nevarno je avtomatizirati nekaj, česar ne razumemo. Če nam testna strategija ni popolnoma jasna, preden začnemo uvajati avtomatizacijo, bo njen rezultat velika zmeda ali pa testna koda, ki je nihče ne bo razumel. Razvojnim inženirjem so običajno dodeljene nove naloge in njihovo delo prevzame nekdo drug, ki vzdržuje sistem avtomatizacije. Vzdrževalci se bojijo kar koli odstraniti iz obstoječih testnih scenarijev, čeprav se stvari zdijo nepomembne, saj bi lahko zavrgli kaj pomembnega. Nabor testnih scenarijev tako še naprej sprejema nove teste ter postaja vedno bolj nerazumljiv in neobvladljiv. Nihče

ne ve, kaj nabor dejansko testira, še manj verjetno pa je, da se bo kdo ukvarjal s tem, da bi to ugotovil.

Dolgo časa se nihče ne zaveda, da ta težava obstaja, dokler nekega dne razvojni upravitelj ne povpraša, kaj testni nabor testira in česa ne, pa mu na vprašanje nihče ne zna odgovoriti. Še huje je, če pride do popolnega sesutja sistema avtomatizacije in nihče ne zna vzpostaviti ponovnega delovanja.

Tudi strategija ročnega testiranja se lahko znajde v pravi zmedi, vendar pa ko teste sestavljamo dinamično, na podlagi relativno majhnega sestava dokumentov, je veliko lažje pregledati in prilagoditi strategijo. Ročno testiranje je resda počasnejše, vendar veliko bolj prilagodljivo in se lažje spopade s kaosom nepopolnih in spreminjajočih se izdelkov in zahtev.

4.3 Smotrni pristop k avtomatizaciji

Kljub pomislekom, izraženim zgoraj, avtomatizacije testiranja ne gre zaničevati. Tako kot je mogoče zagotoviti kakovostno programsko opremo, je mogoče zagotoviti tudi kakovostno avtomatizacijo testiranja, vendar pa moramo biti pri tem previdni. Pot do tega cilja je strma in polna pasti, zato moramo biti pozorni na nekatere ključne principe:

- ohraniti moramo jasno razliko med avtomatizacijo in procesom, ki ga ta avtomatizira. Testni proces bi moral biti v takšni obliki, da ga zlahka pregledamo in se povezuje z avtomatizacijo,
- o avtomatizaciji moramo razmišljati kot o osnovnem naboru testnih scenarijev, ki bo dopolnjeval ročno testiranje,
- previdni moramo biti pri izboru testnega orodja. Dobro je pridobiti mnenja in izkušnje drugih preizkuševalcev in podjetij, po možnosti pa tudi preizkusiti poskusne različice razpoložljivih orodij, preden se odločimo za nakup,
- dobro moramo razmisliti o razliki v izkoristku pri upravljanju testiranja v primeru nakupa in lastne izdelave. Dober sistem za upravljanje testiranja lahko veliko pripomore k bolj preglednemu in lažje vzdrževanemu naboru testnih scenarijev in
- pripraviti moramo poročilo, ki bo za vsako izvajanje nabora testnih scenarijev vključevalo podatke o tem, kateri testni scenariji so bili uspešni in kateri neuspešni ter s kakšnimi napakami smo imeli opravka. Poročilo mora vsebovati tudi podatke o vseh popravilih in izboljšavah na naboru testnih scenarijev. Takšna poročila koristijo pri ugotavljanju stroškovne učinkovitosti avtomatizacije.

Pri avtomatizaciji ne gre pomišljati, saj je to velika zamisel. Da bi postala tudi dobra naložba, moramo najprej razmisliti o testiranju in šele nato o avtomatizaciji. Če je testiranje pot za pridobitev ključnih informacij o kakovosti programske opreme, je avtomatizacija le ena izmed možnosti na tej poti. Tega sicer ni mogoče zaznati iz oglasov, vendar je to le ena izmed strategij, ki podpira učinkovito testiranje programske opreme.

5. ORODJA ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI

Ko se podjetje znajde pred nakupom novega orodja za avtomatizacijo testiranja, se mora najprej seznaniti s ponudbo na trgu. Ugotoviti mora, ali obstoječa orodja izpolnjujejo njihove zahteve ali pa jih morda celo presegajo in ali so s tem v povezavi tudi cenovno ustrezna. Analiza obstoječih različic včasih pokaže, da je najbolje vztrajati pri obstoječem ročnem testiranju ali pa avtomatsko testno orodje razviti znotraj podjetja.

Preden se podjetje spusti v lasten razvoj, pa mora narediti natančno analizo svojih zahtev, sestaviti dolgoročni načrt, ki bo opredelil časovne in finančne naložbe za razvoj in izdelavo orodja, ter določiti vire za njegovo nadgradnjo. Z dobrim načrtovanjem bo ustvarilo orodje, ki bo služilo natančno njihovim potrebam in ne bo preseglo cene obstoječih izdelkov na trgu. Seveda moramo, da bi dosegli ta cilj, upoštevati potrebe celotnega podjetja in orodje izdelati na način, da bo popolnoma združljivo s tehnologijo, ki jo v podjetju že uporabljamo. [6]

Fewster pri razvoju lastnega orodja poudarja naslednje značilnosti:

- orodje je prilagojeno potrebam določenega uporabnika,
- v primeru, da ima uporabnikov sistem nizko stopnjo zmožnosti testiranja, lahko le namensko orodje do popolnosti opravi avtomatsko testiranje,
- zelo verjetno je, da je takšno orodje podprto s slabo dokumentacijo in šolanjem kadra, ki ga bo uporabljal,
- grafični vmesnik ni dovolj podrobno dodelan (avtomatsko orodje, ki ga razvijajo tehnično usmerjene osebe, pogosto ni prijazno za uporabo, saj pri njih prevladuje mnenje, da je ključnega pomena izpopolnjeno delovanje avtomatskega orodja, ne pa tudi enostavnost uporabe).

Ko razvijamo avtomatsko orodje, ni smiselno težiti k izdelavi visokokakovostnega orodja iz visokega cenovnega razreda, ki že obstaja na trgu. Razvoj takšnih orodij običajno traja nekaj let, mi pa moramo razviti orodja, od katerih bomo imeli korist čim prej. Dobro je, da k sodelovanju pritegnemo veliko izkušenih strokovnjakov s področja razvoja testnih orodij in od njih zberemo tudi informacije, ki bi nam pri delu lahko pomagale. [5]

Thirumalai opozarja na visoke začetne stroške naložbe v avtomatizacijo testiranja. Nenazadnje ni njen namen popolnoma nadomestiti ročnega testiranja, temveč delovati kot dodatna metoda v procesu testiranja in razvoja, saj testiranje in dokumentiranje avtomatskega testa traja od tri- do desetkrat dlje od priprave ročnega testa. [11]

5.1 Microsoft Test Manager 2010

Microsoft Test Manager (v nadaljevanju MTM) je del orodja Visual Studio 2010, ki se pojavlja v različicah Ultimate in Test Professional. Omogoča upravljanje celotnega procesa testiranja, tj. izdelavo testnih scenarijev in testnih načrtov, izvajanje ročnih in avtomatskih testnih scenarijev itd. [14]

Zaradi visoke cene MTM ni primeren za manjša podjetja, pa tudi ne zaradi ostalih pogojev, ki so potrebni za delovanje orodja (zahteva, npr. nameščen Team Foundation Server – TFS). Po drugi strani je zelo ustrezen za večja podjetja, ki že uporabljajo Microsoftove izdelke, kot je npr. Visual Studio (v nadaljevanju VS). MTM v kombinaciji z VS ponuja podporo celotnemu razvojnemu procesu:

- razvoj v VS,
- testiranje v MTM in
- vodenje testiranja z Application Lifecycle Management (v nadaljevanju ALM).

Takšna integracija bistveno olajša sodelovanje med ekipama razvijalcev in preizkuševalcev.

Na primer, preizkuševalec izvaja testni scenarij z uporabo MTM. Ko odkrije napako, lahko razvijalcu posreduje ključne informacije o načinu izvajanja navedenega testnega scenarija (vse ključne informacije o izvedbi testiranja sproti zbira MTM). Podatki o odkritih napakah so preizkuševalcem in posledično razvijalcem na voljo v zelo različnih oblikah, kar pomeni, da lahko v določenem trenutku razvijalcu ponudijo le seznam napak ali pa njihov podroben opis, morda celo videoposnetek ipd. Takšna prilagodljivost zagotovo omogoča hitrejše in bolj učinkovito odpravljanje napak.

Poleg vsega navedenega ima MTM skupaj z ostalimi moduli, ki so del Visual Studia 2010 in ostalih komponent (TFS, ALM), tudi ogromno drugih uporabnih lastnosti, ki so pomembne v celotnem razvojnem ciklu programske opreme, vendar se zaradi preobsežnosti na tem mestu v to ne morem poglobljati.

5.2 Software Testing Automation Framework (STAF)

STAF je odprtokodno ogrodje, ki omogoča izvajanje testiranja na več operacijskih sistemih (Windows, Linux, Mac OS, HP-UX idr.). Testne skripte lahko razvijamo v različnih programskih jezikih. Ogrodje STAF predstavlja temelje in poenostavitev za razvoj lastne rešitve avtomatizacije na višji ravni.

Prednosti, ki jih prinaša ogrodje STAF:

- krajši pripravljalni čas,
- skrajšan čas testiranja in
- večji izbor operacijskih sistemov in programskih jezikov.

Ogrodje vsebuje preprostega agenta, ki ga lahko poganjamo na več napravah, ki so del STAF-ovega testnega okolja. Potem lahko vsaka naprava požene storitve na kateri koli drugi napravi, ob upoštevanju stopnje zaupnosti. V praksi to pomeni, da se bo ena naprava obnašala kot nekakšen vodja testa, ki usklajuje izvajanje testov z dodeljevanjem del odjemalcem.

STAX je izvajalni pogon, ki nudi pomoč pri avtomatizaciji distribucije, izvajanja in analize rezultatov testnih scenarijev. Nudi zmogljivo aplikacijo za spremljanje, ki omogoča interakcijo in spremljanje napredka testiranja. STAX močno olajša dodeljevanje del med testne

odjemalce in zbiranje ter beleženje testnih rezultatov. Glavne funkcionalnosti STAX pogona so:

- vzporedno izvajanje,
- uporabniško določena razdrobljenost nadzora izvajanja,
- podpora ugnезdenim testnim scenarijem,
- nadzor nad dolžino izvajalnega časa,
- uvoz modulov med trajanjem izvajanja in
- možnosti razširitve.

S pomočjo teh funkcionalnosti lahko zgradimo napredne skripte za avtomatizacijo celotnega testnega okolja ter zagotavljamo najvišjo učinkovitost in nadzor.

STAX ponuja tudi nadzorno orodje GUI, poimenovano STAX Job Monitor, ki ga običajno poženemo na nadzorni napravi. Orodje je izdelano v Javi in prikazuje procese, ki se izvajajo med delom, kot tudi podatke o testnem primeru (ime testa, status opravljenega oz. neopravljenega, trajanje) za vsak test v programskem naboru.

6. INTERNO OGRODJE ZA AVTOMATSKO IZVAJANJE TESTNIH AKTIVNOSTI

Ogrodje za avtomatsko testiranje (v nadaljevanju OAT), ki smo ga uporabljali v podjetju, je enotno ogrodje, ki je enostavno za uporabo in vključuje nabore testnih scenarijev. Namenjeno je testiranju modulov, funkcionalnemu in regresijskemu testiranju interne aplikacije podjetja. Omogoča pogoste cikle izdaje nove različice (alfa, beta, predizdaja), ki so lahko različno dolgi (od enega do nekaj tednov), krajše pripravljalne čase, krajše čase testiranja, večjo izbiro programskih okolij in večjo izbiro programskih jezikov. Namen OAT je ponovna uporaba in razširljivost, olajšanje testiranja oziroma testnega procesa, predvsem pa enostavnejše izvajanje avtomatskega testiranja. To vključuje izdelavo testnih scenarijev, upravljanje in avtomatizacijo testnega okolja ter ustvarjanje proizvodnega izkoristka.

6.1 Arhitektura in namestitve

Arhitektura OAT je generična in prilagodljiva, kar omogoča enostavno dodajanje novih testnih skript. Sestavljajo jo štirje deli, ki so prikazani na spodnji sliki. **Kontrolni del** skrbi za izvajanje procesa testiranja ne glede na platformo, na kateri izvajamo testiranje, podrobneje pa je predstavljen v podpoglavju 6.2 (opis poteka procesa testiranja). OAT je napisano v programskem jeziku Python, ostale skripte, ki so uporabljene za testiranje, pa so lahko napisane v katerem koli drugem jeziku, kot npr. Perl, shell, Python.



Slika 4: Arhitektura OAT

Del **testne skripte** vsebuje skripte, kjer so uvedene operacije, ki jih izvaja interna aplikacija. Skripte so logično razdeljene, in sicer na splošni del, kjer so skripte neodvisne od programskega okolja, ter glede na programsko okolje, za katero so namenjene (Windows, Linux, HP-UX). Testne skripte se uporabljajo v nastavitvenih datotekah s testnimi nabori.

Knjižnice so glavni del OAT. V tem delu se nahaja celotna logika ogrodja, ki opravlja naslednja opravila:

- razčlenjevanje (angl. parsing) nastavitvene datoteke system.cfg, datotek s testnimi nabori, parametrov itd.,
- nastavljanje sistemskih spremenljivk,
- zagon in izvajanje testiranja,
- preverjanje rezultatov,
- pošiljanje elektronske pošte in
- pisanje datotek z dnevniki.

Drevesna razdelitev imenikov je enaka kot pri testnih skriptah: splošni del, nato pa glede na programsko okolje.

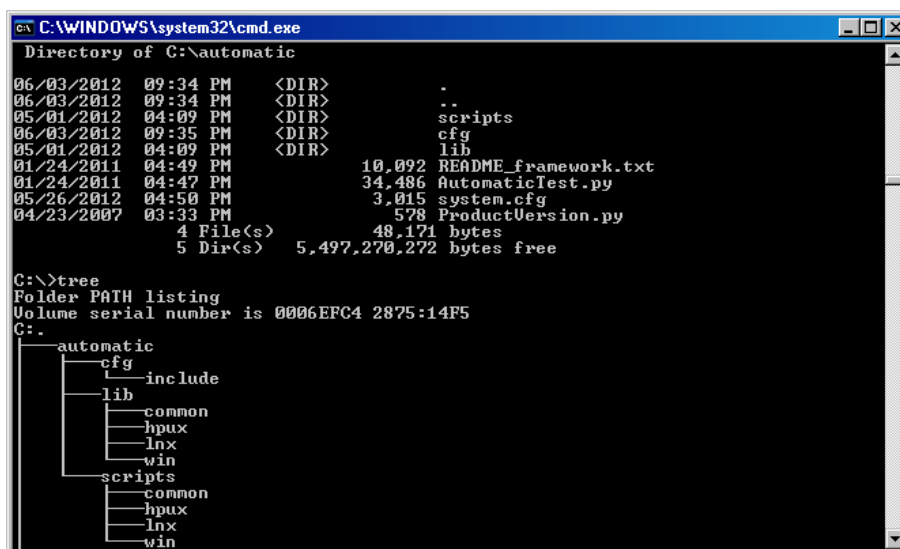
Datoteke s testnimi nabori so podrobneje predstavljene v poglavju 6.1.3.

6.1.1 Predpogoji za namestitvev

Namestitev OAT zahteva prednamestitev Python namestitvenega paketa. V primeru, da bodo izvajani testi napisani v drugem programskem jeziku, mora biti nameščen ustrezen namestitveni paket.

Namestitev sama je zelo enostavna v katerem koli programskem okolju. V okolju Windows zaženemo namestitveni paket `automatic.msi` in izberemo željeni imenik, kamor namestimo OAT. V okolju Linux zgolj odpakiramo `automatic.tar.gz` paket na željeno lokacijo.

Po namestitvi se vsebina OAT nahaja v korenskem imeniku "automatic". Drevesna struktura imenikov je prikazana na spodnji sliki, opis vsebine pa ji sledi.



Slika 5: Drevesna struktura imenikov OAT

automatic\AutomaticTest.py je glavna python skripta, ki razčlenjuje nastavitvene datoteke in skrbi za izvajanje testnih skript. Poleg tega zbira in ustvarja rezultate o izvedenih testih na podlagi vnaprej določenih izhodnih parametrov, ki jih vračajo testne skripte. Dobljene rezultate pošilja na opredeljene elektronske naslove. Poleg glavne skripte se v tem imeniku nahaja tudi nastavitvena datoteka `system.cfg`;

automatic\cfg imenik vsebuje datoteke s testnimi nabori, ki so predpogoj za začetek izvajanja testiranja. `AutomaticTest.py` skripta uporablja datoteke s testnimi nabori iz tega imenika kot parametre za izvajanje testiranja. Prebere datoteko s testnimi nabori in pošlje skripto ali izvršilno datoteko, ki je navedena v njej;

automatic\lib imenik vsebuje skripte, ki predstavljajo motor ogrodja;

automatic\logs tu se nahajajo datoteke z dnevniki, ki jih generira testna skripta. Ime datoteke sestoji iz datuma in ure, kar omogoča enostavnejše iskanje, če imamo več dnevnikov;

automatic\scripts imenik vsebuje testne skripte in ima naslednjo strukturo:

- **automatic\scripts\common** vsebuje testne skripte, ki se lahko izvajajo na vseh operacijskih sistemih in niso odvisne od specifične platforme. Te skripte imajo v imenu predpono »cmn_« za boljši pregled. Na primer: cmn_StartStop.py,
- **automatic\scripts\lnx** ta imenik vsebuje testne skripte, ki vsebujejo operacije, specifične za operacijski sistem Linux. Skripte imajo v imenu predpono »lnx_«. Primer: lnx_Status.py,
- **automatic\scripts\hpux** vsebuje testne skripte, specifične za operacijski sistem HP-UX in
- **automatic\scripts\win** vsebuje testne skripte, specifične za operacijski sistem Windows.

6.1.2 Nastavitvene datoteke

Datoteka `system.cfg` vsebuje splošne parametre, ki so potrebni za izvajanje testiranja. Ti parametri so: ime skupine, ime izdelka, elektronski naslov itd. Vsi parametri morajo imeti predpono »AT_«.

```
#####
###   Nastavitvena datoteka ogrodja za avtomatsko testiranje   ###
#####
# *****
# Ime testiranega produkta
# Možosti so:
#       AT_PRODUCT=ivd
#       AT_PRODUCT=fse
# *****
AT_PRODUCT=ivd
# *****
# Datotečna pot do mesta namestitve testnega produkta
# Primeri privzetih namestitvenih datotecnih poti:
#   Linux:
#       če je AT_PRODUCT ivd - AT_PRODUCT INSTALL PATH=/var/opt/ivd
#       če je AT_PRODUCT fse - AT_PRODUCT INSTALL PATH=/var/opt/fse
#   Windows:
#       če je AT_PRODUCT ivd - AT_PRODUCT INSTALL PATH=C:\Program
Files\<ime_stranke_1>\IVD
#       če je AT_PRODUCT fse - AT_PRODUCT INSTALL PATH= C:\Program
Files\<ime_stranke_2>\FSE
#
# *****
AT_PRODUCT_INSTALL_PATH=/var/opt/ivd
# *****
# Elektronski naslov kamor bo poslana elektronska pošta z obvestili
# AT_EMAIL - elektronski naslov kamor bodo poslana obvestila o vseh
rezultatih (uspešnih in neuspešnih)
# AT_EMAIL_ERROR - elektronski naslov kamor bodo poslana obvestila o
neuspešno izvedenih skriptah
#   AT_SMTP - ime poštne strežnika
#   AT_USER - uporabniško ime (če ga poštni strežnik zahteva)
#   AT_PASSWORD - uporabniško geslo (ce poštni strežnik zahteva identifikacijo)
# *****
AT_EMAIL=janez@novak.si
AT_EMAIL_ERROR= matej@novak.si
AT_SMTP=123.123.123.123
AT_USER=matej
AT_PASSWORD=geslo
# *****
# Datotečna pot do datoteke z dnevnikom, ki vsebuje obvestila o napakah
# Primer:
#       /var/opt/ivd/log/error.log
# *****
AT_checkLogFile=${AT_PRODUCT_INSTALL_PATH}/log/error.log
```

Slika 6: Primer sistemske nastavitvene datoteke

6.1.3 Datoteke s testnimi nabori

Datoteke s testnimi nabori so dejansko nastavitvene datoteke, kjer deklariramo testne skripte v določenih zaporedjih. Ta zaporedja predstavljajo korake testnega scenarija.

Primer testnega scenarija in datoteke s testnim naborom:

- korak 1: v imenikih *test1*, *test2* ter *test3*, ki se nahajajo v imeniku *var/opt/ivd/particija1* ustvarimo različno število datotek, velikosti 1024b, s pripono A. Za ustvarjanje uporabimo ukaz *ivd_fstest*.
- korak 2: nato ustvarimo posebne datoteke.
- korak 3: izvedemo operacijo »migracija«, ki prenese vsebino datotek iz trenutne lokacije na drugo podatkovno skladišče.
- korak 4: preverimo status »migriranih« datotek.

```
#####
###      Nastavitvena datoteka Primer_1      ###
###  Kreiraj in preveri posebne datoteke na Linux sistemu  ###
#####
*****
# Deklaracija spremenljivk
*****
$imeParticije = primer1
$mestoPriklonpneTocke = var/opt/ivd/particijal
$imeImenika1 = ${mestoPriklonpneTocke}/testsl
$imeImenika2 = ${mestoPriklonpneTocke}/tests2
$imeImenika3 = ${mestoPriklonpneTocke}/tests3
$blok = ${mestoPriklonpneTocke}/blok
$znak = ${mestoPriklonpneTocke}/znak
$cev = ${mestoPriklonpneTocke}/cev
$vticnica = ${mestoPriklonpneTocke}/vticnica
$velikostDatoteke = 1024b
$pripona = A
*****
# Seznam skript, ki bodo izvedene v tem sklopu
*****
# ID_testnega_scenarija | ime_skripte | argumenti

001 ivd_fstest --create-tree -P $imeImenika1 10 10 2 $velikostDatoteke $pripona
001 ivd_fstest --create-tree -P $imeImenika2 3 3 1 $velikostDatoteke $pripona
001 ivd_fstest --create-tree -P $imeImenika3 2 2 6 $velikostDatoteke $pripona
001 CreateSpecialFiles.py -o=chardev -p=$znak -n=5
001 CreateSpecialFiles.py -o=blockdev -p=$blok -n=5
001 CreateSpecialFiles.py -o=pipe -p=$cev -n=5
001 CreateSpecialFiles.py -o=socket -p=$vticnica -n=5
001 WaitForFilesToMigrate.py -p=$imeParticije
001 CheckSpecialFilesStatus.py -m=$mestoPriklonpneTocke
```

Slika 7: Primer datoteke s testnim naborom

Nove datoteke s testnimi nabori se morajo nahajati v imeniku *~\automatic\cfg*. Pri navajanju, katere skripte se bodo klicale, mora biti prvi parameter »check parameter«, drugi parameter ime skripte, ostalo pa parametri, ki jih zahteva testna skripta. »Check parameter« preverja trenutno stanje aplikacije, ki jo testira. Parameter ima lahko naslednje tri vrednosti:

- check + (pričakuje se, da je aplikacija aktivna),
- check – (pričakuje se, da aplikacija ni aktivna) in
- nocheck (brez preverjanja).

Če pride pri preverjanju do negativnega rezultata, se izvajanje testnega scenarija prekine. Pri pisanju datoteke s testnimi nabori za posamezni testni scenarij imamo tri načine uporabe:

- določimo lahko spremenljivke, ki jih lahko nato uporabimo, kot je prikazano v spodnjem primeru:

```
$imeParticije = datotečniSistem01 # opredelitev spremenljivke
```

```
CHECK+ MigrateReleaseRecall.py -a=migrate -p=$imeParticije # primer uporabe
```

- lahko pokličemo nastavitveno datoteko, ki vsebuje opredeljene spremenljivke:

```
include partition.cfg
```

```
CHECK+ MigrateReleaseRecall.py -a=migrate -p=$imeParticije
```

- uporabimo lahko nespremenljive (angl. hardcoded) vrednosti:

```
CHECK+ MigrateReleaseRecall.py -a=migrate -p=datotečniSistem01
```

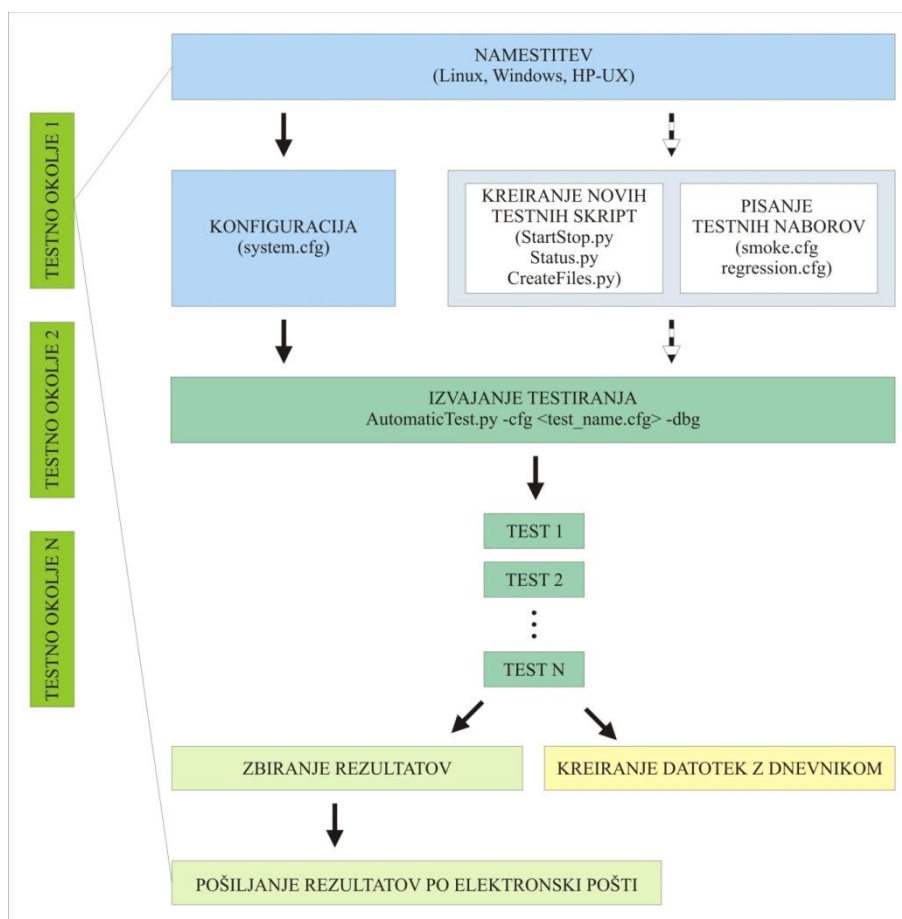
6.2 Opis poteka procesa testiranja

Testiranje zaženemo z ukazom v orodni vrstici, in sicer s klicem glavne skripte *AutomaticTest.py* in z zahtevanimi parametri. Primer zagona testiranja:

```
AutomaticTest.py --cfg »C:\automatic\cfg\regresijski_test.cfg« --dbg »detail«
```

Prvi parameter določa testni nabor, ki ga želimo uporabiti, podatki pa moramo lokacijo, kjer se ta datoteka nahaja. Drugi parameter določa raven razhroščevanja (angl. debug level).

Datoteke s testnimi nabori se nahajajo v datotečnem imeniku *~\automatic\cfg*, njihova podrobnejša zgradba pa je predstavljena v podpoglavju 6.1.3. V teh datotekah so navedene testne skripte. Testne skripte predstavljajo posamezne operacije, npr. zagon in zaustavitev aplikacije, v celoti pa predstavljajo neko logično enoto, to je testni scenarij (npr. zagon aplikacije, izvajanje drugih operacij, preverjanje rezultatov, zaustavitev aplikacije). Ko so vse testne skripte izvedene, se ustvarijo rezultati testiranja, ki se pošljejo na elektronske naslove, določene v nastavitveni datoteki *system.cfg*. Za prepoznavanje in lažje odkrivanje napak, ki smo jih odkrili med testiranjem, imamo na voljo datoteke z dnevniki, ki vsebujejo podrobne informacije o poteku dogajanja.



Slika 8: Procesni model poteka testiranja

Poleg uporabe testnih skript OAT omogoča uporabo in izvajanje raznih drugih orodij, izvršilnih datotek ali skript (npr. fctest.exe (interno orodje), test_script.sh.). V nastavitveni datoteki testnih naborov lahko navedemo ta orodja in jih nato uporabimo v testnem scenariju.

6.3 Predlog izboljšave

Predloga za izboljšavo oziroma nadgradnjo OAT izhajata iz potreb podjetja. Po nekajmesečni uporabi ogrodja na projektu ter poznejši analizi procesa testiranja na njem smo v podjetju prišli do ugotovitve, da bi bilo ogrodju smiselno dodati naslednji izboljšavi.

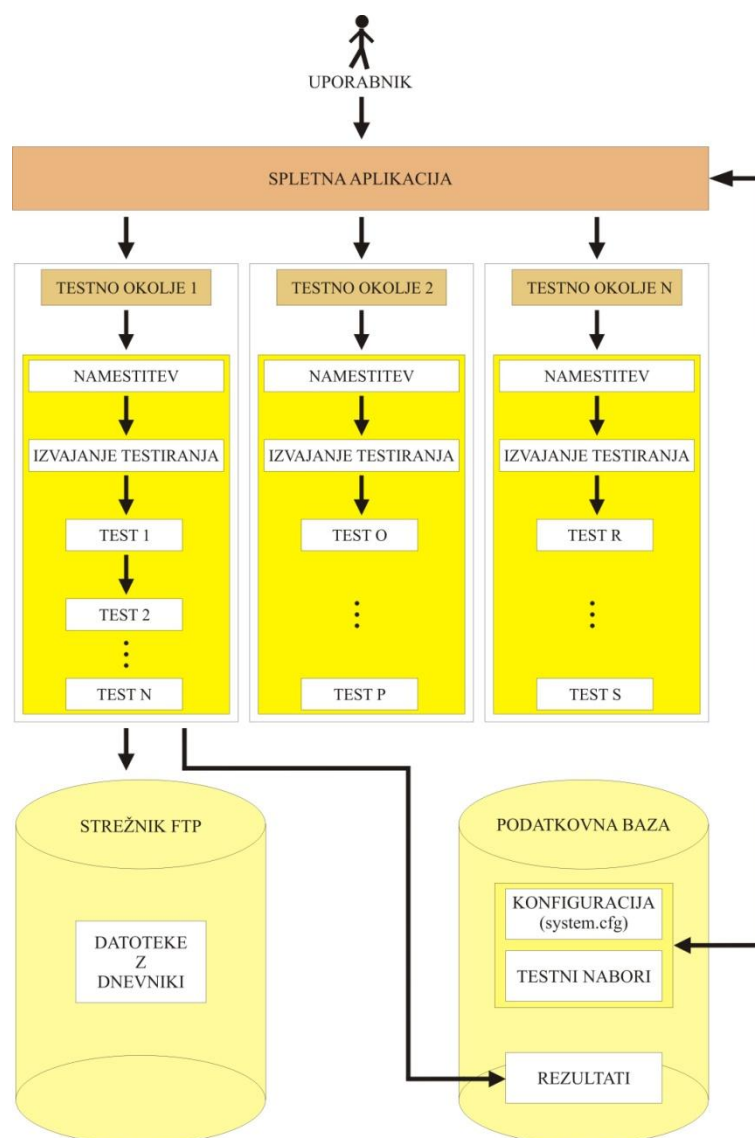
6.3.1 Prvi predlog izboljšave

Ogrodju bi dodali podatkovno zbirko, ki bi omogočala zbiranje rezultatov in samodejno osveževanje rezultatov izvedenih testnih scenarijev. Na podlagi zbranih rezultatov bi lahko izdelali statistiko testiranja (število uspešno in neuspešno izvedenih testnih scenarijev, pokritost testnega načrta v odstotkih). Poleg rezultatov testiranja bi v podatkovni zbirki hranili vsebino nastavitvene datoteke system.cfg. Prednost tega bi se odrazila v primeru, ko bi izvajali testiranje v več različnih testnih okoljih, pa ne bi bilo treba na vsakem testnem okolju vnašati podatkov v datoteko system.cfg, temveč bi te informacije že ob namestitvi OAT pridobili iz podatkovne zbirke.

Prav tako bi v podatkovni zbirki hranili datoteke s testnimi nabori. Prednost tega načina je v tem, da bi se vse datoteke s testnimi nabori nahajale na enem mestu. To je pomembno v primeru, da imamo več oseb, ki izvajajo testiranje in so v stiku z navedenimi datotekami. Če spreminjamo vsebino datoteke s testnimi nabori, naredimo spremembo na enem mestu, ki je posledično vidna vsem udeležencem.

6.3.2 Drugi predlog izboljšave

Ta predlog vključuje spletno aplikacijo, ki bi predstavljala povezavo s podatkovno zbirko. Omogočala bi vnašanje podatkov za nastavitveno datoteko system.cfg, pregled testnih okolij, ki so na voljo za testiranje in zagon testiranja na določenih testnih okoljih, ter pošiljanje in zbiranje datotek z dnevniki na enotni lokaciji (npr. strežnik FTP). To bi olajšalo pregledovanje dnevnikov, saj ne bi bilo treba poiskati okolja, čigar dnevnik želimo pregledati, ter se nato povezati v izbrano okolje in poiskati ustrezni dnevnik.



Slika 9: Procesni model poteka testiranja z upoštevanjem predlogov izboljšav

7. SKLEPNE UGOTOVITVE

Pri ustvarjanju nove programske opreme ali pri posodabljanju obstoječe podjetja običajno namenijo veliko pozornosti njenemu razvoju. V razvojni ekipi lahko najdemo najboljše inženirje z najboljšimi pogoji dela, na žalost pa se prevečkrat pozablja na proces testiranja in testno ekipo pri načrtovanju novega izdelka.

Testiranje delovanja novega oziroma posodobljenega izdelka bi moralo predstavljati enakovreden del njegovemu razvoju. To pomeni, da moramo testiranju nameniti ravno toliko poudarka kot samemu razvoju. Ključno je, da vodstvo podjetja ne usmeri vseh moči (denarja, najboljšega kadra in časa) v razvoj programske opreme, temveč da te moči porazdeli med razvojno in testno ekipo.

Pri sprejemanju odločitve za testiranje je podjetje kmalu postavljeno pred izbiro ročnega in avtomatskega testiranja. Vsako ima svoje prednosti in slabosti, zanimivo pa je, da, kot pogosto v življenju, tudi tukaj nazadnje pridemo do spoznanja, da nam najvišjo kakovost zagotavlja kombinacija obeh.

Med delom na tej diplomski nalogi sem spoznal, da je izrednega pomena, da testiranju ne namenimo premalo pozornosti, časa in denarja. Pri samem zastavljanju novega projekta (razvoj nove programske opreme ali posodabljanje obstoječe) moramo njegovemu razvoju in testiranju nameniti približno enako usposobljene kadre. Razvijalci bodo namreč proizvedli dobro opremo, ki jo bodo dobri preizkuševalci dobro preizkusili. Skupaj bodo odkrili možnosti za izboljšave in lahko bistveno izboljšali kakovost programske opreme. Pomembno je, da obema ekipama namenimo tudi sorazmerno količino sredstev. Preizkuševalci, ki bodo slabo plačani za svoje delo in tudi sicer v podjetju videni kot le neko nujno zlo, bodo svoje delo opravljali slabše: zanj ne bodo motivirani, kar pa je ključnega pomena pri preizkušanju in razmišljanju o morebitnih pomanjkljivostih programske opreme. Nenazadnje pa je pomemben tudi čas. Običajno je, da je bistveno večja količina časa v procesu ustvarjanja programske opreme namenjena razvoju kot pa samemu testiranju, vendar pa je prav to lahko razlog za slabo kakovost končnega izdelka. Če testna ekipa nima dovolj časa za izvedbo poglobljenega testiranja, namesto tega pa izvede nekakšno testiranje, ki je le samo sebi namen, so tudi rezultati dela te ekipe slabi.

Po tem ko se podjetje odloči testiranju nameniti določena sredstva, čas in kadre, ima načeloma na izbiro dve vrsti procesa testiranja: ročno in avtomatsko. Oboje prinaša tako koristi kot tudi slabosti, zato je treba odločitev dobro pretehtati. Omeniti velja, da lahko ročno testiranje na videz deluje dražje od avtomatskega zaradi večjih potreb po kadrih, vendar pa tudi kakovostno avtomatsko testiranje zahteva vodstvo dobrih inženirjev. V primeru, da se podjetje odloči za avtomatizacijo testiranja (delno ali popolno), mora v naslednjem koraku izbrati ustrezno orodje za svoje potrebe. V diplomskem delu sem se odločil opisati le dve izmed njih, zavedati pa se moramo, da je seznam avtomatskih orodij iz dneva v dan daljši.

V podjetju, kjer sem si nabiral izkušnje, smo za testiranje uporabljali ogrodje za avtomatsko testiranje. V šestem poglavju diplomske naloge sem ogrodje predstavil in opisal njegovo delovanje ter predlagal možne izboljšave zanj.

Na splošno velja, da lahko podjetje v iskanju ustreznega avtomatskega testnega orodja vedno uporabi izdelavo svojega ogrodja, s čimer bi si zagotovilo pripomoček, popolnoma prilagojen svojim potrebam. Vsekakor pa je pred sprejetjem take odločitve treba razmisliti o njenih posledicah.

8. SEZNAM UPORABLJENIH VIROV

- [1] F. Solina, *Projektno vodenje razvoja programske opreme*, Ljubljana: Univerza v Ljubljani, 1997.
- [2] K. Jelnikar, *Avtomatsko funkcionalno testiranje programske opreme*, Ljubljana: Univerza v Ljubljani, 2009.
- [3] S. Kodarin, *Avtomatizacija testiranja programske opreme*, Ljubljana: Univerza v Ljubljani, 2011.
- [4] V. Ljevaja, *Evalvacija orodij za avtomatsko testiranje telekomunikacijske opreme*, Ljubljana: Univerza v Ljubljani, 2005.
- [5] M. Fewster, *Software Test Automation: Effective User of Test Execution Tools*, UK, 1999.
- [6] E. Dustin, *Automated Software Testing: Introduction, Management and Performance*, New York, 1999.
- [7] Automated Tests Should Find Bugs. Vir na spletu:
<http://dorothygraham.blogspot.com/2010/05/automated-tests-should-find-bugs-no.html>
- [8] Best Practices for implementing Automated Functional Testing Solutions. Vir na spletu:
http://www.xlsoft.com/en/products/automatedqa/products/testcomplete/Automated_Testing_Whitepaper_BP_Functional_Testing.pdf
- [9] Black-box Testing. Vir na spletu:
<http://softwaretestingfundamentals.com/black-box-testing/>
- [10] Classic Testing Mistakes. Vir na spletu:
<http://www.exampler.com/testing-com/writings/classic/mistakes.html>
- [11] Cost Effective Automated Testing. Vir na spletu:
http://www.stickyminds.com/sitewide.asp?sid=1&sidx=1&sqry=*Z%28SM%29*J%28MIXED%29*R%28relevance%29*K%28simplesite%29*F%28Thirumalai%29*&sopp=10&ObjectId=6470&Function=DETAILBROWSE&ObjectType=ART
- [12] Does Anyone Know Good Automation? Vir na spletu:
http://www.linkedin.com/groups/Does-anyone-know-good-automation-86204.S.107986970?view=&srctype=discussedNews&gid=86204&item=107986970&type=member&trk=eml-anet_dig-b_pd-ttl-cn&ut=3z2nyDQAXiKlc1
- [13] Information Engineering. Vir na spletu:
http://en.wikipedia.org/wiki/Information_engineering
- [14] Introducing Microsoft Test and Lab Manager. Vir na spletu:
<http://visualstudiomagazine.com/articles/2010/01/19/introducing-microsoft-test.aspx>
- [15] Managers Know a Lot About Testing. Vir na spletu:
http://www.linkedin.com/groups/Managers-know-lot-about-test-86204.S.88853737?view=&srctype=discussedNews&gid=86204&item=88853737&type=member&trk=eml-anet_dig-b_pd-ttl-cn&ut=193SzIW7IAJ541
- [16] Manual vs. Automated Testing, Again. Vir na spletu:
http://blogs.msdn.com/b/james_whittaker/archive/2008/10/29/manual-v-automated-testing-again.aspx

- [17] Software Testing Life-cycle. Vir na spletu:
<http://soft-engineering.blogspot.com/2010/12/software-testing-life-cycle.html>
<http://discuss.itacumens.com/index.php?topic=8130.0>
- [18] Software Testing Life-cycle. Vir na spletu:
http://en.wikipedia.org/wiki/Software_testing_life_cycle
- [19] Software Testing. Vir na spletu:
http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/
- [20] Splošen koncept načrtovanja primerov. Vir na spletu:
http://saturn.uni-mb.si/~razno/vv_iii/Splošen_koncept_nacrtovanja-testnih-primerov.pdf
- [21] Test Automation Architectures: Planning for Test Automation. Vir na spletu:
<http://www.softwarequalitymethods.com/Papers/autoarch.pdf>
- [22] Test Automation Framework. Vir na spletu:
<http://www.qatestingtools.com/auto-article-18>
- [23] Test Automation. Vir na spletu:
http://en.wikipedia.org/wiki/Test_automation
- [24] Test Automation: Snake Oil. Vir na spletu:
http://www.satisfice.com/articles/test_automation_snake_oil.pdf
- [25] Test Automation: When, how and how much. Vir na spletu:
<http://searchsoftwarequality.techtarget.com/tip/Test-automation-When-how-and-how-much>
- [26] The Product LifeCycle for Automation Software Test. Vir na spletu:
<http://forum.onestoptesting.com>
- [27] The Top Five Software Testing Problems and How to Avoid Them. Vir na spletu:
http://www.edn.com/article/503158-The_top_five_software_testing_problems_and_how_to_avoid_them.php
- [28] What is software testing. Vir na spletu:
http://bazman.tripod.com/what_testing.html
- [29] What Percent of Software Testing should be Automated? Vir na spletu:
http://www.linkedin.com/groups/What-percent-software-testing-should-86204.S.97226625?view=&srctype=discussedNews&gid=86204&item=97226625&type=member&trk=eml-anet_dig-b_pd-ttl-cn&ut=0m8J1RNdkafRc1
- [30] When to Use Manual vs Automated Software Testing Tools. Vir na spletu:
<http://searchsoftwarequality.techtarget.com/feature/When-to-use-manual-vs-automated-software-testing-tools>
- [31] White-box Testing. Vir na spletu:
http://en.wikipedia.org/wiki/White-box_testing
- [32] Interna literatura TopIT d.o.o.