Računalni alati i tehnike za upravljanje i kontrolu kvalitete u testiranju softvera

Drempetić, Leo

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
NASTAVNIČKI SMJER INFORMATIKE
Ak. god. 2022./2023.

Leo Drempetić

Računalni alati i tehnike za upravljanje i kontrolu kvalitete u testiranju softvera

Diplomski rad

Mentor:
doc. dr. sc. Ivan Dunđer

Zagreb, veljača 2024.

Izjava o akademskoj čestitosti

Izjavljujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Lea Drempetić

(potpis)

Sadržaj

1. Uvod

Na početku ovog uvodnog dijela valja istaknuti da je tržište rada u neprekidnom razvoju i zahtijeva poznavanje najnovijih tehnologija. Obrazovni sustav bi trebao pratiti taj razvoj i postavljati mladima temelje za razumijevanje procesa, usvajanje znanja i navika te pripreme za uspješnu budućnost.

Poznato je da već sada na tržištu postoje razni alati i tehnologije poput umjetne inteligencije (eng. Artificial Intelligence, AI), koji nedvojbeno mogu olakšati rad i učenje, ali istovremeno mogu stvoriti niz nepredviđenih problema. Te probleme je ponekad teško razumjeti bez poznavanja rada te vrste tehnologije.

U ovom radu izlaže se pregled računalnih alata i tehnika za upravljanje i kontrolu kvalitete koji se koriste tijekom testiranja softverskih rješenja.

U drugom poglavlju obrađene su glavne razlike između Osiguranja kvalitete (eng. Quality Assurance, QA) i Kontrole kvalitete (eng. Quality Control, QC) te tehnike za osiguravanje standarda za kontrolu kvalitete prilikom testiranja softverskih proizvoda.

Treće i četvrto poglavlje donosi važnost uloga i odgovornosti tijekom razvojnog procesa za kontrolu kvalitete. Moramo uočiti razlike u odgovornosti kada različiti razvojni timovi sudjeluju u osiguranju kvalitete tijekom cijelog razvojno-životnog ciklusa (QA) ili kada je za to odgovoran samo tim za testiranje softverskog testnog ciklusa (QC). Navedena poglavlja također obuhvaćaju opis razvojnih modela i pristupa testiranja za pojedini model („Waterfall“, „V“ ili „Agile“).

Peto poglavlje obrađuje područja na koje se svaki inženjer za kvalitetu mora usmjeriti kako bi uspješno osmislio plan testiranja i strategiju održavanja kvalitete softvera.

U glavnom dijelu ovog rada iznijeti će se prikaz računalnih alata, koji nam mogu pomoći u oba procesa u osiguranju kvalitete i njezinoj kontroli. Računalni alati biti će obrađeni u tri kategorije: Alati za upravljanje i tehničku analizu, Alati za inspekciju (analizu programskog koda) i Alati za testiranje i automatizaciju.

Svaki alat proći će kratku analizu, istaknuti će se glavne prednosti i značajke, uvidjeti koje rezultate ti alati mogu generirati (poput grafikona, dijagrama uzroka i posljedica …) te kako ti rezultati mogu biti od pomoći pri procesu osiguranja i kontrole kvalitete.

Na samom kraju predstavit će se najbolje prakse korištenja alata za kontrolu i osiguranje kvalitete programskog koda, prikazati neke od uvida, te probat će se prenijeti neka ključna znanja i prakse u kontekstu osiguranja i kontrole kvalitete. Diskutirati će se o važnom predznanju potrebnom za testiranje softverskih rješenja, te će se usmjeriti na materijale za nastavnike koji žele poučavati nove generacije učenika i studenata koji žele postati budući inženjeri za kontrolu kvalitete.

Na završetku diplomskog rada bit će iznesen zaključak o važnosti kontrole kvalitete i njezinom razvojnom procesu programskih aplikacija. Važnost ove teme i njezino predstavljanje učenicima je od bitnog značaja. Zašto? Zato da bi tijekom kasnijeg školovanja mogli primjereno razmišljati o usmjerenim potrebama za razvoj njihove karijere Inženjera za kvalitetu (QA inženjera).

2. Osiguranje kvalitete („QA“) i Kontrola kvalitete („QC“)

Ovaj diplomski rad započinje citatom iz knjige *Out of the crisis* (Izlazak iz krize) W. Edward Deming-a: „You cannot inspect quality into a product“ (Ne možete inspekcijom ugraditi kvalitetu u proizvod). Izvorni citat Harold F. Dodge-a objašnjava zašto inspekcijom ne možemo niti popraviti niti garantirati kvalitetu. Inspekcija je krenula prekasno. Kvaliteta može biti dobra ili loša i već je ugrađena u proizvod (Deming, 2018).

U današnje vrijeme je teško naći razvojni tim koji nema postavljene alate i procese za provjeru kvalitete programskog koda ili testnih procesa za provjeru ispravnog rada aplikacije ili softverskog rješenja (proizvoda).

Da bi razvili dobro softversko rješenje potrebno je sagledati sve aspekte koji bi mogli utjecati na pouzdanost, kvalitetu i lakoću održavanja, počevši od izrade novih testnih strategija, alata i metodologija koji se mogu primijeniti na životni ciklus razvoja softverskog rješenja. Bez obzira na to koliko dobro razvijamo metode testiranja završne verzije proizvoda, koliko opširne i detaljne dokumentacije imamo, neovisno o strukturiranosti razvojne metodologije, planiranju razvoja, redovnim inspekcijama koda, posvećivanju pažnje kvaliteti baze podataka i kontroli svih procesa konfiguracije, bez obzira na to koristimo li najnaprednije alate i tehnike za razvoj, završni proizvod može biti osuđen na propast, ako sustav upravljanja kvalitetom nije učinkovit (Iqbal & Qureshi, 2009).

Kvaliteta softvera je uvjet koji moramo postaviti i planirati prije nego li je prva naredba novog programskog koda uopće napisana. Da bismo to postigli, moramo razumjeti što je Osiguranje i Kontrola kvalitete softvera (QA i QC).

2.1. Što je Osiguranje kvalitete softvera (SQA)?

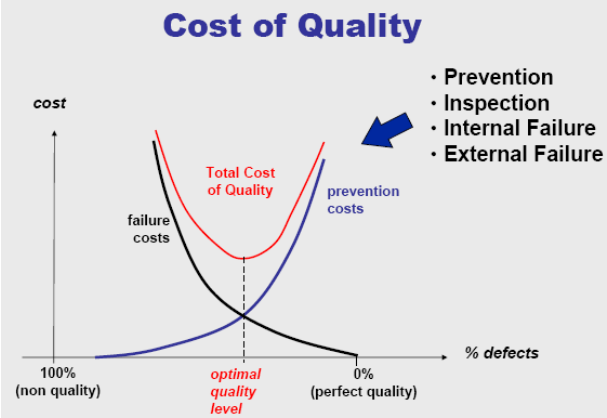
Osiguranje kvalitete softvera SQA (eng. Software Quality Assurance) je cjelokupna aktivnost evaluacije softvera, kojom se osigurava da aplikacija zadovoljava ili premašuje unaprijed određene standarde kvalitete. Ova aktivnost provodi se u svim fazama za razvoj softvera uz istovremeno korištenje inspekcija, metodologija testiranja, modela i tehnika. Osobito su važne sljedeće značajke softvera, poput broja pogrešaka, njegove učinkovitosti, složenosti koje možemo mjeriti primjenom povezanih metrika kvalitete. Cilj osiguranja kvalitete je smanjivanje potrebe za naknadnim ispravljanjem pogrešaka (Papakitsos, 2022).

U istom članku možemo pročitati da se proces osiguranja kvalitete softvera sastoji od procesa tehnika i alata koji osiguravaju da softverski proizvodi slijede unaprijed dogovorene

standarde tijekom faze razvoja ili životnog ciklusa (eng. life cycle). U životni ciklus softvera možemo ubrojiti:

- Fazu izvedivosti
- Analizu
- Dizajn
- Izvedbu/implementaciju
- Isporuku/instalaciju
- Redovni rad/održavanje ili povlačenje

Bez ovih standarda, osiguranje kvalitete softvera ne jamči da je određeni softverski proizvod posve u skladu s kvalitetom ili da premašuje minimum industrijske ili komercijalno prihvatljive razine kvalitete.



Slika 1. Trošak kvalitete

Izvor: Teli i sur. (2010).

Pri primjerenom određenju kvalitete trebamo ponajviše težiti optimalnoj kvaliteti (Slika 1).

Pojam SQA (eng. Software Quality Assurance) odnosi se na sustavne aktivnosti koje pružaju dokaz o prikladnosti konačnog softverskog proizvoda za upotrebu. To se postiže primjenom prihvaćenih pravila kontrole kvalitete kako bi se provjerila cjelovitost i produžio vijek trajanja softvera (Vukašinović, 2023).

Upravljanje životnim ciklusom (eng. Application Lifecycle Management, ALM) ukazuje na koordinaciju aktivnosti i upravljanja artefaktima tijekom životnog ciklusa izrade softverskog proizvoda (Otibine i sur., 2017).

Da bi znali kako optimizirati troškove inspekcija i ispravaka, moramo ponajprije znati kako to ostvariti. Jedan od načina je kontrola kvalitete (QC).

2.2. Što je Kontrola kvalitete softvera (SQC)?

Svi aspekti kontrole kvalitete definirani su certifikatom **ISO 9216** (pouzdanost, funkcionalnost, učinkovitost, prenosivost, upotrebljivost i iznad svega, mogućnost održavanja).

Kontinuirana kontrola kvalitete koristi se za prepoznavanje i rješavanje nedostataka u kvaliteti u što ranijoj fazi razvoja proizvoda, dok je provedba tih protumjera još uvijek jeftina. Procjena trenutnog stanja kvalitete sustava jedna je od bitnijih aktivnosti u kontroli kvalitete. Postoji velik broj različitih aspekata kvalitete koje je potrebno kontrolirati da bi osigurali ispravan rad proizvoda. Moramo pripaziti da troškovi povezani s tim procjenama ne nadmašuju stvarnu korist njihovog izvođenja.

Kontinuirana kontrola kvalitete u praksi može funkcionirati samo ako je osigurana odgovarajuća podrška za alate i sve druge aspekte kvalitete koji se mogu automatski ili jednostavno procijeniti. Spomenuti alati moraju biti učinkoviti i korišteni kao ključni alati koji pokrivaju sve kategorije koje kontroliramo.

Kontrola kvalitete sastoji se od tri ključna elementa:

- jasno definiranih ciljeva kvalitete,
- tehnika, alata i procesa za analizu trenutnog stanja kvalitete,
- odgovarajuće mjere koje će reagirati na otkrivene nedostatke kvalitete.

Za kvalitetu softvera odgovoran je softverski tim, tj. skupina kompetentnih ljudi predanih ostvarenju zajedničkih ciljeva, te jedna ili više stručnih osoba odgovornih za ostvarenje kvalitete softvera ili sustava. Odgovornu osobu za kontrolu kvalitete najčešće imenujemo stručnim nazivom Inženjer za kvalitetu (QA Engineer).

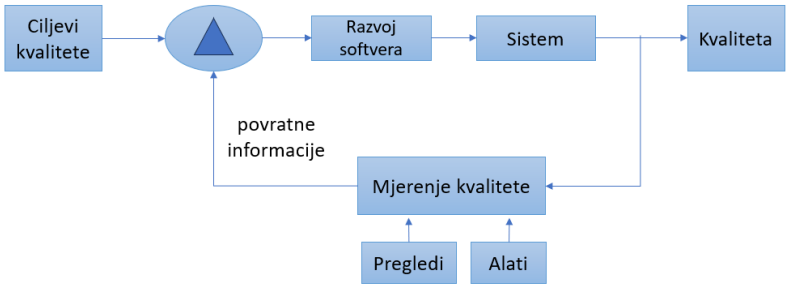
Osnovni zadatak inženjera kvalitete je definirati zadane ciljeve za kvalitetu proizvoda, primjerice u obliku kvalitetnih modela, korištenih standarda tzv. KPI-ova (Ključnih pokazatelja uspješnosti) ili softverske metrike i analitike.

Prilikom pregleda proizvoda inženjer kvalitete može koristiti manualnu tehniku, kao što su ručni ili automatizirani pregledi programskog koda, testiranja jediničnih komponenti (najmanjih oblika inačica programskog koda koje možemo testirati), kao i razne specijalizirane

alate za analizu kvalitete. Inženjer za kvalitetu također može koristiti i rezultate „unit“ testova (jedinično testiranje), statičkih analizatora programskog koda ili drugih tipova testova radi procjene stanja kvalitete softverskog proizvoda.

Nakon odrađenih testova neophodno je usporediti rezultate mjerenja s definiranim ili zadanim ciljevima kvalitete. Na temelju dobivenih rezultata, inženjer za kvalitetu mora surađivati s odgovornim programerima te ih usmjeriti ka poboljšanju kvalitete softverskog proizvoda.

Te promjene često rezultiraju novom revizijom softverskog sustava ili proizvoda (programske verzije), koji naknadno mora ponovno proći ocjenu kvalitete. U svrhu poboljšanja ovog procesa, programeri se dodatno educiraju kako bi povećanje kvalitete moglo biti dugoročno održivo. Ukoliko postoji potreba za tim, inženjer za kvalitetu može reagirati na promjenjive zahtjeve za kvalitetom i poboljšanjem ciljeva kvalitete (Slika 2.).



Slika 2. Petlja za kontrolu kvalitete

Izvedeno iz izvora: Deissenboeck i sur. (2008).

2.3. Koja je glavna razlika između kontrole kvalitete i osiguranja kvalitete?

Osiguranje kvalitete (eng. Quality Assurance, QA) i kontrola kvalitete (eng. Quality Control, QC) dva su najbitnija procesa u razvoju računalnih programa (softvera). Oba pristupa imaju istu namjenu i vrlo su bitni za kvalitetan razvoj softvera, premda imaju posve drugačiji pristup, okruženje i alate. Također je potrebno znati kuda usmjeriti fokus prilikom korištenja tehnika usmjerenih na proces kvalitete (QA) i tehnika usmjerenih na sam proizvod (QC). Postavljanje ciljeva od iznimne je važnosti radi smanjenja pojave grešaka (eng. bug-ova) te njihovog lakšeg otkrivanja. Istim tehnikama osiguravamo standarde za kontrolu kvalitete prilikom testiranja softverskih proizvoda.

Seljan (2018) navodi da se proces Upravljanja kvalitetom (eng. Quality Management) sastoji od osiguranja kvalitete (eng. Quality Assurance, QA), kontrole kvalitete (eng. Quality control, QC) i provjere kvalitete (eng. Quality Verification, QV). Kontrola kvalitete (QC) predstavlja najosnovniju razinu upravljanja kvalitetom što uključuje provjeru, testiranje ili provjeru proizvoda/usluge radi otkrivanja i ispravljanja pogrešaka. Obično se izvodi na kraju procesa, koji je uglavnom proizvodno orijentiran i teži ka poboljšanju kvalitete.

Osiguranje kvalitete (QA) usmjereno je na aktivnosti planiranja i dokumentiranja (koraci, pravila, aktivnosti, postupci) kako bi se spriječile pogreške i postigla kvaliteta. Obično se izvodi na početku procesa, usmjerenog na kvalitetu.

Provjera kvalitete (QV) uključuje aktivnosti evaluacije u cilju pružanja povratne informacije sustavu u svrhu stalnog poboljšanja. QA i QC se često međusobno zamjenjuju, iako su blisko povezani.

Isto tako obrazlaže pojam cjelokupnog upravljanja kvalitetom TQM (eng. Total Quality Management) kao ideju da se koordiniraju i osiguraju svi procesi u tvrtki, uključujući stalna poboljšanja, usmjerenost na korisnika i angažman zaposlenika. Upravljanje kvalitetom (eng. Quality Management, QM) ima za cilj ispuniti standarde i specifikacije kroz kontrolu kvalitete. Također daje smjernice na koji način planiranjem postići rezultate, koji su kompatibilni sa specifikacijama kroz osiguranje kvalitete, te kako poboljšati usluge putem korektivnih povratnih informacija kroz poboljšanje kvalitete (Seljan, 2018).

Slika 3. Odnos Osiguranja i kontrole kvalitete

Na osiguranje kvalitete i kontrolu kvalitete ne treba gledati kao na dva posve odvojena procesa nego kao na procese koji se odvijaju jedan unutar drugoga (Slika 3.). Može se činiti da su oba procesa slična i zamjenjiva, no nakon podrobnijeg razmatranja postaje jasno da između ta dva pojma postoje određene razlike. Važno je zapamtiti da se Osiguranje kvalitete (QA) bavi

sprječavanjem kvara, dok se Kontrola kvalitete (QC) bavi prepoznavanjem nedostataka u softveru.

Glavne razlike između Osiguranja kvalitete (QA) i Kontrole kvalitete (QC):

	Osiguranje kvalitete (QA)	Kontrola kvalitete (QC)
Definicija	Skup aktivnosti koje pomažu odrediti kvalitetu procesa kojim se proizvodi razvijaju.	Skup aktivnosti koje osiguravaju kvalitetu proizvedenih proizvoda
Aktivnost	Uspostavlja i ocjenjuje proizvodni proces proizvoda.	Provjerava zadovoljava li proizvod unaprijed definirane standarde.
Postupak	Pomaže uspostaviti procese.	Utvrđuje zadani proces.
Procjena	Postavlja planove mjerenja za procjenu procesa.	Provjera specifičnih atributa ili svojstva, tj. obilježja koji se pripisuju određenom proizvodu ili usluzi.
Identificira	Identificira slabosti u procesima i poboljšava ih.	Identificira nedostatke za primarnu svrhu ispravljajući pritom pogreške.
Odgovornost	Odgovornost je cjelokupnog tima.	Za kontrolu kvalitete odgovoran je ispitivač ili tim za testiranje.
Nedostaci	Sprječava uvođenje problema ili nedostataka.	Otkriva, izvješćuje i ispravlja nedostatke.
Vrijeme	Izvršava se prije kontrole kvalitete.	Izvršava se nakon što je osiguranje kvalitete završeno.
Osnova	Manualna provjera dokumenata ili datoteka.	Računalno izvršeno izvođenje programa ili koda.
Razina aktivnosti	Može otkriti i uhvatiti i one pogreške koje QC ne može, zbog čega ga se smatra nižom razinom aktivnosti.	Može uhvatiti pogrešku koju QA ne može uhvatiti, zbog čega se smatra visokom razinom aktivnosti.
Alati	Upravljački alat.	Korektivni alat.
Vrijeme	Vremenski zahtjevna aktivnost.	Dugotrajna aktivnost.
Značenje	Planiranje izvršavanja procesa.	Postupak izvršenja procesa.
Proizvod	Poboljšava proces koji se može primijeniti na više proizvoda.	Poboljšava razvoj određenog proizvoda ili usluge.
Aktivnost	Smatra se proaktivnim (pronalazi slabosti u procesima).	Smatra se reaktivnim (pronalazi i ispravlja nedostatke).
Osoblje	Provodi kontrolu kvalitete kako bi se utvrdilo da sustav doista radi.	Obavlja zadatke osiguranja kvalitete kada je to potrebno.
Primjer	Verifikacija.	Utvrđivanje ispravnosti, istinitosti, pravovaljanosti. Važno je naglasiti da verifikacija utvrđuje podatke.

Tablica preuzeta sa: *Spot the difference* (n.d.). https://hr.spot-the-difference.info/difference-between-qa

Osiguranje kvalitete (QA) ne osigurava kvalitetu, nego stvara i osigurava praćenje procesa, kako bi se osigurala kvaliteta softvera. Kontrola kvalitete (QC) ne osigurava njegovu kvalitetu, nego prije svega služi mjerenju kvalitete softvera. Rezultati mjerenja kvalitete mogu

se koristiti za ispravljanje ili prilagođavanje pri procesu osiguranja kvalitete. Isti procesi se mogu uspješno primijeniti i na nove projekte.

Aktivnosti kontrole kvalitete (QC) uglavnom su usmjerene na sam proizvod, dok su aktivnosti osiguranja kvalitete ponajviše usredotočeni na procese koji iz njih proizlaze, a radi ostvarivanja željenog rezultata.

Glavni cilj inženjera za kvalitetu je zajedno koristiti QA i QC, te njihove tehnike kako bi se osiguralo da isporučeni proizvodi budu visoke kvalitete i udovoljavaju očekivanjima kupca.

2.4. Proces testiranja softvera

Kontrola i osiguranje kvalitete nepojmljivo je bez procesa testiranja ili provjeravanja ispravnosti softvera. Kvalitetu je potrebno testirati kako bi je mogli što bolje kontrolirati. Osiguranje kvalitete ponajviše brine o tome kako bi se sam proces testiranja vodio na ispravan način.

Slika 4. Odnos procesa testiranja unutar procesa osiguranja i kontrole kvalitete

Sam proces testiranja spada pod kontrolu kvalitete (QC) i bavi se izvedbom specifičnih testova (Slika 4.) (Difference between QA and QC, n.d.).

Zamislimo prije svega da trebamo razviti plan praćenja problema, kako bismo prijavili pogreške tijekom testiranja neke web stranice ili aplikacije.

Iz osiguranja kvalitete (QA) treba preuzeti i definirati standarde koji uključuju programsku podršku, alate, praćenje pogrešaka, formulare za izvještaje i za reprodukciju grešaka, te način na koji se greške (bug-ovi) prijavljuju. Da bismo mogli te izvještaje popuniti konkretnim detaljima, koristimo kontrolu kvalitete (QC), kojom će se nakon odrade procesa testiranja (manualni ili automatizirani) te dokumente popuniti detaljima (slikom zaslona, koracima za reprodukciju itd.), da bi programeri kasnije mogli što lakše otkloniti grešku u softveru.

Softversko testiranje je skup aktivnosti uz pomoć kojih se iste mogu unaprijed planirati i sustavno provoditi. Glavni cilj testiranja je pronaći pogrešku pri izvršavanju računalnog programa. Osnovni cilj ispitivanja je provjeriti zadovoljava li projektirani softver specifikacije koje je zadao kupac (Ahamed, 2010).

Testiranjem treba ispuniti sljedeće kriterije:

- Test treba započeti na razini najmanjeg modula i raditi prema integraciji cjeline kako bi se pokrio cijeli računalni sustav.
- Test treba biti prikladan i različit ovisno o različitim trenucima u vremenu.
- Uspješno ostvarenje ovisi o tome postoji li neovisna ispitna skupina koja bi uspješno provodila razvojno testiranje različitih softvera i projekata.
- Budući da su testiranje i otklanjanje pogrešaka različite aktivnosti, otklanjanje pogrešaka bi trebalo biti uključeno u svaku aktivnost testiranja.
- Metodologija testiranja softvera treba uključivati testove niske i visoke razine koji su neophodni za provjeru. Mali segment izvornog koda mora biti ispravno implementiran, kao i svi testovi visoke razine. Ti testovi potvrđuju da glavne funkcije sustava rade prema zahtjevima korisnika, a sve to usmjerava ka što boljem pisanju i praćenju testova.

Testabilnost softvera je proces provjere koliko adekvatno određeni skup testova pokriva proizvod. To omogućava osobama odgovornima za testiranje da lakše osmisle učinkovite testne slučajeve (Ahamed, 2010).

Testabilnost softvera je postupak kojim se postiže što jednostavniji način na koji se računalni program može testirati. Jasno je da testiranje nije mali proces. Isti proces je vrlo složen. Važno je znati što se treba učiniti da se složenost istog procesa, što je više moguće pojednostavi (Ahamed, 2010).

Osiguranje kvalitete (QA) također propisuje plan po kojem će testiranje biti izvršeno.

Plan testiranja koristimo za primjereno obavljanje bilo koje aktivnosti testiranja. Svako planiranje započinje planom testiranja. Plan testiranja je opći dokument koji se koristi za cijeli projekt. Definira opseg, pristup koji treba poduzeti i raspored testiranja, kao i identifikaciju ispitnih elemenata za cijeli proces testiranja, te osoblje koje je odgovorno za različite aktivnosti ispitivanja. Ulazni podaci za plan testiranja su vizuali, plan projekta, dokumenti sa zahtjevima korisnika i projektni dokument za dizajn sustava. Idealan plan testiranja trebao bi sadržavati sljedeće (Ahamed, 2010):

- Specifikaciju ispitne jedinice
- Značajke koje treba ispitati
- Pristup testiranju
- Rezultate ispitivanja
- Raspored testiranja
- Raspodjelu osoblja (osobe zadužene za testiranja)

2.4.1. Što testiranje nije?

- Brzopleto i predvidivo pronalaženje svih scenarija brzim pritiscima miša.
- Fokusiranje samo na osnovne kriterije koje je zadao kupac, te je li proizvod zadovoljio zadane standarde.
- Dokazivanje da se program izveo točno iz prvog pokušaja, te da se time njegov razvoj uspješno završio.

Prilikom testiranja proizvodu treba pristupiti s profesionalnošću i ne svesti ga samo na brzinsku provjeru zahtjeva kupca. Testiranje se ne može vršiti korištenjem isključivo manuale tehnike, nego je bitno uključiti i druge tehnike, kao što su već spomenuto automatizacijsko testiranje te istraživačko testiranje. No da bi bili posve sigurni da smo nešto testirali (provjerili) moramo to i istražiti.

Testirano = Provjereno + Istraženo

Istraživačko testiranje ili (AD-hoc testiranje) je vrsta testiranja koju često provodimo bez nekog unaprijed zadanog plana ili dokumentacije. Testiranje se uglavnom provodi neformalno i nasumično bez unaprijed zadanih rezultata. Ovim testiranjem možemo obuhvatiti cijeli sustav, a ne samo uobičajene testne slučajeve. Ovako možemo testovima pokriti i neuobičajene situacije koje ne bi bile pokrivene uobičajenim dokumentiranim testovima (5 Software testing methods, 2018).

3. Uloga inženjera za kontrolu kvalitete

3.1. Koja je uloga inženjera za kvalitetu?

Inženjer za kvalitetu (QA Engineer) softvera ima važnu ulogu u osiguravanju kvalitete softverskih proizvoda. To uključuje sudjelovanje u svim fazama životnog ciklusa razvoja softvera, korištenje različitih pristupa za osiguravanje kvalitete, rješavanje ključnih problema prilikom svih faza testiranja softvera, sprječavanje nedostataka i održavanje kvalitete komponenti i programskog koda (QA Touch, 2023).

Uloga Inženjera za kvalitetu:

1. **Razvija i implementira testne planove:** Cilj je stvoriti detaljne testne planove, razviti ih i strukturirati. Planovi moraju obuhvaćati sve testne slučajeve na temelju korisničkih zahtjeva. Implementirani testni planovi moraju sadržavati pristup testiranju, testne resurse, raspored testiranja i slično.
2. **Razvijanja testne strategije:** Cilj je razviti strategiju utemeljenu na projektnim zahtjevima i rasporedima te što ranije započeti s testiranjem radi smanjenja cijena naknadnih ispravaka i grešaka (bug-ova).
3. **Izvršava testove:** Cilj je izvesti različite vrste testiranja kao što su funkcionalno i regresijsko testiranje, testiranje performansi izvedbe, upotrebljivosti, istraživačko te sigurnosno testiranje.
4. **Dokumentira rezultate testova i izvještaje:** Osnovni je cilj dokumentirati rezultate svih testova, sve otkrivene pogreške (bug-ove), te pratiti njihovo rješavanje koristeći razne alate za praćenje pogrešaka (Bugzila, Jira) Također se treba brinuti da su sve pogreške ispravnog prioriteta i da se rješavaju po planu.
5. **Surađuje s razvojnim timom:** Cilj je blisko surađivati s razvojnim timom kako bi promjene u softveru bile pravovremeno primijećene i prijavljene. Razvojnom timu treba pružiti povratne informacije s jasnim objašnjenjem: a) funkcionira li proizvod i b) može li se pravilno upotrebljavati.
6. **Sudjeluje u agilnom procesu:** Osnovni cilj je ako se u razvoju softvera koristi agilni pristup, QA inženjer treba sudjelovati na dnevnim sastancima, pripremama i planiranju Sprinta i sastancima Retrospektive. Svrha je razumijevanje korisničkih priča i kriterija za softver koji se razvija, te izvještajima o rasporedu rješavanja pogrešaka.

7. **Poboljšava procese za kvalitetu:** Cilj je konstantno tražiti načine za poboljšanje QA procesa, od implementacija novih testnih alata ili metodologija. Tu je i briga o testnom okruženju, te provjeravanju ispravnosti i ažuriranosti.
8. **Osigurava usklađenosti:** Cilj je da softver uspješno zadovolji sve važeće standarde i propise (vezane za industriju, pristupačnost i zaštitu podataka).
9. **Provodi analize temeljnog uzroka:** Cilj je da se identificiraju svi nedostaci radi lakšeg otkrivanja uzroka kvara, a i posljedičnog sprečavanja sličnih kvarova u budućnosti.
10. **Komunicira s ključnim sudionicima:** Cilj je ključne sudionike (eng. stakeholders) – programere, voditelje i poslovne analitičare upoznati s jasnim i razumljivim izvještajima o stanju kvalitete proizvoda (QA Touch, 2023).

3.1.1. Koja je uloga voditelja za kontrolu kvalitete?

Voditelj kvalitete softvera (QA manager) odgovoran je za osiguranje kvalitete softverskih proizvoda. Odgovorni zadaci su učinkovito upravljanje resursima, predviđanje i upravljanje nedostacima (pogreškama), poboljšanje procesa razvoja i balansiranje produktivnosti i kvalitete. Uloga voditelja također podrazumijeva vođenje specijalističkog tima za kontrolu kvalitete i korištenje alata za metriku i praćenje promjena, te mjerenje kvalitete procesa. Odgovornosti su navedene u nastavku (Goodwin University, 2020).

1. **Razvija i implementira QA strategiju**: Odgovoran je za razvoj, implementaciju i održavanje QA strategije unutar tvrtke. Cilj je definirati matricu za kvalitetu, postaviti zadane standarde za kvalitetu i odlučiti o potrebnim i dozvoljenim alatima i procesima koji će se koristiti na pojedinom projektu kako bi se osigurala kvaliteta projekta.
2. **Upravlja timom za kvalitetu:** Odgovoran je za upravljanje timom za kvalitetu, što uključuje zapošljavanje, obuku i mentorstvo inženjera za kvalitetu. Bitni je cilj davati zadatke i upravljati radnim opterećenjem tima za kvalitetu, ne bi li sve aktivnosti testiranja bile dovršene na vrijeme.
3. **Nadgleda procese testiranja:** Odgovoran je za nadziranje svih aktivnosti testiranja, od planiranja do izvršenja zadanih testova. Osnovni je cilj osigurati da se svi testovi provode prema zadanom planu i da se sve pogreške rješavaju u zadanom vremenu.
4. **Prati i obavještava o učinku tima za kontrolu kvalitete:** Odgovoran je za praćenje tima za kvalitetu softvera. Cilj je pripremiti izvještaje i višim voditeljima predstaviti izvješća o učinku i rezultatima tima za osiguranje kvalitete.
5. **Surađuje s drugim timovima:** Odgovoran je za blisku suradnju s drugim ili sličnim timovima, uključujući razvoj, upravljanje proizvodima i korisničku podršku. Cilj je

osigurati da se kontrola kvalitete softvera uzima u obzir u svim fazama životnog ciklusa proizvoda.

6. **Upravlja rizikom i drugim problemima:** Odgovoran je za prepoznavanje potencijalnih rizika i problema s kvalitetom softvera, razvija planove za njihovo ublažavanje i rješavanje još u fazi razvoja. Cilj je upravljanje svim mogućim problemima koji se znaju pojaviti tijekom procesa testiranja i osigurati njihovo što brže razrješavanje.
7. **Poboljšava procese kvalitete:** Odgovoran je za kontinuirano traženje načina poboljšanja procesa kvalitete softvera. Cilj je uključivanje i implementiranje novih metodologija testiranja, alata za testiranje ili praksi testiranja.
8. **Osigurava usklađenosti:** Odgovoran je da su proizvodi tvrtke osigurani u skladu sa svim relevantnim propisima i standardima. Cilj je uključiti specifične standarde za industriju, pristupačnosti ili propise za zaštitu podataka.
9. **Provodi revizije:** Odgovoran za provođenje interne revizije, kako bi se provjerilo i osiguralo da se svi procesi za kvalitetu softvera poštuju unutar tvrtke. Osnovni je cilj postaviti sve uvjete, da bi se mogle provesti vanjske revizije, koje mogu provoditi klijenti ili druga regulatorna tijela.
10. **Održava odnos s klijentom:** U nekim slučajevima voditelji kvalitete softvera mogu kontaktirati s klijentima. Cilj je što bolje razumjeti očekivanja klijenata u vezi s kvalitetom, te riješili sve nedoumice vezane za kvalitetu softvera ili probleme s proizvodom.

Treba napomenuti da sve ove uloge mogu varirati ovisno o veličini tvrtke ili specifičnog softverskog proizvoda koji se razvija.

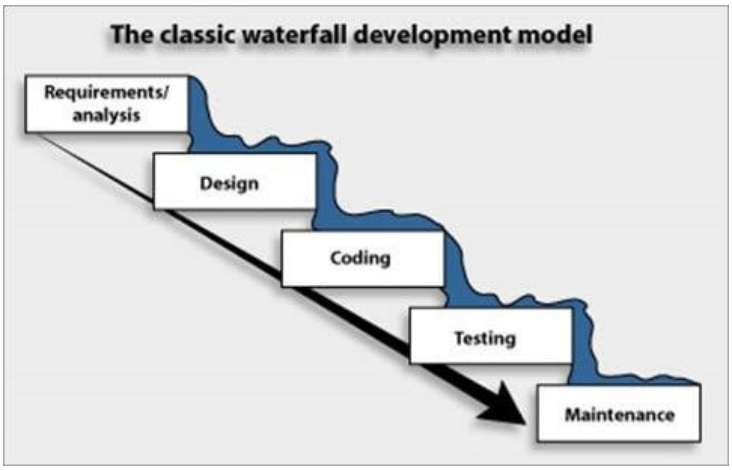
4. Razvojni ciklus i testiranje računalnih aplikacija

Razvojni ciklus i modeli testiranja aplikacija izrazito su važni. Bitno je kada i na koji način započeti s testiranjem te na koji način će se testiranje i razvoj softvera izvesti. Prvi modeli testiranja započeli su vrlo jednostavno i to na samom kraju razvojnog procesa, no tada je već prekasno za testiranje. Testiranje bi trebalo započeti što je ranije moguće, zato jer je popravljanje pogrešaka u završnoj fazi implementacije puno skuplje nego u ranijim fazama (npr. fazi dizajniranja).

4.1. Model vodopada (Waterfall)

Ovaj model vođenja projekta osmislio je znanstvenik Winston W. Royce 1970. godine. Radi se o rasporedu razvojnih faza prema slijedu njihova izvođenja (jedna iza druge). Ovo je jedan od tradicionalnih modela vođenja projekata. Iz istog modela proizašle su i sve druge metodologije vođenja projekata. Prilikom vođenja ovakvih vrsta projekata bitno je sve vremenski unaprijed planirati i definirati. Ovaj model je podijeljen u nekoliko posebnih faza. Jedna se faza poput vodopada prelijeva u drugu (Slika 5.) (Matković & Tumbas, 2010).

1. Analiza i definiranje zahtjeva
2. Oblikovanje specifikacija (dizajn)
3. Implementacija (kodiranje)
4. Testiranje
5. Integracija i održavanje



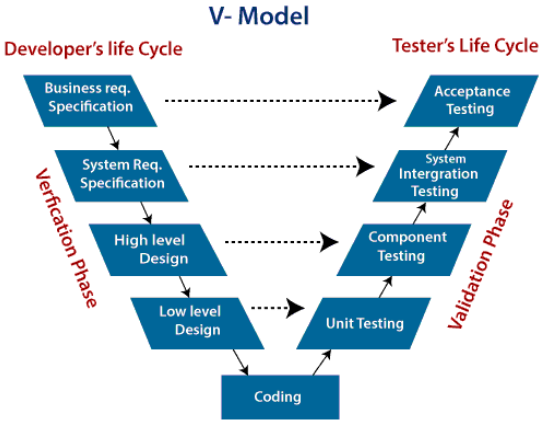
Slika 5. Model vodopada

Izvor: https://productivehut.com/v-model-agile-waterfall-spiral/

4.2. V- model

V-model (eng. Validation and Verification model) je modificirana verzija modela vodopada. Osnovna razlika između V-modela i modela vodopada je da V-model nudi više istovremenih aktivnosti i interakcija (Slika 6.). Razvojni proces je uravnotežen, no i dalje se oslanja na rezultate iz prethodnih faza, te se svaka faza provjerava popratnim testovima (Balaji & Murugaiyan, 2012).

1. Analiza i definiranje zahtjeva
2. Oblikovanje specifikacija (dizajn)
3. Arhitektonski dizajn
4. Dizajn modula
5. Implementacija (kodiranje)
6. Unit testovi
7. Integracijski testovi
8. Sistemski testovi
9. Testovi prihvaćanja



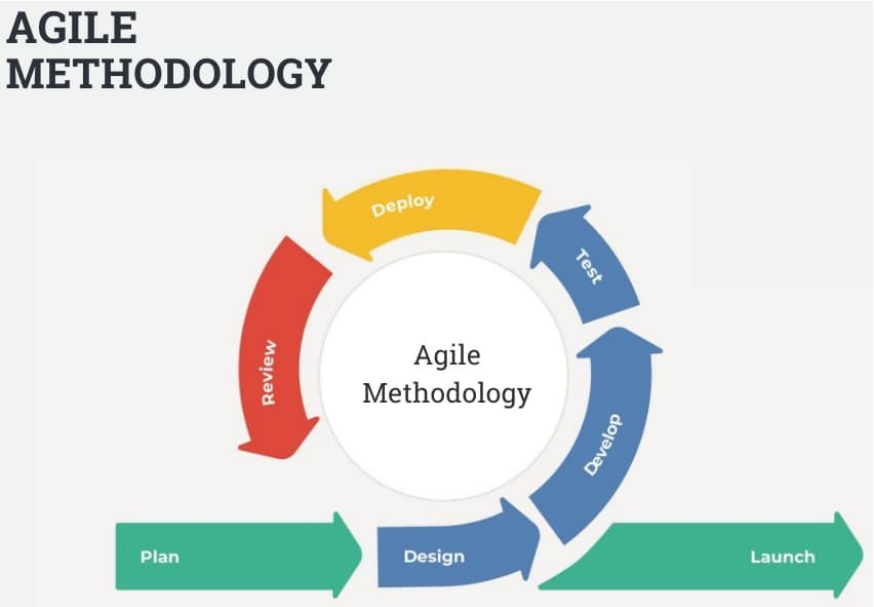
Slika 6. V-model

Izvor: https://www.javatpoint.com/software-engineering-v-model

4.3. Agilni model

Agilni model razvoja razlikuje se od klasičnih modela. Ovaj model osmišljen je 2001. godine kada su projektni stručnjaci zaduženi za razvijanje softvera pokušali pronaći neku novu vrstu rješenja koja će biti fleksibilnija od tradicionalnih modela. Rezultat je bio Agilni manifest koji se koristio kao inicijalni vodič za agilno vođenje projekata. Cilj ovog modela je kontinuirana isporuka, kratki ciklusi razvoja, viša razina komunikacije, te bolja preglednost.

Ovaj model je trenutno najkorištenija metodologija za razvoj softvera. Agilne metodologije bazirane su na iterativnom razvoju, gdje je vrlo bitna suradnja među članovima razvojnog tima. Agilna metoda zamišljena je kako bi se omogućila isporuka visokokvalitetnog softvera što brže, te lakšom prilagodbom na brze promjene tijekom razvoja. Agilni model je poznat po tome da se promjene mogu odraditi i u kasnim fazama projektnog razvoja pomoću manjih iteracija (Slika 7.) (Mekni i sur., 2018).



Slika 7. Agilni model razvoja

Izvor: https://ded9.com/what-is-agile-methodology/

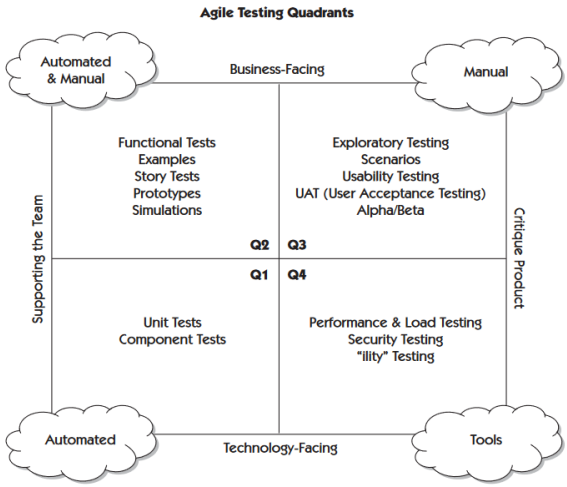
5. Područja na koja se svaki inženjer za kvalitetu mora usmjeriti

Inženjer za kvalitetu prilikom izrade testnog plana i QA pristupa ili strategije mora znati područja testiranja na koja se mora usmjeriti. To najčešće ovisi o veličini projekta ili tima, te softvera na kojem će se pratiti i mjeriti kvaliteta. Najlakše je uzeti u obzir već gotove predloške. U skladu s njima treba skicirati pristup ili strategiju. Jedan od tih predložaka su Agilni kvadranti za testiranje, po kojima vrlo lako možemo odabrati testove koje želimo provesti i kada ih je najbolje provesti.

5.1. Kvadranti za agilno testiranje

Kvadrante za agilno testiranje možemo koristiti kao vodič za naš razvojni tim. U početku je ove kvadrante kreirao Brian Marick, predstavljajući numeriranu matricu podijeljenu na četiri ćelije. Svaki broj ćelije podrazumijeva potreban redoslijed koje treba slijediti.

Prema Crispinu i Gregoryju (2009), kvadranti za agilno testiranje su jednostavna taksonomija, koja omogućuje planiranje testova za agilni razvojni tim. To pruža minimalni broj zahtjeva za samoorganizaciju, kako bi što prije mogli započeti razvoj softvera unutar Sprinta (Slika 5.) (de Castro Martins i sur., 2018).



Slika 8. Kvadranti za agilno testiranje

Izvor: Crispin & Gregory (2009).

S lijeve strane imamo testove koji pomažu timu, dok su s desne testovi koji daju jasnu kritiku proizvodu. Testovi pri vrhu su više poslovne naravi, dok su ovi niže tehnološki orijentirani (Crispin & Gregory, 2009).

Kvadrant 1 (Q1): Kvadrant koji se nalazi niže lijevo predstavlja razvoj baziran na jediničnom testiranju (unit testovima) i pojedinačnom testiranju komponenti. Razvoj vođen testovima (eng. Test driven development, TDD) je glavnina agilnog pristupa razvoju softvera. Ovi testovi su često automatizirani i razvijeni pomoću programskih alata xUnit ili Jest. Cilj im je izmjeriti unutarnju kvalitetu softvera (kvalitetu programskog koda). TDD programerima daje više slobode prilikom razvijanja korisničkih zahtjeva (priča), bez bojazni za nenamjernim većim promjenama u sustavu. TDD testove često definiraju programeri i pokreću se prilikom dodavanja novog programskog koda, te odmah daju povratnu informaciju o njegovoj kvaliteti (interna, tj. unutarnja kvaliteta). Ovaj skup testova je teško razumjeti s poslovne strane, jer je za njihovo razumijevanje potrebna viša razina tehnološkog znanja.

Kvadrant 2 (Q2): Kvadrant koji se nalazi lijevo gore i podržava razvojni tim, ali na višoj razini. Ovaj tip testova je više poslovne namjene, te više pogoduje krajnjem korisniku (vanjska kvaliteta). Kao i u prvom kvadrantu, ovaj skup testova pomaže razvoju softvera, ali na višoj razini. Unutar agilnog razvoja isti su testovi nastali iz primjera i iskustva timova korisnika. Oni detaljno opisuju korisničke priče (zahtjeve za softver). Ovi poslovno orijentirani testovi rade na funkcijskoj razini, te svaki zadovoljava određeni uvjet. Razlikuju se po tome što ih je lakše razumjeti s poslovne strane te ne zahtijevaju visoku razinu tehnološkog znanja. Poslovni stručnjaci ih često koriste kako bi što bolje definirali vanjsku kvalitetu, a ti zahtjevi mogu pomoći pri definiranju takvih tipova testova. Postoji mogućnost da neki od testova u prvom kvadrantu budu identični onima u drugom, no testovi u drugom kvadrantu dati će indikaciju za ispravan rad softvera i sistema na višoj razini.

Postoje mnogi benefiti ako se većina ovih testova automatizira. Na taj način kvadranti Q1 i Q2 mogu dati važne i brze informacije, te tako pridonijeti brzom rješavanju pogrešaka. Potrebno ih je što češće provoditi, jer tako lakše otkrivamo probleme u softveru u mnogo kraćem vremenskom roku nego ručnim testiranjem.

Kvadrant 3 (Q3): Kvadrant koji se nalazi desno gore služi za kritiziranje proizvoda. Takvu vrstu kritike nikada ne smijemo shvatiti u negativnom smislu, već isključivo kao informaciju koja će nam pomoći usavršiti proizvod. Ovi testovi su poslovne naravi i pomažu timu dizajnirati željeni proizvod. Ponekad razna poslovna rješenja mogu biti prepreka samoj funkcionalnosti proizvoda, te zahtjevi korisnika postaju prepreka normalnom funkcioniranju softvera isključivo zbog nepoznavanja tehničkih rješenja. Postoje i primjeri gdje su programeri krivo protumačili zahtjeve korisnika, te premda testovi zadovoljavaju sve kriterije, proizvod ne zadovoljava potrebe korisnika. Ovim testovima također možemo provjeriti može li se naš

softver usporediti s konkurentnim softverom slične namjene. Ideja te provjere sastoji u simulaciji testova krajnjih korisnika u realnim i uobičajenim uvjetima korištenja. Zato se ti testovi ne automatiziraju i preferira se ručno izvođenje. U ovaj tip testova spadaju i istraživački testovi, koji se vrše radi provjere poslovne vrijednosti. Istraživački testovi će također pokriti neočekivane scenarije, koje neće biti pokriveni ni Q1 ni Q2 testovima.

U ovoj grupi se nalaze testovi korisnika i naručitelja softvera, koje često nazivamo testovima korisničkog prihvaćanja User Acceptance testing (UAT). Isti testovi se koriste u svrhu završne provjere, kako bi bili posve sigurni da su sve naručene funkcije i zahtjevi softvera pravilno isporučeni naručitelju softvera. Ovi tipovi testiranja često se rade izvan samog razvojnog tima i daju najbolje rezultate kada se nezavisna tijela bave njegovim ispitivanjem (razne fokus grupe korisnika, ponašanje korisnika itd.)

Kvadrant 4 (Q4): Kvadrant koji se nalazi desno dolje, spada u vrlo bitno područje agilnog razvoja, te razvoja softvera općenito. Ovaj skup testova je tehnički orijentiran i teško ga je razumjeti bez prethodnog tehničkog znanja. Testovi u ovom kvadrantu također imaju namjenu dati jasnu kritiku proizvodu, poput brzine izvođenja, robusnosti i sigurnosti korištenja. Razvojni timovi već posjeduju neka od znanja za izvođenje tih testova, poput korištenja jediničnih testova te ponavljanjem više testova paralelno. Uobičajeno je za takvu vrstu testiranja koristiti specijalizirane alate, za čije su korištenje ponekad potrebne i dodatne kompetencije. Ukoliko se dovoljno ne posvetimo ovom području, postoji mogućnost pojave problema. Kada se razvojni tim više fokusira na zahtjeve korisnika i poslovne priče, testovi ovog kvadranta često padnu u drugi plan.

Korisnicima je postalo vrlo bitno vrijeme čekanja i odaziva. Kako bi se što brže dobio željeni rezultat, stručnjaci su razvili posebne alate koji mogu pomoći prilikom testiranja. Današnje aplikacije sve više koriste internet kao medij, što otvara mogućnosti raznih ranjivosti u sigurnosti i mogućih krađi osobnih podataka korisnika. Da bismo se zaštitili ne smijemo zanemariti ovaj tip testova.

6. Pregled računalnih alata za kontrolu i osiguranje kvalitete koji se koriste za testiranje i održavanje računalnih aplikacija

Računalni alati za kontrolu i osiguranje kvalitete vrlo su bitni u procesu razvoja softverskih proizvoda. Služe nam kako bismo ostvarili i uvjerili se da završna verzija proizvoda zadovoljava sve standarde za kvalitetu, te da su svi korisnički zahtjevi ispunjeni. Osnovni cilj alata je da pomognu implementirati zadane metodologije u svrhu poboljšavanja razvojnog procesa, standarada i procedura za kontrolu kvalitete na samom početku razvojnog procesa. Pomažu nam identificirati, istražiti i pomoći u ispravljanju pogrešaka i problema u softveru. Također pomažu pri pravilnom funkcioniranju, pouzdanosti, brzini performansi i sigurnosti (imunosti na ranjivost) softverskih proizvoda.

Kako vrijeme prolazi pojedini softverski sustavi trpe postupno opadanje kvalitete što znači da troškovi za održavanje takvog sustava mogu neprestano rasti. Zbog toga je neophodno poduzeti nužne mjere da bismo pravovremeno izbjegli dodatne troškove. Isti nam alati također pomažu pri donošenju odluka te ranom otkrivanju problema s kvarom, dok je njihovo uklanjanje još jeftino. Kao stalne nuspojave, pravovremene povratne informacije pomažu programerima i stručnom osoblju za održavanje i razvijanja svojih vještina, a samim time i te smanjivanja vjerojatnost budućih nedostataka pri održavanju kvalitete softvera. Da bi kontrola kvalitete softvera bila izvediva, mora biti visoko automatizirana, a rezultati procjene moraju biti prikazani na cjeloviti način kako bi se izbjeglo nepotrebno preopterećenje korisnika podacima (Deissenboeck i sur., 2008).

6.1. Alati za upravljanje i tehničku analizu

6.1.1. Alati za upravljanje:

1. **Jira (QA i QC)** je često korišten alat za upravljanje projektima koji se najčešće koristi za praćenje statusa naloga, pogrešaka (bug-ova) te agilno upravljanje projektima. Pomaže timovima pri planiranju, praćenju i upravljanju projektima razvoja softvera. Alat se proširuje pomoću raznih dodataka, ovisno o potrebama za koje se koristi. (https://www.atlassian.com/software/jira)
2. **Trello (QA):** Popularan alat za suradnju koji organizira projekte na ploče. Koristan je za upravljanje i praćenje tekućih zadataka i stanje njihovog rješavanja. (https://trello.com/)

3. **Microsoft Project (QA):** Poznati softverski alat koji je tvrtka Microsoft razvila za upravljanje projektima. Alat je dizajniran ponajviše kao pripomoć voditeljima projekata, ali i za razvoj planova, rasporede za rješavanje zadataka, praćenje napretka, upravljanje proračunima, kao i za posjedovanje i analizu opterećenja.
   (https://www.microsoft.com/en-us/microsoft-365/project/project-management-software)
4. **Asana (QA):** Web i mobilna aplikacija dizajnirana da pomogne timovima organizirati, pratiti i upravljati zadacima. Važno je da promovira rad između više povezanih funkcija.
   (https://asana.com/)
5. **VersionOne (QA):** Poslovna softverska platforma dizajnirana je za jednostavnije vođenje razvojnih procesa, te objedinjuje Agile i DevOps (tima odgovornog za infrastrukturu). Može se uskladiti sa zahtjevima više timova, projekata i portfelja poduzeća.
   (https://www.agilealliance.org/sponsors/version-one/)
6. **TestRail (QC):** Testna platforma koji koristi internet za upravljanje testnim scenarijima. Koriste ga timovi za upravljanje, praćenje i organiziranje testiranja. Služi ponajviše za centralizaciju i skaliranje procesa testiranja i praćenja kvalitete.
   (https://www.testrail.com/)
7. **qTest (QC):** Skalabilno softversko rješenje za upravljanje testiranjem koje omogućava brzi uvid u proces testiranja u vremenu. Alat koji pruža podatke o analitici te služi za centralizaciju procesa testiranja kroz razvoj.
   (https://www.tricentis.com/products/unified-test-management-qtest)
8. **Zephyr (QC):** Robustan alat za upravljanje testiranjem i komunikaciju s testnim timom. Podržava cijeli životni ciklus testiranja i odličan je alat za praćenje rezultata testova. Može se integrirati s Jirom i sličnim platformama.
   (https://smartbear.com/test-management/zephyr/)
9. **Redmine (QC):** Alat otvorenog koda koji služi za upravljanje projektima koji uključuje i sustav praćenja pogrešaka (bug-ova). Lako se konfigurira i može se koristiti za upravljanje različitih vrsta platformi.
   (https://www.redmine.org/)

6.1.2. Alati za tehničku analizu:

1. **SonarQube/SonarCloud (QA):** Platforma koja služi za kontinuiranu provjeru kvalitete tj. čišćenje programskog koda. Koristi se za automatske preglede i statičke analize programskog koda u svrhu ranog otkrivanja pogrešaka. Radi brze analize koda istodobno oslobađajući više vremena za fokusiranje na inovacije. (https://www.sonarsource.com/products/sonarcloud/)
2. **PMD (QA):** Alat koji služi za statičku analizu programskog koda. Ovo je alat otvorenog tipa koji podržava više programskih jezika te izvještava o mogućim problemima pronađenima unutar softverskog koda. (https://pmd.github.io/)
3. **Checkstyle (QA):** Razvojni alat koji pomaže programerima prilikom pisanja Java programskog koda. Pripomaže im da se pridržavaju standarda kodiranja. Automatizira proces provjere Java koda. (https://checkstyle.sourceforge.io/)
4. **FindBugs (QA):** Besplatni program koji koristi statičku analizu za traženje pogrešaka u Java kodu. Može otkriti razne uobičajene pogreške kodiranja, uključujući probleme s performansama. (https://findbugs.sourceforge.net/)
5. **Coverity Scan (QA):** Besplatni alat čija je osnovna svrha analizirati statički kod. Isti alat razvojnim programerima pomaže otkriti moguće probleme u kvaliteti softvera te njegove sigurnosne slabosti. Može se integrirati s Github softverskim rješenjima. (https://scan.coverity.com/)
6. **JUnit (QC):** Jednostavni softverski alat za pisanje testova u programskom jeziku Java. Testovi se mogu izvoditi višestruko. Koristi se za jedinično testiranje (eng. unit testing) i može se spajati s drugim Java softverskim okvirima i alatima za testiranje. (https://junit.org/junit5/)
7. **Selenium (QC):** Popularan alat za automatizaciju softvera koji koristi internetski pretraživač kojim može upravljati uz pomoć WebDriver-a. Utemeljen je na otvorenom kodu i omogućuje snimanje i reprodukciju potrebnu za provođenje testova bez provođenja testnog programskog jezika. (https://www.selenium.dev/)
8. **Postman (QC):** Alat za testiranje aplikacijskog programskog sučelja (API). Korisnicima omogućuje slanje različitih vrsta HTTP zahtjeva Hypertext Transfer

Protokola i provjeru valjanosti njihovih odgovora. Pojednostavljuje svaki korak u API životnom ciklusu što unaprjeđuje kvalitetu.
(https://www.postman.com/)

9. **Apache JMeter (QC):** Softver otvorenog koda dizajniran za izvođenje funkcionalnog testiranja te mjerenja performansi. Može se koristiti za simulaciju opterećenja na poslužiteljem, grupi poslužitelja, mreži ili objektu kako bi se testirala njegova snaga i stabilnost ili analizirala sveukupna brzina izvođenja pod različitim vrstama opterećenja.
(https://jmeter.apache.org/)
10. **Load Runner professional (QC):** Alat za testiranje brzine izvođenja tvrtke Micro Focus (Opentext). Koristi se za testiranje aplikacija, mjerenje ponašanja sustava i brzine izvođenja pod opterećenjem.
(https://www.microfocus.com/en-us/products/loadrunner-professional/overview)

6.2. Alati za inspekciju (analizu programskog koda)

Alati za inspekciju koriste se za utvrđivanje standarda kvalitete programskog koda. Možemo ih staviti u obje kategorije (QA i QC).

1. **Crucible (Atlassian)**: Ovaj alat je softversko proširenje tvrtke Atlassian i koristi se za kolaborativni pregled programskog koda. Među timovima, osim suradnje, on nudi i detaljan pregled programskog koda, raspravu o promjenama, dijeljenje znanja i prepoznavanje nedostataka. Lako se spaja s drugim softverskim proizvodima poput Jira i Bitbucket.
(https://www.atlassian.com/software/crucible)
2. **Gerrit:** Besplatni alat za timsku suradnju putem interneta. Omogućuje programerima pregled izmjena na izvornom kodu te odobravanje ili odbijanje napravljenih promjena. Koristi se zajedno s Git-softverskim repozitorijem.
(https://www.gerritcodereview.com/)
3. **GitHub:** Odnosi se na uslugu internetskog poslužitelja, utemeljenu na sustavu za kontrolu softverskih verzija uz pomoć različitih softverskih rješenja Git-a ili sustava otvorenog koda za distribuciju i kontrolu mogućih verzija. Također omogućuje nekoliko značajnih suradnji, kao što je praćenje naloga, pogrešaka (bug-ova), zatim zahtjeva za novim funkcijama, upraviteljem zadataka i pregledima koji uključuju odobravanje programskog koda.
(https://github.com/)

4. **GitLab:** Alat sličan GitHub-u koji je također i DevOps (tima odgovornog za infrastrukturu) alat za upravljanje životnim ciklusom softvera. Sadrži upravitelj Git-repozitorija temeljenog na otvorenom kodu te druge repozitorije, poput sustava za praćenje stavki i CI/CD infrastrukture za automatizaciju testiranja.
   (https://about.gitlab.com/)
5. **Review Board**: Internetski alat za pregled koda temeljen na otvorenom kodu, koji programerima nudi jednostavniji pregled programskog koda. Dobro se skalira od manjih projekata prema većim i programerima nudi niz alata za što uspješnije i jednostavnije preglede programskog koda.
   (https://www.reviewboard.org/)

6.3. Alati za automatsko testiranje i automatizaciju

Alate za automatizaciju poput JUnit, Selenium i Postman već smo spomenuli.

1. **Cucumber:** Odnosi se na način pisanja, ali koristi se i kao alat za pokretanje automatiziranih testova prihvaćanja. Napisan u stilu vođenim ponašanjem u razvoju (eng. Behavior-Driven-Development, BDD). Često se koristi za testiranje web aplikacija.
   (https://cucumber.io/)
2. **TestNG:** Okvir za testiranje koji je inspiriran softverskim okvirima JUnitom i NUnitiom, a unosi neke nove funkcije koje ga čine moćnijim i lakšim za korištenje. Osmišljen je ponajviše zato da pokrije sve kategorije testova.
   (https://testng.org/doc/)
3. **Apache JMeter:** Već smo ga spomenuli.
   (https://jmeter.apache.org/)
4. **Jenkins:** Poslužitelj za automatizaciju otvorenog koda koji programerima omogućuje izradu, testiranje i implementaciju softvera. Koristi se za implementaciju prakse kontinuirane integracije i isporuke (CI/CD), te automatizaciju pokretanja automatiziranih testova.
   (https://www.jenkins.io/)
5. **Ranorex**: Set softverskih alata koji služe za automatizaciju na različitim softverskim okruženjima kao što su desktop i web, te mobilnim uređajima. Može nam poslužiti i za pisanje testova u kojima se otklanjaju nepotrebne varijacije, te tako razvija bolje testne scenarije.
   (https://www.ranorex.com/)

6.3.1. Korisni alati za svakog inženjera za kvalitetu

Uređivač teksta i programskog koda:

Notepad++ https://notepad-plus-plus.org/download/

Visual Studio Code https://code.visualstudio.com/Download

JetBrains Aqua https://www.jetbrains.com/aqua/

Internetski pretraživač:

Chrome https://www.google.com/chrome/

Firefox https://www.mozilla.org/en-US/firefox/new/

Dodaci za internetski pretraživač:

Prijepis internetske adrese ili QR koda za testiranje aplikacija putem mobilnog uređaja

https://chrome.google.com/webstore/detail/qr-codegenerator/

PDF čitač:

Adobe Reader https://get.adobe.com/reader/

PDF24 https://tools.pdf24.org/en/creator

Alati za spremanje izgleda ekrana u obliku slike:

Screenpresso https://www.screenpresso.com/download/

Lightshot (Win/Mac) https://app.prntscr.com/en/

Greenshot https://getgreenshot.org/downloads/

Alat za snimanje ekrana u gif formatu:

LICECap https://www.cockos.com/licecap/

Alat za promemoriju:

Ditto https://ditto-cp.sourceforge.io/

Upravljanje lozinkama:

Keypass https://keepass.info/

Alati za testiranje opcija za korisnike s posebnim potrebama:

Čitač zaslona: NVDA (za Windows operacijski sustav) https://www.nvaccess.org/

Glasovni čitač: VoiceOver (samo za macOS i iOS operativni sustav)
https://help.apple.com/voiceover/info/guide/10.12/?lang=en

WAVE Evaluation Tool (dodatak za preglednik Chrome)
https://chrome.google.com/webstore/detail/wave-evaluation-tool/

Alati za automatsko testiranje:

E2E testiranje, UI testiranje, web komponente i CI/CD integracije:

- Cypress https://www.cypress.io/
- Selenide https://selenide.org/
- Playwright https://playwright.dev/
- Testrigor (AI) https://testrigor.com/

Testovi brzine izvođenja i opterećenja:

- Jmeter https://jmeter.apache.org/
- Gatling https://gatling.io/

API testiranje:

- Postman https://www.postman.com/

Vođenje, upravljanje, dokumentacija i izvedba testnih planova unutar Jire:

- Xray https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-Jira
- Zephyr https://marketplace.atlassian.com/apps/1014681/zephyr-squad-test-management-for-Jira

Kvaliteta koda:

- SonarQube https://www.sonarqube.org/

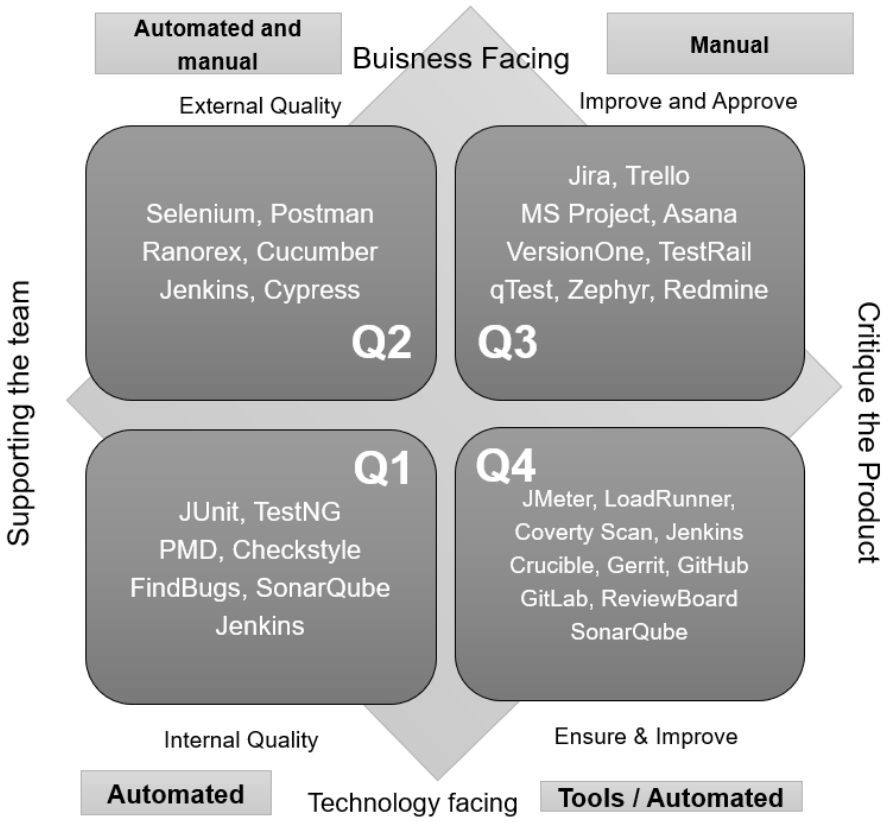
Virtualno testiranje na raznim mobilnim uređajima:

- Browserstack https://www.browserstack.com/

Slanje i primanje e-mail poruka:

- MailCatcher: https://mailcatcher.me/

Možemo sve navedene alate (QA i QC) prikazati poredane prema agilnim kvadrantima (Q1-Q4) (Slika 9.).



Slika 9. QA i QC alati poredani prema agilnim kvadrantima

7. Najbolje prakse korištenja alata za kontrolu i osiguranje kvalitete programskog koda.

Bitno je znati koje su mogućnosti i ograničenja pojedinog alata. U prikazu agilnih kvadranata već smo otkrili da nisu svi alati prikladni za sve vrste testiranja. Prije nego li krenemo koristiti alate, potrebno je osigurati potrebna znanja, kako bi svi članovi tima za testiranje bili adekvatno upoznati s radom pojedinog alata te razumjeli povratne informacije koje ti alati nude. Alate treba integrirati kako bi se uklopili u proces razvoja softvera. To se najčešće postiže postavljanjem testova za automatsko testiranje, kontrolom verzija te povezivanje rezultata s alatom za vođenje projekta. Alate je potrebno redovito ažurirati i pratiti pojavu novih i boljih alata koji mogu pomoći i pojednostaviti proces kontrole kvalitete.

Potrebno je pratiti i dokumentirati sve rezultate rada s alatima kako bismo znali koristimo li ih pravilno, te zapisati sve moguće probleme koji bi se mogli pojaviti prilikom njihovog korištenja u svrhu razmjene znanja među članovima tima. Na taj način možemo povećati efikasnost korištenja alata čija je svrha dobivanje što boljeg uvida u stanje kvalitete proizvoda i zadovoljavanje svih potrebnih standarda za kvalitetu.

7.1. Razlozi za automatizaciju

Već smo ustanovili da je glavni nedostatak manualnog testiranja zapravo vrijeme potrebno za izvođenje testova. Glavni cilj automatizacije testiranja je pravovremeno prepoznavanje pogreške u softveru te njeno uklanjanje. Što je dulje čekanje, cijena otklanjanja pogreške rapidno raste (Slika 10.) (Dawson i sur., 2010).

Trošak ispravljanja grešaka:

100 puta više od ispravljanja greške u fazi dizajna
15 puta više od ispravljanja greške tijekom faze implementacije

Trošak pogrešaka tijekom druge razvojne faze:

Zadatak traje **dvostruko duže**.

Dvostruko više grešaka nego u fazi dizajna

Slika 10. Relativna cijena troška za ispravljanje pogreške

Primijetimo li pogrešku dok je još u fazi dizajna i prije nego li je kod u fazi implementacije, cijena za otklanjanje pogreške gotovo je zanemariva. Zato je bitno da se QA inženjeri što ranije uključe u proces razvoja (Slika 10.) (Czerwinski et al., 2004).

7.2. Distribucija automatiziranih testova ili testna piramida

Kada pričamo o automatiziranim testovima u testnim kvadrantima, najbolji primjeri za automatizaciju su Jedinični testovi (eng. unit tests) i regresijski testovi. Međutim, uvijek se postavlja pitanje u kojem omjeru ti testovi moraju biti pisani. Jedno od strategija za automatsko testiranje je „Testna piramida“ koju je osmislio Mike Cohn 2009. godine. Ona najbolje opisuje kako distribuirati testove unutar agilne organizacije. Unit testovi su na dnu piramide i njih će uvijek biti najviše. Integracijski testovi (API i testovi komponenti) su u sredini. Testovi sučelja UI (eng. user interface) testovi su na vrhu piramide, koji će provjeriti sve glavne poslovne scenarije. Iznad piramide nalazi se oblačić s natpisom: Manualni testovi, na koje se ne smije zaboraviti, jer krajnji rezultat treba i manualno provjeriti (Slika 11.) (Radziwill & Freeman, 2020).

Da bi olakšali i ubrzali te automatizirali pregled i kontrolu kvalitete programskog koda, dobro je koristiti alat poput *SonarQube-a*, koji će nam dati detaljni uvid u stanje našeg programskog koda.

Slika 11. Distribucija automatiziranih testova (Testna piramida)

Možda u praksi naš softverski proizvod nikada nećemo moći pokriti sa 70% unit testova, ali ako dogovorimo koji je minimum potreban da bi se zadovoljila minimalna kvaliteta (eng. quality gate), onda sve ono što je ispod tog minimuma možemo smatrati nezadovoljenom kvalitetom programskog koda.

7.3. Kako napraviti analizu alata koje bi mogli koristiti na projektu

Da bi novi alat predstavili projektnom timu ili organizaciji, potrebno je prije donošenja odluke dobro razmisliti koji je mogući učinak tog alata. Ta odluka može zahvatiti razna područja, kao što su produktivnost i učinkovitost tima.

Nužno je dobro razumjeti strategiju testova na projektu te potrebe organizacije koja te testove treba izvoditi.

1. **Definirati kriterije za evaluaciju** – može li se alat lako integrirati u sustav, posjeduje li skalabilnost, lakoću korištenja, prihvatljivu cijenu te potrebnu potporu i lakoću održavanja?
2. **Identificirati moguće alate** – odabrati nekoliko sličnih alata koji mogu zadovoljiti kriterije postavljene u drugom koraku.
3. **Detaljna analiza** – na odabranim alatima provesti detaljnu analizu. Pronaći dobre i loše strane, potencijal, te moguće probleme koje bi taj alat mogao prouzročiti. Potrebno je pribaviti alat za probnu demonstraciju te ga koristiti tijekom probnog roka.
4. **Rad u timu** – kada se uvjerimo u učinkovitost alata, slijedi predstavljanje timu te dokumentiranje svih povratnih informacija. Ponekad je potrebno organizirati i radionice, te članove tima upoznati s korištenjem određenog alata.

5. **Podrška za alat** – treba provjeriti je li alat redovito održavan i podržan od strane koja je za to zadužena. Također treba provjeriti ima li alat svu potrebnu dokumentaciju.
6. **Ukupna cijena alata** – treba provjeriti je li za alat potrebno platiti licencu ili je posve besplatan. Bio alat besplatan ili ne, treba proučiti uvjete korištenja: koliki je vremenski period potreban za održavanje te postoje li periodi kada je alat radi održavanja nemoguće koristiti te cijenu obuke zaposlenika.
7. **Odabir alata** – kada su prikupljene sve potrebne informacije, možemo donijeti informiranu odluku, prezentirati alat svim potrebnim stranama te prikupiti sva potrebna odobrenja.
8. **Plan za implementaciju** – potrebno je izraditi detaljan plan po kojem ćemo odraditi implementaciju alata u postojeće procese u organizaciji. Koraci za instalaciju i integraciju, plan obuke zaposlenika i period tranzicije ili migracije postojećih rješenja. Također bi trebali moći mjeriti efikasnost nakon što smo alat uspješno integrirali u procese u organizaciji.

7.4. Kako učenicima i studentima približiti temu osiguranja i kontrole kvalitete?

Tema kontrole i osiguranja kvalitete se rijetko spominje te u pravilu ne postoji kao zaseban predmet, već se često pojavljuje samo u kontekstu organizacije razvojnih procesa i kvalitete velikih podataka. Zato je učenicima i studentima bitno ovu temu prikazati putem praktične demonstracije te objasniti zašto je osiguranje i kontrola kvalitete važna za razvojni proces. Potrebno je prikazati primjere postavljanja visokih standarda kvalitete proizvoda koje koristimo, demonstrirati primjere proizvoda loše kvalitete, te potencijalne probleme koji mogu nastati ako proizvod ne zadovoljava minimalne zahtjeve za kvalitetu.

Karijera svakog inženjera traži i dobro poznavanje osnovnih razvojnih procesa softvera. Ta iskustva moguće je prenijeti još za vrijeme naobrazbe, da se budući inženjeri što ranije i lakše mogu prilagoditi zahtjevima razvojnog tima. Najlakši način je da se potrebna znanja prenose putem rada na radionicama i konkretnim radom na projektu. Tako se na jednostavnim primjerima može upozoriti na potencijalne greške koje nastaju prilikom stvaranja programskog koda.

Učenicima i studentima nakon svake radionice treba omogućiti pristup potrebnoj dokumentaciji i znanstvenim člancima za što lakše razumijevanje teme. Bitno je dati što više primjera iz stvarnog života i savjete kako ih riješiti. Moguće je i planirati posjete raznim IT

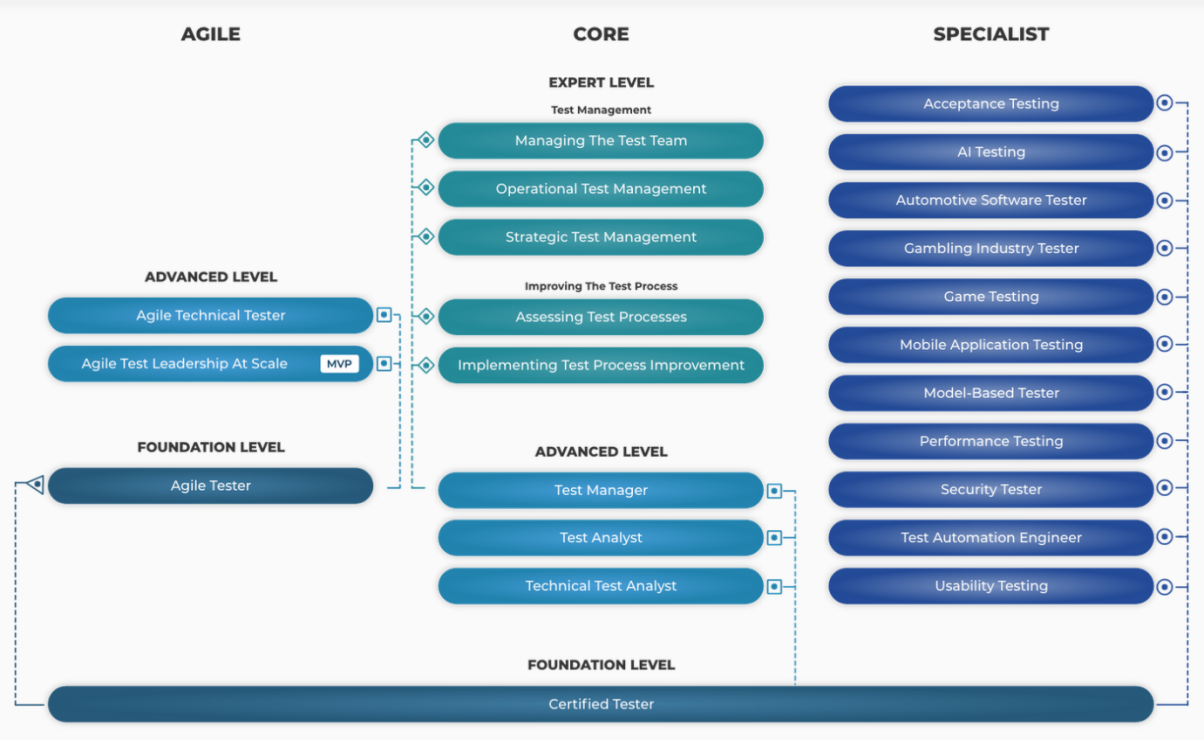
tvrtkama kako bi učenici i studenti mogli na licu mjesta vidjeti kako se na dnevnoj bazi ti procesi organiziraju, organizirati javna predavanja na koja se mogu pozvati predstavnici IT tvrtki i gosti predavači koji mogu s učenicima podijeliti najnovija iskustva i razvojne tehnologije.

Razvojne tehnologije i tehnike su trenutno na samom pragu velikih promjena. Svakim danom predstavljaju se novi alati za kontrolu kvalitete bazirani na umjetnoj inteligenciji koja bi trebala pomoći i olakšati inženjerima kvalitete rad u ovoj industriji. Tehnike pisanja kvalitetnih upita prema umjetnoj inteligenciji te razumijevanje izlaznih informacija svakim su danom sve traženija vještina. Umjetna inteligencija nikada neće moći u potpunosti zamijeniti zanimanje QA-a, no od budućih inženjera tražiti će se kompetencije za puno bolje razumijevanje cjelokupne „velike slike“ (eng. big picture).

Potrebno je poznavati osnove programiranja prilikom pisanja automatiziranih testova, paziti na sigurnosne prijetnje ili propuste koje takve tehnologije mogu stvoriti i vršiti provjere dizajna aplikacije prije nego li je proces kodiranja uopće započeo. AI alate biti će potrebno uvesti u postojeće procese razvoja, te osvijestiti da korištenje takvih alata može biti i vrlo riskantno po rezultate poslovanja. Jedan od primjera je i krađa podataka trećih strana te dodavanje programskog koda koji, ako nije pravilno prilagođen, može naštetiti projektu.

Mladim generacijama je posebno važno prenijeti određene vrijednosti, kao što su etika i zaštita privatnosti. Nikako ne smijemo testirati na podacima kojima bi mogli kompromitirati nečiju privatnosti, pa je stoga bitno izmiješati testne podatke kako ne bismo prekršili određene zakone vezane za zaštitu privatnih podataka. Sve je rezultate potrebno višestruko provjeriti jer se lako može dogoditi da točnost podatka varira s obzirom na različita mjerenja.

Edukacija inženjera za kvalitetu ponekad zahtjeva i dodatnu edukaciju i popratne certifikate koji se mogu položiti nakon školovanja kako bi se usvojili određeni standardi za testiranje poput ISTQB (eng. International Software Testing Qualifications Board) (Slika 12.).



Slika 12. ISTQB certifikati (www.istqb.org)

Poput temeljnog certifikata (eng. foundation level) i naprednog (eng. advanced level) tu su i certifikati za agilno testiranje, te specijalizirani certifikati za automatizaciju i sigurnosno testiranje. Uz svaki certifikat ova svjetska organizacija organizira popratne edukacije putem interneta ili kroz certificirane ustanove koje su opremljene posebnom opremom za polaganje ovakvih vrsta certifikata.

Jedna takva ustanova koja nudi usluge edukacije i polaganja certifikata je tvrtka CROZ d.o.o, te ProTest Solutions d.o.o. i Quality house (Srbija), no za detaljan popis mjesta gdje se može certificirati najbolje je koristiti službenu stranicu jer se popis lokacija često mijenja: https://www.istqb.org/certifications/find-an-exam-provider

8. Zaključak

U Hrvatskoj postoji malo diplomskih radova na temu kontrole i osiguranja kvalitete. Također u školovanju i predviđenom programu još uvijek nema predmeta koji se bavi isključivo kontrolom i osiguranjem kvalitete softvera u svrhu zanimanja Inženjer za kvalitetu. Današnji poslodavci od kandidata koji se prijavljuju na tu poziciju očekuju i povezane kompetencije (poput osnova programiranja, rad s bazama podataka te API servisima), a uz to traže i dodatno poznavanje tehnika za osiguranje i kontrolu kvalitete. Najčešće se te kompetencije uče kroz rad, no svrha škole i fakulteta je da učenicima i studentima prenesu određena znanja koja će im pomoći da nauče i savladaju potrebne tehnike, te da nastave školovanje.

Nakon što formalno obrazovanje završi, kandidati za ovo zanimanje uz rad mogu polagati i popratne certifikate (poput ISTQB-a), te se školovati u posebno ovlaštenim ustanovama kako bi proširili svoja stečena znanja. Stečeni certifikat ne garantira i iskustvo obavljanja ovog zanimanja, no može se koristiti kao platforma za usavršavanje tehnika osiguranja i kontrole kvalitete. Edukacija se može vršiti unutar same organizacije u kojoj radimo (kroz kratka predavanja i treninge), održavanjem javnih tribina ili organiziranjem radionica (poput projekta Alumni).

Glavni cilj je da se razina kvalitete softvera podigne na zadovoljavajuću razinu. To je moguće jedino ako se kroz suradnju sa svim povezanim zanimanjima u IT-u ova znanja šire dalje. Na taj način će svi članovi tima za razvoj softvera bolje razumjeti jedni druge. Cilj je bolje komunicirati, jer nametanje određenih pravila bez jasnih objašnjenja može biti krivo protumačeno, što lako može dovesti do konflikata unutar tima. Treba širiti poruku da su svi članovi tima zaduženi za kvalitetu softverskog proizvoda, a ne samo Inženjer za kvalitetu.

Uloga voditelja za kvalitetu nije samo nadgledati, već i upoznati timove za razvoj s najnovijim tehnikama i alatima koji se koriste za što efikasnije i kvalitetnije upravljanje kvalitetom. Također treba znati kreirati popratnu dokumentaciju, testne planove i testnu strategiju, te odabrati popratne alate i tehnike koje će zadovoljavati potrebe projekta na kojima se radi. Svaki projekt i svaki tim je drugačiji, te se iste tehnike, strategije i korišteni alati mogu dodatno prilagođavati i usavršavati.

Učinak rada alata za kontrolu i osiguranje kvalitete treba znati mjeriti kako bi mogli donijeti određene odluke koje alate odabrati. Sam softver i softverski alati se često mijenjaju te je potrebno konstantno istraživanje. Primjeri koji su navedeni u ovom radu odraz su vremena u kojem je ovaj rad pisan. Moguće je da će u sljedećim godinama AI alati dominirati ovim područjem i trenutni alati više neće biti u upotrebi. Moguće je i da postojeća podrška za ove

alate više neće biti moguća i logika nalaže da će se sustavi migrirati prema naprednim alatima i tehnologijama.

Već sada postoje AI alati kojima je dovoljno napisati koje testove želimo generirati, te će alat prilikom svakog pokretanja provjere aplikacije generirati nove automatizacijske testove i prilagoditi se promjenama u aplikaciji. Također treba paziti da se kod aplikacije ne izloži vanjskim stranama, te strogo kontrolirati promjene koje takvi alati rade, kako ne bi oštetili organizaciju za koju radimo. Moramo precizno i jasno naučiti kako komunicirati s AI alatom, te će se u skoroj budućnosti tražiti osobe koje znaju što preciznije definirati upite (eng. prompt) za AI alate.

Kada naučimo dobro koristiti neki alat, bitno je znanje o alatu prenijeti timu. Cijeli tim odgovoran je za kvalitetu aplikacije, te je inženjer za kvalitetu tu da upozori, pomogne i surađuje kako bi svi zajedno proizveli što bolji proizvod, na zadovoljstvo korisnika i naručitelja.

9. Literatura

1. 5 Software testing methods (2018). *Test Automation Resources*. Preuzeto 30.06.2023. iz https://testautomationresources.com/software-testing-basics/software-testing-methods/

2. Ahamed, S. S. (2010). *Studying the feasibility and importance of software testing: An Analysis*. *arXiv preprint arXiv:1001.4193*. Preuzeto 29.06.2023. iz https://arxiv.org/pdf/1001.4193v1

3. Balaji, S., & Murugaiyan, M. S. (2012). *Waterfall vs. V-Model vs. Agile: A comparative study on SDLC*. International Journal of Information Technology and Business Management, 2(1), 26-30. Preuzeto 30.06.2023 iz https://mediaweb.saintleo.edu/Courses/COM430/M2Readings/WATEERFALLVs%20V-MODEL%20Vs%20AGILE%20A%20COMPARATIVE%20STUDY%20ON%20SDLC.pdf

4. Crispin, L., & Gregory, J. (2009). *Agile testing: A practical guide for testers and agile teams*. Pearson Education.

5. Czerwinski et al. (2004): *A diary study of task switching and interruptions*. Preuzeto 01.08.2023. iz https://www.cs.drexel.edu/~dds26/courses/cs680-s11/BK/readings/Czerwinski-CHI04.pdf

6. Dawson et al. (2010): *Integrating Software Assurance into the Software Development Life Cycle (SDLC)*. Preuzeto 01.08.2023. iz https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDLC

7. de Castro Martins, J., Pinto, A. F. M., Goncalves, G. S., Shigemura, R. A. L., Neto, W. C., da Cunha, A. M., & Dias, L. A. V. (2018). Agile Testing Quadrants on Problem-Based Learning Involving Agile Development, Big Data, and Cloud Computing. In *Information Technology-New Generations: 14th International Conference on Information Technology* (pp. 429-441). Springer International Publishing.

8. Deissenboeck, F., Juergens, E., Hummel, B., Wagner, S., Mas y Parareda, B., & Pizka, M. (2008). *Tool support for continuous quality controlling* Preuzeto 20.06.2023. iz https://arxiv.org/pdf/1611.09116v1.pdf

9. Deming, W. E. (2018). *Out of the Crisis, reissue*. MIT press, str. 27.

10. Difference between QA and QC. Preuzeto 29.06.2023. iz https://hr.myservername.com/difference-between-quality-assurance

11. Goodwin University (2020). *What Does a Quality Assurance Manager Do?* Preuzeto 29.06.2023. iz https://www.goodwin.edu/enews/what-does-a-quality-assurance-manager-do/

12. International Software Testing Qualifications Bord (2023). *Improve & Certify your skills*, Preuzeto 01.07.2023. iz https://www.istqb.org/

13. Iqbal, N., & Qureshi, M. (2009). *Improvement of key problems of software testing in quality assurance*. (arXiv preprint) *arXiv:1202.2506*. Preuzeto 26.06.2023. iz https://arxiv.org/abs/1202.2506

14. Matković, P., & Tumbas, P. (2010). A comparative overview of the evolution of software development models. *International Journal of Industrial Engineering and Management*, *1*(4), 163. Preuzeto 30.06.2023. iz https://www.researchgate.net/publication/267711880_A_Comparative_Overview_of_the_Evolution_of_Software_Development_Models

15. Mekni, M. , Buddhavarapu, G. , Chinthapatla, S. and Gangula, M. (2018). *Software Architectural Design in Agile Environments. Journal of Computer and Communications*, 6(1). Preuzeto 30.06.2023. iz https://www.scirp.org/(S(czeh2tfqyw2orz553k1w0r45))/journal/paperinformation.aspx?paperid=81436

16. Otibine, Tobias & Mbuguah, Samuel & Kilwake, Humphrey & Tsinale, Harriet. (2017). Application Lifecycle Management Activities For Quality Assurance In Software Development. International Journal of Trend in Research and Development. 4. 2394-9333. Preuzeto 02.01.2024. iz https://www.researchgate.net/publication/320978845_Application_Lifecycle_Management_Activities_For_Quality_Assurance_In_Software_Development

17. Papakitsos, E. C. (2022). Robust Software Quality Assurance. *Bull. Georg. Natl. Acad. Sci*, *16*(2). Preuzeto 26.06.2023. iz https://www.academia.edu/download/89163037/03_Papakitsos_Informatics.pdf

18. Seljan, S. (2018) Total Quality Management Practice in Croatian Language Service Provider Companies. ENTRENOVA '18 - ENTerprise REsearch InNOVAtion Conference, 431-439. Preuzeto 02.01.2024. iz https://www.researchgate.net/publication/327776881_Total_Quality_Management_Practice_in_Croatian_Language_Service_Provider_Companies

19. QA Touch, Herra P. (2023). *Roles And Responsibilities of QA in Software Development*. Preuzeto 29.06.2023. iz https://www.qatouch.com/blog/roles-and-responsibilities-of-qa-in-software-development/

20. Radziwill, N., & Freeman, G. (2020). Reframing the test pyramid for digitally transformed organizations. *arXiv preprint arXiv:2011.00655*. Preuzeto 30.06.2023. iz https://arxiv.org/ftp/arxiv/papers/2011/2011.00655.pdf

21. Teli, S. N., Majali, V. S., & Bhushi, U. M. (2010). Role of Cost of Quality in the Automotive Industry. In *National Conference on Recent Trends in Mechanical Engineering*. Preuzeto 26.06.2023. iz https://www.researchgate.net/profile/Shivagond-Teli/publication/317428436_Role_of_Cost_of_Quality_in_the_Automotive_Industry/links/5947f58aaca272f02e0ace87/Role-of-Cost-of-Quality-in-the-Automotive-Industry.pdf

22. Vukašinović, M. (2023). Software quality assurance. Preuzeto 02.01.2024. iz https://www.researchgate.net/publication/374755430_SOFTWARE_QUALITY_ASSURANCE

Popis slika

Popis tablica

1. Spot the difference (n.d.). *Razlika između QA i QC*. Preuzeto 29.06.2023. iz https://hr.spot-the-difference.info/difference-between-qa

Računalni alati i tehnike za upravljanje i kontrolu kvalitete u testiranju softvera

Sažetak

Ovaj diplomski rad daje pregled računalnih alata i tehnika za upravljanje i kontrolu kvalitete koji se koriste tijekom testiranja softverskih rješenja. Osiguranje kvalitete („Quality Assurance“, QA) i Kontrola kvalitete („Quality Control“, QC) su dva najbitnija procesa u razvoju računalnih programa (softvera). Oba pristupa imaju istu namjenu i vrlo su bitni za kvalitetan razvoj softvera, no imaju drugačiji pristup, okruženje i alate. Za oba procesa prikazano je trajanje i njihov međusobni odnos. Također moramo znati gdje usmjeriti fokus prilikom korištenja tehnika usmjerenih na proces kvalitete (QA) i tehnika usmjerenih na sam proizvod (QC). Moramo naučiti kako postaviti važne ciljeve i smanjiti pojavu softverskih grešaka, te kako ih otkriti. Tim tehnikama osiguravamo standarde za kontrolu kvalitete prilikom testiranja softverskih proizvoda. Prije svega moramo znati objasniti važnost uloga i odgovornosti tijekom razvojnog procesa vezano za kontrolu kvalitete. Moramo uočiti razliku u odgovornosti kada različiti razvojni timovi sudjeluju u osiguranju kvalitete, tijekom cijelog razvojno-životnog ciklusa (QA) ili kada je za to odgovoran samo tim za testiranje softverskog testnog ciklusa (QC). U glavnom dijelu predstavljene su najbolje prakse i neki od računalnih alata koji nam mogu pomoći u oba procesa, tj. kod osiguranja kvalitete i kontrole kvalitete. Računalne alate podijelili smo u tri kategorije: „Alati za upravljanje i tehničku analizu“, „Alate za inspekciju (analizu programskog koda)“ i „Alate za testiranje“. Na kraju rada postavljene su osnove upravljanja kvalitetom, a predstavljeni su i uvidi, te ključna razmatranja u vezi osiguranja i kontrole kvalitete.

Ključne riječi: osiguranje kvalitete, kontrola kvalitete, softverske greške, testiranje softvera, razvoj softvera

Computer tools and techniques for quality assurance and quality control in software testing

Summary

This master thesis gives an overview of Quality Assurance and Quality Control tools and techniques used in software testing. Quality Assurance (QA) and Quality Control (QC) are the two most important processes in the software development process. Both have the same purpose and are very important for quality software development, but they have different approaches, environments and tools. The duration of both processes and how they relate to each other are explained. In addition, it is important to know where to place the focus, when using the process-oriented techniques (QA) and product-oriented techniques (QC). We need to learn how to set important goals, minimize software bugs and learn how to detect them. This is needed in order to ensure quality control standards while testing software products. First, we must be able to explain the importance of roles and responsibilities in the development process regarding Quality Assurance. We need to establish the difference in responsibilities when various software teams are involved in the quality assurance process during the entire development life cycle (QA), or when only the testing team takes the responsibility in the entire development software testing life cycle (QC). In the main part of this thesis, the best practices are presented and some of the tools that can assist in both the Quality assurance and the Quality Control processes are analyzed. The tools were divided into categories such as: “Management and technical analyses tools”, “Inspection tools (code reviews)” and “Testing tools”. At the end of this thesis, the basics of quality management are laid out, and insights and key considerations regarding quality assurance and control are presented.

Key words: quality assurance, quality control, software bugs, software testing, software development