Uklanjanje pogrešaka i testiranje softvera

Kurtić, Anđela

SVEUČILIŠTE U ZAGREBU

FILOZOFSKI FAKULTET

ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI

Ak. god. 2023./2024.

Anđela Kurtić

Uklanjanje pogrešaka i testiranje softvera

Završni rad

Mentor: doc. dr. sc. Ivan Dunđer

Zagreb, lipanj 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.

(potpis)

Sadržaj

1. Uvod

U svijetu sveprisutne digitalizacije, softver je postao temelj naše svakodnevice. Od mobilnih aplikacija koje koristimo za komunikaciju do kompleksnih sustava koje korporacije koriste za vođenje svojih poslova, pouzdanost softvera postaje ključni čimbenik uspjeha. U skladu s ubrzanjem razvoja tehnologije, potreba za softverom koji radi bez grešaka postaje sve važnija. Međutim, ni najpažljivije planiranje i najkvalitetniji razvoj ne mogu jamčiti potpunu ispravnost softvera. Softver može nastaviti skrivati nedostatke i pogreške koje mogu ugroziti njegovu funkcionalnost i izvedbu.

Pogreške u softveru ne samo da mogu uzrokovati frustraciju kod korisnika, već mogu imati i ozbiljne posljedice, uključujući gubitak podataka, financijske gubitke ili čak ugrožavanje sigurnosti. Iz tog razloga, testiranje softvera i uklanjanje pogrešaka postaju neizbježni koraci u razvojnom procesu i životnom ciklusu softvera. Testiranje softvera predstavlja sustavan proces provjere svih aspekata softverskog sustava kako bi se osigurala njegova funkcionalnost, pouzdanost, sigurnost i izvedba.

Ovaj proces nije samo koristan, već i nužan za osiguravanje kvalitete proizvoda i zadovoljstva korisnika. Tijekom testiranja programa dolazi do pronalaska pogrešaka koje uzrokuju probleme u programu. Kada se pronađe greška u softveru, proces uklanjanja pogrešaka zahtijeva detaljno istraživanje uzroka problema i implementaciju ispravaka. Ovo može uključivati pregledavanje izvornog koda ili korištenje alata za upravljanje greškama kako bi se pratio i dokumentirao svaki korak procesa otklanjanja grešaka. Uključuje detaljnu analizu, rješavanje problema i modificiranje baze koda kako bi se uklonile pogreške.

Iako testiranje softvera i uklanjanje pogrešaka predstavljaju dodatni trošak i vrijeme u razvojnom procesu, njihova važnost ne može se zanemariti. Investiranje u temeljito testiranje i uklanjanje pogrešaka može dugoročno rezultirati smanjenjem troškova održavanja, povećanom produktivnošću razvojnog tima te poboljšanim korisničkim iskustvom. U konačnici, kvalitetan softver koji radi bez grešaka postaje ključni konkurentski faktor na sve zahtjevnijem tržištu softverskih proizvoda.

U ovom radu objasnit će se što je testiranje softvera, kakve vrste testiranja postoje, koji su razni modeli testiranja te što je životni proces testiranja softvera. Objasnit će se proces uklanjanja pogrešaka, koje metode i pristupi se koriste te kakvi alati postoje na tržištu.

2. Testiranje softvera

2.1. Softver

Kako bi razumjeli što je testiranje softvera, potrebno je prvo objasniti što je softver. Softver se nalazi svuda oko nas i cijelo vrijeme smo okruženi nekom vrstom softvera. Softver se definira kao skup uputa, podataka ili programa koji se koriste za upravljanje računalima i izvršavanje određenih zadataka (Rosencrance, 2021). Softver je suprotnost hardveru jer se hardver odnosi na fizičke komponente računala poput matične ploče ili tipkovnice.

Karakteristike softvera su funkcionalnost, učinkovitost, pouzdanost, održivost, prenosivost i upotrebljivost (Geeksforgeeks.org: Software Characteristics – Software Engineering, 2024). Funkcionalnost je karakteristika softvera koja se odnosi na skup značajki i mogućnosti koje softverski program ili sustav pruža svojim korisnicima.

Učinkovitost je karakteristika softvera koja se odnosi na njegovu sposobnost da koristi pojedine računalne resurse na optimalan način. Ti resursi mogu biti memorija, snaga procesora itd. Visoka učinkovitost postiže se kada softver može obavljati sve svoje predviđene funkcije uz minimalnu upotrebu resursa.

Pouzdanost je karakteristika softvera koja se odnosi na njegovu sposobnost da ispravno i dosljedno tijekom vremena obavlja predviđene funkcije.

Održivost je karakteristika softvera koja se odnosi na lakoću kojom se izvršavaju izmjene u softverskom sustavu.

Prenosivost je karakteristika softvera koja se odnosi na sposobnost softvera da se prenese iz jednog okruženja u drugo bez minimalnih promjena.

Upotrebljivost je karakteristika softvera koja se odnosi na opseg u kojem se softver može koristiti s lakoćom.

Softver se dijeli na sistemski softver i aplikacijski softver (Geeksforgeeks.org: Software and its Types, 2023). Sistemski softver je softver koji izravno upravlja hardverom računala i pruža osnovnu funkcionalnost korisnicima. Sistemski softver podrazumijeva operacijski sustav (Windows, Linux i sl.), procesiranje jezika (pretvaranje programskog koda u naredbe) i *drivere*

uređaja (pogonski program koji kontrolira neki uređaj i pomaže pri izvođenju funkcija tog uređaja, npr. driver za tipkovnicu ili pisač).

Aplikacijski softver je softver koji je dizajniran za obavljanje određenog zadatka za krajnje korisnike. Aplikacijski softver dijeli se na softver opće namjene, prilagođeni softver i uslužni softver.

Softver opće namjene, eng. „general purpose software“, koristi se za različite zadatke i nije ograničen na izvođenje samo jednog određenog zadatka. Primjeri ove vrste softvera su MS-Word i MS-Excel.

Prilagođeni softver, eng. „customized software“, koristi se za obavljanje specifičnih zadataka i dizajniran je za određene organizacije. Primjerice, koristi se u zrakoplovnom sustavu, sustavu željeznica itd.

Uslužni softver, eng. „utility software“, koristi se za podršku računalne infrastrukture, dizajniran je za analizu, konfiguraciju, optimizaciju i održavanje sustava te brine o njegovim zahtjevima. Primjeri ove vrste softvera su antivirusni programi, testeri memorije, čitači diska itd.

Softver su mobilne igrice, aplikacije za online kupovinu, internet bankarstvo, web stranice, strojni prevoditelji i mnoge druge svakodnevne stvari koje su postale neizbježan dio današnje rutine u svijetu visoko napredne tehnologije. Vrlo je važno da sve te stavke funkcioniraju onako kako se očekuje kako bi se izbjegle neugodne pogreške, poput sporog učitavanja web stranice ili problema s transakcijama u internetskim trgovinama. Kako bi se izbjegli potencijalni problemi, softverski proizvodi, tj. programi, moraju se konstantno i temeljito testirati prije nego što izađu na tržište kako bi se održala kvaliteta proizvoda i zadovoljstvo korisničkog iskustva.

2.2 Testiranje softvera

Međunarodni Odbor Kvalifikacija za Testiranje Softvera, eng. International Software Testing Qualifications Board (ISTQB), definira testiranje softvera kao proces unutar životnog ciklusa razvoja softvera koji ocjenjuje kvalitetu komponente ili sustava i povezanih radnih proizvoda (ISTQB Glossary: Testing, bez dat.). Osobe koje testiraju softver nazivaju se „testerima“, a pogreške koje se pronađu tijekom testiranja nazivaju se „bug“. Testeri ručno komuniciraju sa

softverom ili izvršavaju testne skripte za pronalaženje bugova, pritom osiguravajući da softver radi onako kako se očekuje (Geeksforgeeks.org: What is Software Testing?, 2024).

Bitno je naglasiti da se kroz testiranje provjerava jesu li ispunjena poslovna očekivanja, postoje li nedostaci u zahtjevima koje je potrebno odmah riješiti i kakva je valjanost koda. Testiranje softvera važan je dio razvojnog ciklusa softvera jer u slučaju da testiranje nije provedeno, može doći do skrivenih bugova u programu koje u nekom trenutku nisu otkrivene. Tijekom daljnje upotrebe tog softvera, ti bugovi „izlaze na površinu“ i negativno utječu na korištenje programa. Danas su aplikacije, programi i ostale verzije softvera puno kompliciranije te imaju kompleksne i velike kodove koji su „plodno tlo“ za razne bugove.

Testiranje softvera se koristi za identifikaciju ranjivosti u financijskom, medicinskom, pravnom ili bilo kojem softveru koji se bavi osjetljivim informacijama. Aplikacije koje se bave ovom vrstom softvera, ne mogu si dopustiti zastoje, oštećenje podataka ili kvarove sustava jer to direktno utječe na ljudske živote (Graham, Veeneendaal, Evans i Black, 2008). Ponekad greške u ovakvim softverima mogu dovesti do nepopravljive štete, velikog gubitka tvrtke i ugroziti ljudske živote. Iz tog razloga naglašava se važnost reguliranja kvalitete i smanjenja količine bugova, kako bi se izbjegli potencijalni problemi poput rušenja aplikacija ili programa, kako bi aplikacije i programi mogli nesmetano raditi i, najvažnije, pružati uslugu za koju su namijenjeni.

Važnost testiranja leži u razvoju aplikacija te njihovom nesmetanom korištenju. Nažalost, razvoj softvera nije linearan i aplikacije su jako osjetljive na pogreške i nedostatke. Otkrivanje tih nedostataka jedna je od najvažnijih zadaća koje razvojni tim treba riješiti. U suštini, testiranje softvera je identifikacija pogrešaka i nedostataka.

Moderni softver sastoji se od međusobno povezanih komponenti koje moraju raditi zajedno kako bi omogućile željenu izvedbu (Katalon.com: What is Software Testing? Definition, Types, and Tools, bez dat.). Jedna neispravna komponenta može imati katastrofalne rezultate i srušiti cijele aplikacije i programe. Brzim otklanjanjem pogrešaka u kodu izbjegavaju se potencijalni problemi koji bi mogli prouzročiti ozbiljne poteškoće.

Pored nesmetanog korištenja, važno je održavati i poboljšavati kvalitetu softvera. Kada se razmišlja o kvaliteti, to se razmišljanje zapravo odnosi na značajke softvera koje, ne samo da ispunjavaju očekivanja korisnika ili kupaca, nego ih nadmašuju. Od svake aplikacije ili softvera

očekuje se da rade pravilno, a kako bi se osigurala pravilnost rada, potrebno je održavati i poboljšavati proizvod. Softver se održava i poboljšava tako što se periodično testiraju svojstva upotrebljivosti, kompatibilnosti i sigurnosti koda kako bi se osiguralo da navedeni aspekti ispunjavaju zadane standarde. Održavanje softvera pruža uvid u kvalitetu proizvoda razvojnom timu i pomaže pri ažuriranju postojeće dokumentacije vezane za taj proizvod kako bi se mogao kontinuirano poboljšavati.

Testiranje softvera važno je za povećanje povjerenja i zadovoljstvo samih korisnika. U većini slučajeva nije racionalno očekivati da će proizvod biti apsolutno bez greške, ali bitno je dostići razinu stabilnosti, pouzdanosti i konstantnog zadovoljavanja potreba korisnika koji će na kraju dovesti do pozitivnog korisničkog iskustva. Pouzdanost, stabilnost i zadovoljstvo postiže se najboljom praksom upravljanja kvalitetom softvera te višestrukim testiranjem samog proizvoda (Katalon.com: What is Software Testing? Definition, Types, and Tools, bez dat.).

Ukratko, važnost testiranja softvera leži u poboljšanju kvalitete proizvoda, identifikaciji područja za poboljšanje, otkrivanju i ispravljanju pogrešaka kako bi sam softver bio pouzdaniji, u osiguravanju softvera da radi onako kako se očekuje, provjeravanju ispunjenja korisničkih očekivanja i osiguranju integracije softvera.

2.3. Verifikacija i validacija softvera

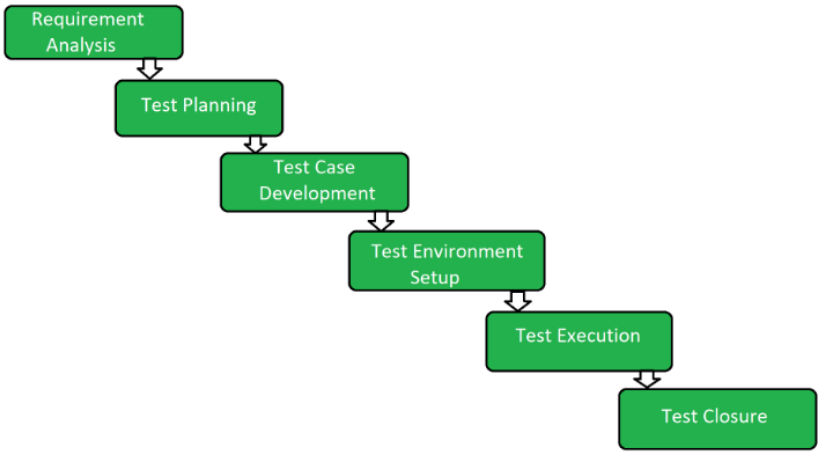
Postoje dva veoma važna procesa koja su uključena u provjeru zadovoljava li softver zadane uvjete i specifikacije te ispunjava li u potpunosti svoju svrhu, a to su procesi verifikacije i validacije.

Verifikacija je proces provjere postiže li softver svoje ciljeve bez pojave pogrešaka. Ovim procesom osigurava se da je proizvod ispravan i da ispunjava sve zadane uvjete. Osiguravanje kvalitete najvažniji je faktor u verifikacijskom testiranju. Verifikacijsko testiranje naziva se i statičkim testiranjem (Geeksforgeeks.org: Differences between Verification and Validation, 2024). Testiranje ove vrste pomaže pri identifikaciji pogrešaka u ranoj fazi razvoja softvera te se izvršava prije validacijskog testiranja. Time pomaže u prevenciji pogrešaka. Metode koje se koriste u verifikaciji su pregledi, provjere i inspekcije. Ono odgovara na pitanje „izrađuje li se proizvod onako kako treba?“. U ovom procesu ne dolazi do izvršavanja koda (Javatpoint.com: Verification and Validation Testing, bez dat.).

Validacija je proces provjere valjanosti proizvoda, odnosno provjerava se da je ono što se razvija pravi proizvod. Kontroliranje kvalitete najvažniji je faktor u validacijskom testiranju. Validacijsko testiranje naziva se još i „dinamičko testiranje“ (Geeksforgeeks.org: Differences between Verification and Validation, 2024).. Kod validacijskog testiranja se pronalaze pogreške koje se nisu uspjele pronaći u fazi verifikacije. Validacijom dolazi do pronalaženja pogrešaka u kodu. Metode koje se koriste u validaciji su testiranje crne kutije, testiranje bijele kutije i nefunkcionalno testiranje, koji će se objasniti kasnije u radu. U ovom procesu dolazi do izvršavanja koda. Proces validacije odgovara na pitanje „izrađuje li se pravi proizvod?“ (Javatpoint.com: Verification and Validation Testing, bez dat.).

2.4 Životni ciklus testiranja softvera

Životni ciklus testiranja softvera ili tzv. STLC (eng. Software Testing Life Cycle) sustavni je pristup koji obuhvaća različite faze i aktivnosti koje se provode tijekom procesa razvoja softvera kako bi se osigurala kvaliteta i funkcionalnost softverskog proizvoda (Guru99.com: STLC (Software Testing Life Cycle), 2024).



Slika 1. Životni ciklus testiranja softvera (Geeksforgeeks.org: Software Testing Life Cycle (STLC), 2024)

STLC sastoji se od 6 ključnih faza kako bi se osiguralo postizanje svih ciljeva kvalitete softvera, a one glase: analiza zahtjeva, planiranje testiranja, razvoj testnog slučaja, postavljanje testnog okruženja, izvođenje testa, i zatvaranje ciklusa ispitivanja (Guru99.com: STLC (Software Testing Life Cycle), 2024). Svaka od faza ima definirane ulazne i izlazne kriterije, aktivnosti i rezultate povezane s njima (Slika 1).

Prva faza je analiza zahtjeva gdje tester ispituje zahtjeve navedene od klijenta te izrađuje plan testiranja kako bi se utvrdilo ispunjava li softver sve zahtjeve. Aktivnosti koje se provode u ovoj fazi su identifikacija vrste testa koji će se izvoditi, pripremanje matrice praćenja zahtjeva (Requirement Traceability Matrix, RTM) koja služi za mapiranje zahtjeva za testne slučajeve, prikupljanje detalja o prioritetima testiranja i identifikacija detalja okruženja u kojemu bi se test trebao odviti. Rezultati koje ova faza nosi su RTM i izvješće o izvedivosti automatizacije ako je primjenjivo (Guru99.com: STLC (Software Testing Life Cycle), 2024).

Druga faza je stvaranje plana testiranja gdje se definiraju sve strategije testiranja, odnosno gdje tester određuje procijenjeni napor i trošak cijelog projekta. Aktivnosti koje se provode u ovoj fazi su pripreme testnog plana za različite vrste testiranja, izbor alata za testiranje, procjena napora testa, planiranje resursa i određivanje uloga odgovornosti. Rezultati koje ova faza donosi su plan testiranja ili strateški dokument te dokument o procjeni napora (Guru99.com: STLC (Software Testing Life Cycle), 2024).

Treća faza je razvoj testnog slučaja gdje se stvaraju, provjeravaju i prerađuju testni slučajevi i testne skripte nakon što je planiranje testa završeno. Aktivnosti vezane za ovu fazu su stvaranje testnog slučaja i skripti za automatizaciju, pregledavanje osnovnih testnih slučajeva i skripti i stvaranje testnih podataka. Rezultati ove faze su testni slučajevi i skripte te podaci o testu (Guru99.com: STLC (Software Testing Life Cycle), 2024).

Četvrta faza je postavljanje testnog okruženja koje odlučuje o softverskim i hardverskim uvjetima pod kojima će se testirati. Ova faza smatra se jednom od važnijih dijelova procesa životnog ciklusa testiranja softvera i može se paralelno izrađivati s fazom razvoja testnog slučaja. Aktivnosti koje se u ovoj fazi provode su razumijevanje arhitekture sustava, postavljanje okruženja i pripremanje popisa hardverskih i softverskih zahtjeva za testno okruženje. U ovoj fazi se može provoditi i test dima koji će se objasniti kasnije u radu. Rezultati koje ova faza nosi su spremno okruženje s postavljenim testnim podacima i rezultat testa dima (Guru99.com: STLC (Software Testing Life Cycle), 2024).

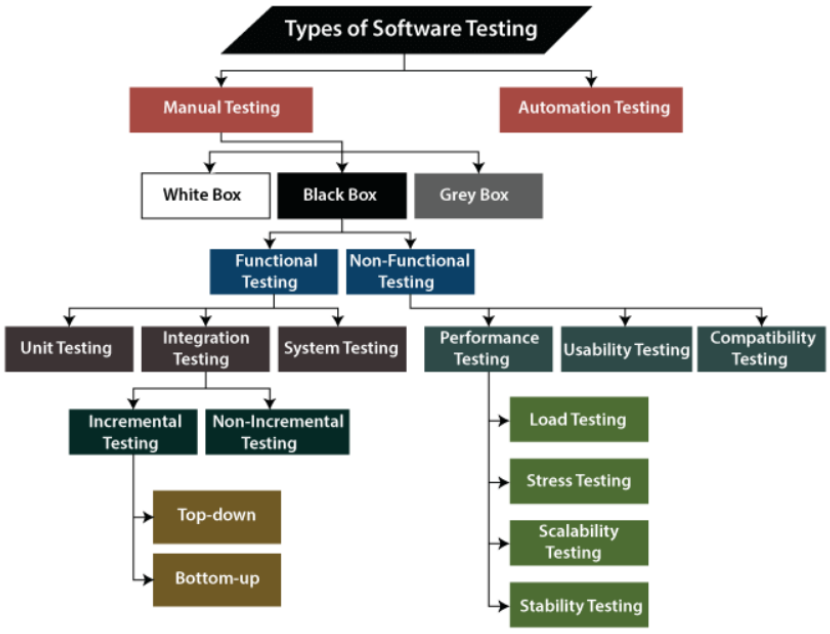
Peta faza procesa je izvođenje testiranja, gdje testeri počinju s razvojem i izvršenjem testnih slučajeva. Oni sastavljaju detaljne testne slučajeve i pripremaju potrebne testne podatke. U ovoj fazi testeri provode testiranje prema planovima testiranja i unaprijed pripremljenim testnim slučajevima. Proces uključuje izvođenje testnih skripti, njihovo održavanje te prijavljivanje

otkrivenih grešaka. Ako se otkriju greške, softver se vraća razvojnom timu na ispravak, a testiranje se zatim ponovno provodi. Aktivnosti koje se provode su izvođenje testa, dokumentiranje rezultata i pogrešaka, mapiranje pogrešaka u testnim slučajevima i ponovno testiranje popravka pogreške. Rezultati ove faze su aktualizirani testni slučajevi s rezultatima, izvješća o greškama i dovršeni RTM sa statusom izvršenja (Guru99.com: STLC (Software Testing Life Cycle), 2024).

Šesta faza je zatvaranje ciklusa testiranja gdje se procjenjuje strategija razvoja, postupak testiranja i mogući nedostaci za buduće softvere s istom specifikacijom. Aktivnosti koje se provode su ocjena kriterija završetka ciklusa na temelju vremena, pokrivenosti testom, cijene, ključnih poslovnih ciljeva i kvalitete, pripreme izvješća o završetku testa i analize rezultata kako bi se otkrila distribucija pogrešaka po tipu i fatalnosti. Rezultat ove faze je izvješće o završetku testa (Guru99.com: STLC (Software Testing Life Cycle), 2024).

2.5. Vrste testiranja softvera

Postoji mnogo različitih vrsta testiranja koje se dijele na druge podvrste. Koriste se za provjeru različitih aspekata funkcionalnosti, sigurnosti i kvalitete softvera. Međutim, samo su dvije glavne podjele na kojima se ostale vrste testiranja temelje, a to su automatsko i ručno testiranje (Slika 2) (Geeksforgeeks.org: Types of Software Testing, 2024).



Slika 2. Vrste testiranja softvera (Javatpoint.com: Types of Software Testing, bez dat.)

2.5.1 Automatsko i ručno testiranje

Automatsko testiranje je proces korištenja softverskih alata za testiranje softvera. Ovim procesom omogućuje se pouzdano i brzo testiranje softvera bez potrebe za ručnim testiranjem softvera. Automatsko testiranje je tehnika gdje tester napiše skriptu i onda koristi odgovarajući softver ili alat za automatizaciju kako bi testirao zadani program (Geeksforgeeks.org: Types of software testing, 2024). Prednosti automatiziranog testiranja leže u jednostavnosti izvršavanja testa, u poboljšanju pouzdanosti testa, povećavanju količine pokrivenosti testa i smanjuje ljudsku interakciju (Geeksforgeeks.org: Types of software testing, 2024). Neki od popularnih alata za testiranje softvera su Selenium, TestRail, Appium, Katalon, Ranorex, Test Complete, SpiraTest i mnogi drugi (Zaptest.com: Alati za testiranje softvera – 30 najboljih proizvoda za testiranje softvera na tržištu u 2024, bez dat.).

Ručno testiranje softvera je proces u kojem ljudski testeri ručno izvršavaju testove kako bi provjerili funkcionalnost, performanse, sigurnost ili druge karakteristike softverskog proizvoda. Ručno testiranje softvera može se nazvati i tradicionalnim načinom testiranja jer uključuje direktnu interakciju testera s aplikacijom kako bi se simuliralo stvarno korisničko iskustvo. Ono što je karakteristično za ručno testiranje su testni slučajevi, ručno izvođenje, rezultati i bilješke, istraživačko testiranje, korisničko iskustvo i ispitivanje granica (Geeksforgeeks.org: Manual Testing – Software Testing, 2024). Testne slučajeve razvija sam tester ili koristi već postojeće kako bi obavio testiranje. Ručno izvođenje odnosi se na testera koji ručno izvršava testne slučajeve koristeći se raznim scenarijima i podacima kako bi se provjerila funkcionalnost softvera. Tester bilježi rezultate svakog testa, uključujući identificirane pogreške ili probleme. Istraživačko testiranje je oblik ručnog testiranja gdje tester istražuje softver bez unaprijed definiranih testnih slučajeva i pritom traži neobična ponašanja ili skrivene greške. Ručno testiranje omogućuje testerima da ocijene korisničko iskustvo koje se veže na korisničko sučelje, navigaciju i interakciju s aplikacijom. Ispitivanje granica omogućuje testeru da provjeri kako se softver ponaša u ekstremnim uvjetima ili na granici svojih kapaciteta. Iako ručno testiranje omogućuje niz prednosti, velik je nedostatak što zahtijeva puno vremena, podložno je ljudskoj grešci i manje je efikasno od automatiziranog testiranja. Ručno testiranje važan je dio procesa za osiguravanje kvalitete, posebno za funkcionalnosti koje nisu jednostavne za automatizirati i omogućuje ljudskim testerima da primijete probleme koji se ne bi mogli otkriti automatiziranim testiranjem.

Ručno testiranje krovni je izraz za ostale vrste i podvrste testiranja, a ono se dijeli na testiranje crne i bijele kutije, odnosno „black box“ i „white box“ testiranje (Geeksforgeeks.org: Types of Software Testing, 2024).

2.5.2 Testiranje bijele i crne kutije

Testiranje bijele kutije usredotočuje se na unutarnje strukture i detalje izvornog koda, a testiranje crne kutije na vanjsko ponašanje softvera bez obzira na implementaciju (Geeksforgeeks.org; Differences between Black Box Testing and White Box Testing, 2024). Oba pristupa su važna i koriste se zajedno kako bi se osigurala potpuna pokrivenost i kvaliteta softvera. Testiranje bijele kutije je kategorija testiranja softvera koja se fokusira na internu strukturu i dizajn softvera (Zaptest.com: Testiranje bijele kutije: što je to, kako funkcionira, izazovi, metrika, alati i više!, bez dat.). Testiranje bijele kutije se ponekad može nazivati i transparentnim testiranjem ili testiranje otvorene kutije (Geeksforgeeks.org: Differences between Black Box Testing and White Box Testing, 2024). Prednosti testiranja bijele kutije su temeljito testiranje budući da se testira cijeli kod i struktura, također optimizira kod uklanjanjem pogrešaka, pomaže u uklanjanju dodatnih redova koda, rano otkriva nedostatke, može započeti u najranijoj fazi jer ne zahtijeva nikakvo sučelje za testiranje i otkriva složene nedostatke koji se ne mogu otkriti drugim tehnikama testiranja.

Testiranje crne kutije odnosi se na proces testiranja sustava ili dijela softvera bez ikakvog prethodnog znanja o načinu na koji sustav funkcionira iznutra. Testiranje crne kutije temelji se na vanjskom ponašanju softvera, a testeri ne trebaju pristup izvornom kodu i testiraju softver samo na temelju njegovih ulaznih i izlaznih kriterija, kao što bi to činio najobičniji korisnik (Zaptest.com: Testiranje crne kutije – što je to, vrste, procesi, pristupi, alati i više!, bez dat.).

Testiranje crne kutije dijeli se na druge dvije vrste testiranja, a to su funkcionalno i nefunkcionalno testiranje (Geeksforgeeks.org: Types of Software Testing, 2024). Funkcionalno testiranje je vrsta testiranja softvera gdje se provjerava je li aplikacija dala očekivani rezultat. Kod funkcionalnog testiranja nije bitno kako se obrada odvija, već daje li obrada ispravne rezultate ili pokazuje pogreške (Zaptest.com: Što je funkcionalno testiranje? Vrste, primjeri, kontrolni popis i implementacija, bez dat.). Testiranje je usmjereno na procjenu funkcija ili komponenti softvera kako bi se osiguralo da svaka funkcija ispravno radi. Funkcionalno testiranje najčešće se koristi

kod testiranja funkcija unosa podataka, testiranja funkcija pretraživanja te testiranja korisničkog sučelja.

Nefunkcionalno testiranje je testiranje čiji je cilj provjera performansi, sigurnosti, pouzdanosti, upotrebljivosti i drugih nefunkcionalnih aspekata softvera. Nefunkcionalno testiranje je neophodno jer ocjenjuje bitne korisničke kriterije kao što su izvedba i upotrebljivost te provjerava radi li softver prema očekivanjima izvan svoje osnovne funkcionalnosti (Zaptest.com: Nefunkcionalno testiranje: što je to, vrste, pristupi, alati i više!, bez dat.). Testiranje se fokusira na provjeru kako softver reagira u različitim uvjetima na različite zahtjeve.

2.5.3 Vrste funkcionalnog testiranja

Funkcionalno testiranje dijeli se na jedinično testiranje, integracijsko testiranje i sistemsko testiranje. Jedinično testiranje je metoda testiranja individualnih jedinica ili komponenti softvera (Geeksforgeeks.org: Types of Software testing, 2024). Jedinične testove provode programeri i koriste se kako bi se osiguralo da sve jedinice rade kako je predviđeno. Ova metoda testiranja osigurava da čak i najmanji funkcionalni dijelovi softvera rade ispravno. Prednosti jediničnog testiranja su da „hvataju“ greške rano u razvojnoj fazi, osiguravaju da promjene u kodu ne uvode dodatne greške, čine kod jednostavnijim za razumijevanje i održavanje te poboljšavaju ukupnu kvalitetu i pouzdanost softvera (MoldStud.com: Exploring the benefits of unit testing, 2024). Neki od alata za jedinično testiranje su JUnit, NUnit i xUnit. Treba naglasiti da je jedinično testiranje samo jedan oblik testiranja softvera i treba ga koristiti u kombinaciji s drugim vrstama testiranja kako bi se osiguralo da softver zadovoljava potrebe korisnika (Geeksforgeeks.org: What is Software Testing?, 2024).

Integracijsko testiranje odnosi se na proces testiranja sučelja između dviju komponenti ili softverskih modula kako bi se procijenio prijenos podataka između njih (Zaptest.com: Što je integracijsko testiranje? Duboko zaronite u vrste, procese i implementaciju, bez dat.). Koristi se za prepoznavanje i rješavanje problema koji se mogu pojaviti tijekom integracije različitih softverskih komponenti. Integracijsko testiranje obično se odvija nakon jediničnog testiranja jer je važno da se provede tek kada su pojedinačne komponente već testirane i kada je potvrđeno da rade ispravno. Postoje različiti načini izvođenja integracijskog testiranja, a to su top-down integracijsko testiranje, tj. odozgo prema dolje, zatim bottom-up integracijsko testiranje, tj. odozdo prema gore, *big bang* integracijsko testiranje te inkrementalno integracijsko testiranje (Geeksforgeeks.org:

Types of Software Testing, 2024). Testiranje odozgo prema dolje provodi se iz modula visoke razine i odvaja ga od modula niske razine. Testiranje odozdo prema gore započinje testom niske razine i povezuje ga s testom visoke razine. Big bang test integrira sve module i integrira ih istovremeno. Inkrementalno testiranje kombinira module u male grupe i testira svaku grupu kako se dodaje.

Sistemsko testiranje je vrsta testiranja softvera kojim se procjenjuje ukupna funkcionalnost i učinkovitost cjelovitog i potpuno integriranog softverskog rješenja (Geeksforgeeks.org: Types of software testing, 2024). Ovim testiranjem provjerava se ispunjava li sustav zahtjeve i je li prikladan za isporuku krajnjem korisniku. Ova vrsta testiranja provodi se nakon integracijskog testiranja jer se komponente koje prođu integracijski test koriste kao ulaz.

2.5.4 Vrste nefunkcionalnog testiranja

Nefunkcionalno testiranje dijeli se na testiranje performansi, testiranje upotrebljivosti i testiranje kompatibilnosti.

Testiranje performansi je vrsta testiranja softvera koja osigurava ispravno funkcioniranje softverskih aplikacija pod očekivanim radnim opterećenjem. To je tehnika testiranja koja se provodi kako bi se odredila izvedba sustava u smislu osjetljivosti, reaktivnosti i stabilnosti pod određenim radnim opterećenjem (Geeksforgeeks.org: Types of software testing, 2024).

Testovi performansi su vrsta nefunkcionalnog testa koji provjerava koliko dobro rade različite softverske komponente. Svrha testiranja performansi je izmjeriti performanse sustava pod različitim opterećenjima i osigurati da sustav može podnijeti očekivani broj korisnika ili procesa (Geeksforgeeks.org: Types of software testing, 2024).

Testiranje upotrebljivosti koristi se za procjenu proizvoda ili usluga testiranjem s korisnicima. U testiranju upotrebljivosti, tim za razvoj i dizajn će se koristiti ovom vrstom testiranja kako bi identificirali probleme prije programiranja, a kako bi se greške pravovremeno riješile. Testiranje kompatibilnosti koristi se nad softverom kako bi se provjerila kompatibilnost na različitim platformama i okruženjima. Takvo testiranje odvija se tek kada softver postane stabilan. Ova vrsta testiranja je veoma važna u razvoju i implementaciji softvera i koristi se da bi se izbjegli budući problemi vezani za kompatibilnost (Geeksforgeeks.org: Types of software testing, 2024).

2.5.5 Vrste testiranja performansi

Testiranje performansi sastoji se od nekoliko vrsta testiranja, a to su testiranje opterećenja, testiranje otpornosti na stres, testiranje skalabilnosti i testiranje stabilnosti (Lakra, 2022).

Testiranje opterećenja temelji se na ponašanju aplikacija kada ih koristi više korisnika u isto vrijeme. Simulira opterećenje koje bi se moglo dogoditi u stvarnom svijetu i promatra kako se softver ponaša pod stresom.

Kod testiranja otpornosti na stres, softver se stavlja pod ekstremne i nepoželjne uvjete kako bi se provjerilo kako sustav funkcionira pod takvim uvjetima. Ovi se testovi provode kako bi se izmjerila brzina i izvedba programa.

Testiranje skalabilnosti fokusira se na procjenu sposobnosti softverske aplikacije, sustava, mreže ili procesa da se prilagodi povećanju ili smanjenju opterećenja. Cilj testiranja skalabilnosti je osiguranje stabilne i pouzdane izvedbe bez obzira na promjenu u opterećenju.

Testiranje stabilnosti je vrsta testiranja koja se fokusira na provjeru kako softver reagira na različite parametre okoline. To je sposobnost proizvoda da nastavi funkcionirati tijekom dužeg vremena bez kvara ili prekida.

2.5.6 Druge vrste testiranja softvera

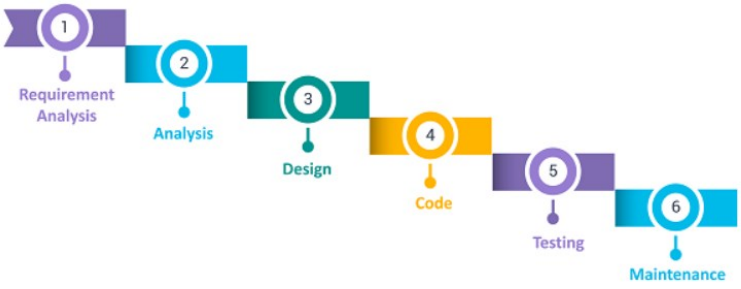
Postoji još mnogo drugih vrsta testiranja koje vrijedi spomenuti: testiranje dima, testiranje uračunljivosti, regresijsko testiranje, testiranje prihvatljivosti, testiranje korisničke prihvatljivosti, eksplorativno testiranje, ad hoc testiranje, sigurnosno testiranje, globalizacijsko testiranje, alfa testiranje, beta testiranje i sl. (Geeksforgeeks.org: Types of software testing, 2024).

Testiranje dima koristi se kako bi se osiguralo da je softver koji se testira dovoljno stabilan za izvođenje testova. Testiranje uračunljivosti spada pod regresijsko testiranje i koristi se kako bi se osiguralo da promjene u kodu ispravno rade. Regresijsko testiranje je proces testiranja modificiranih dijelova koda i dijelova koji bi mogli biti zahvaćeni zbog modifikacija, gdje se osigurava da ne postoji dodatnih problema u kodu nakon što su promjene već napravljene. Testiranje prihvatljivosti provode sami korisnici kako bi provjerili ispunjavaju li isporučeni proizvodi željene uvjete. Testiranje korisničke prihvatljivosti je proces testiranja u kojem krajnji korisnici sudjeluju u testiranju proizvoda kako bi osigurali da proizvod radi u skladu s njihovim

zahtjevima. Eksplorativno testiranje je vrsta testiranja softvera u kojem tester može odabrati bilo koju metodu po svom izboru za testiranje softvera. Ad hoc testiranje je vrsta testiranja koja se provodi neformalno i neočekivano nakon završetka redovnog testiranja kako bi se pronašli nedostaci u sustavu. Sigurnosno testiranje je vrsta testiranja koja identificira ranjivosti u sustavu i utvrđuje jesu li podaci i resursi sustava zaštićeni od neovlaštenog pristupa. Globalizacijsko testiranje vrsta je softverskog testiranja koje se provodi kako bi se vidjelo funkcionira li sustav ili softverska aplikacija bez obzira na geografski i kulturni kontekst. Alfa testiranje je vrsta validacijskog testiranja koje provodi osoblje za osiguranje kvalitete prije nego što se proizvod pusti u prodaju, odnosno isporuči kupcu. Beta testiranje provodi krajnji korisnik softvera na jednom ili više korisničkih mjesta te je ova verzija dostupna samo ograničenom broju korisnika za testiranje u stvarnom okruženju.

2.6 Modeli testiranja softvera

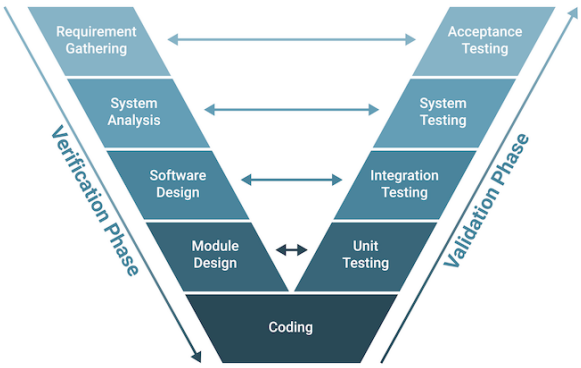
Testiranje softvera izvršava se na razne načine i postoje razni modeli testiranja, a najpoznatiji modeli su tzv. model vodopada, V-model, agilni model, spiralni model i iterativni model (Edureka.co: What are the Types of Software Testing Models?, 2020). U modelu vodopada programeri slijede niz aktivnosti odozgo prema dolje do konačnog cilja. Procesi koji slijede su: analiza zahtjeva, faze analize, dizajn softvera, programska implementacija, testiranje i održavanje (Edureka.co: What are the Types of Software Testing Models?, 2020). Svaka faza razvoja može započeti tek nakon završetka prethodne faze, otuda i naziv modela (Slika 3). Međutim, model može proizvesti povratne petlje između susjednih faza koje zahtijevaju promjene u prethodnoj fazi.



Slika 3. Model vodopada (Edureka.co: What are the Types of Software Testing Models?, 2023)

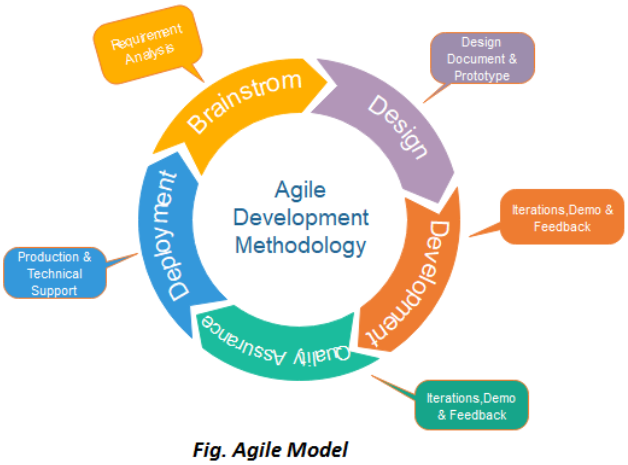
V-model sličan je modelu vodopada, ali se smatra “superiornijim“ jer se aktivnosti razvoja i testiranja provode paralelno. Testiranje počinje na razini jedinice i širi se po cijelom sustavu. Ovaj model poznat je pod nazivom verifikacijski i validacijski model jer se s jedne strane vrši

verifikacija, a s druge validacija. Sastoji se od više faza čija je osnovna podjela na verifikacijske i validacijske faze (Slika 4) (Geeksforgeeks.org: SDLC V-Model – Software Engineering, 2024). Verifikacijski dio V-modela uključuje statičku analizu bez izvršenja koda jer se fokusira na evaluaciju razvoja proizvoda kako bi se utvrdilo jesu li ispunjeni specifični zahtjevi. Faze verifikacije su: analiza poslovnih zahtjeva, dizajn sustava, arhitektura sustava, dizajn modula i faza programiranja (Geeksforgeeks.org: SDLC V-Model – Software Engineering, 2024). Validacijski dio V-modela uključuje dinamičko testiranje koje se vrši izvođenjem koda kako bi se utvrdilo ispunjava li softver očekivanja i zahtjeve korisnika. Faze validacije su: jedinično testiranje, integracijsko testiranje, sistemsko testiranje i test korisničke prihvatljivosti (Geeksforgeeks.org: SDLC V-Model – Software Engineering, 2024). Dijeli se na dvije faze, faza dizajna i testiranja. U fazi dizajna nalaze se faze iz verifikacijskog dijela, a u fazi testiranja faze iz validacijskog dijela (Geeksforgeeks.org: SDLC V-Model – Software Engineering, 2024).



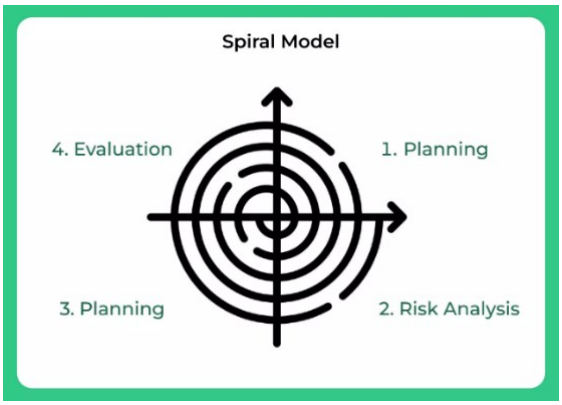
Slika 4. V-model testiranja (Builtin.com: What Is the V-Model in Software Development?, 2023)

Agilni model prepolovljuje zadatak na manje dijelove ili iteracije (Slika 5). Kada se softver podijeli na manje dijelove, odnosno iteracije, onda to pomaže pri smanjenju rizika i zahtjeva prije isporuke proizvoda (Javatpoint.com: Agile Model, bez dat.). Uključuje nekoliko timova koji rade zajedno na razvijanju rješenja i analizi zahtjeva. Agilni model naziva se i iterativnim modelom i karakterizira ga adaptivni fokus na korisnike i njihovo zadovoljstvo (Edureka.co: What are the Types of Software Testing Models?, 2020).



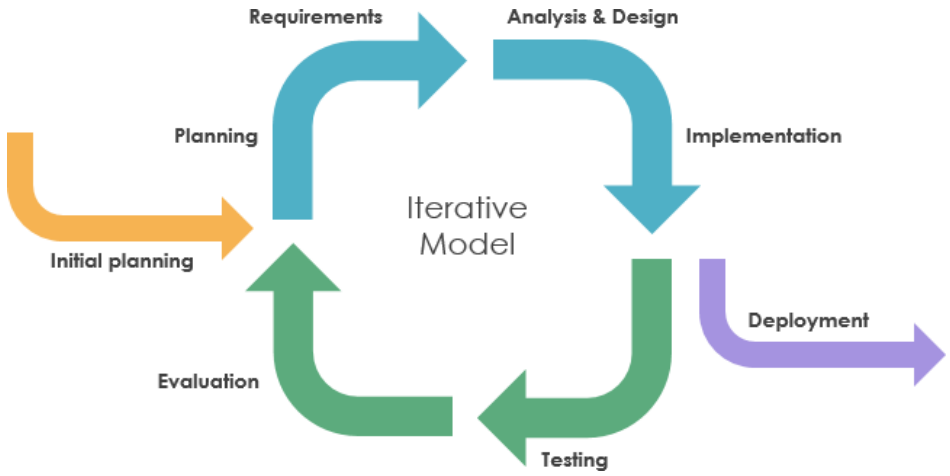
Slika 5. Agilni model (Javatpoint.com: Agile Model, bez dat.)

Spiralni model sličan je agilnom modelu, ali se više fokusira na analizu. Proces projektiranja uključuje različite faze kao što su planiranje, analiza rizika, izgradnja i procjena, kao i zadatke prikupljanja osnovnih zahtjeva i provođenja procjene rizika (Edureka.co: What are the Types of Software Testing Models?, 2020). Funkcionira na način da se prikupe zahtjevi i odradi procjena rizika na osnovnoj razini i onda se svaka sljedeća gornja spirala gradi na tome (Slika 6).



Slika 6. Spiralni model (Prepinsta.com: Spiral Model in SDLC, bez dat.)

Iterativni model ne zahtijeva puni popis zahtjeva prije pokretanja projekta, ali proces počinje sa zahtjevima i funkcionalnim dijelom koji se zatim može proširiti, a sam proces se ponavlja, dopuštajući nove verzije proizvoda u svakoj fazi (Slika 7) (Edureka.co: What are the Types of Software Testing Models?, 2020).



Slika 7. Iterativni model (Medium.com: Software Development Framework — Iterative Model, 2023)

3. Uklanjanje pogrešaka

Uklanjanje pogrešaka ili tzv. „debugging“ je proces prepoznavanja i rješavanja „bugova“, tj. grešaka u softveru (Geeksforgeeks.org: What is Debugging in Software Engineering, 2024). To je važan aspekt softverskog inženjerstva jer greške mogu uzrokovati neispravan rad softvera i dovesti do loših performansi ili netočnih rezultata. Rješavanje problema može biti dugotrajno i teško, ali važno je provjeriti radi li softver ispravno. Kada postoji softverski problem, programeri analiziraju kod kako bi otkrili zašto nešto ne radi ispravno. Koriste različite alate za uklanjanje pogrešaka kako bi ispitali kod korak po korak, pronašli problem i izvršili potrebne ispravke. Uklanjanje pogrešaka razlikuje se od testiranja softvera po tome što se testiranje fokusira na pronalazak pogrešaka, a uklanjanje pogrešaka nastupa tek kada se pogreška identificira.

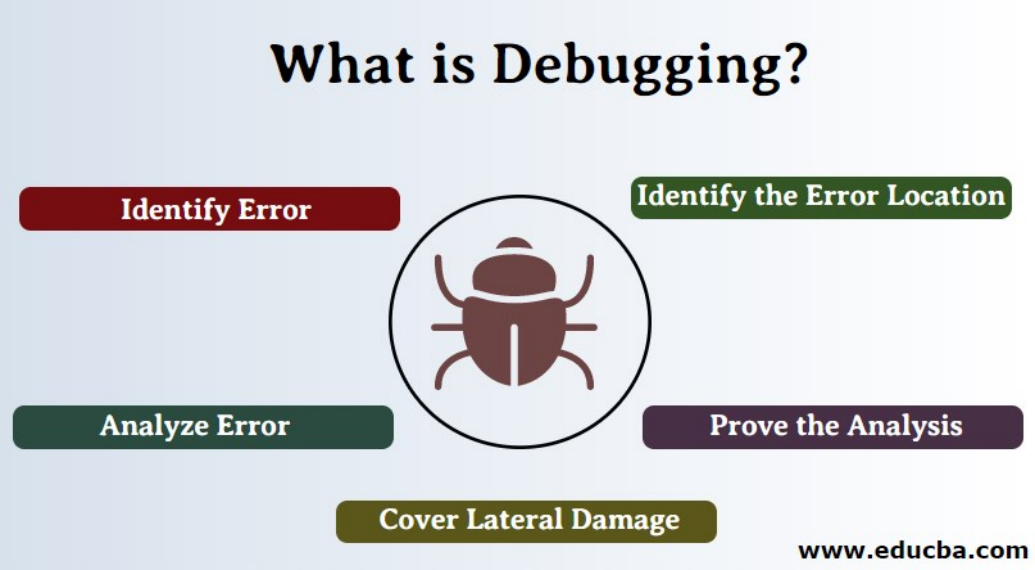
Inače izraz „bug“, tj. greška, dolazi iz incidenta 1940-ih kada se studentici na Harvardu, Grace Hopper, pokvarilo računalo zbog jednog moljca (Education.nationalgeographic.org: Sep 9, 1947 CE: World’s First Computer Bug, 2023). Dakle termin uklanjanja pogrešaka, na engleskom „debugging“, dolazi od situacije u kojoj je moljac uzrokovao štetu na računalu, a uklanjanjem tog moljca riješen je problem na računalu. Uklanjanje pogrešaka ne služi samo za poboljšavanje softvera, ono pomaže u poboljšanju korisničkog iskustva. Korisnici su frustrirani kada se susreću s greškama, stoga brzo uklanjanje pogrešaka pomaže u očuvanju zadovoljstva korisnika. Uklanjanje pogrešaka je važno jer osigurava da softver funkcionira onako kako je zamišljen i kako se očekuje te da su korisnici zadovoljni upotrebom. Pogreške koje se pojavljuju u kodu mogu biti sintaktičke pogreške, logičke pogreške, pogreške tijekom izvođenja, beskonačne petlje, pogreške u tipu itd. (Geeksforgeeks.org: What is Debugging in Software Engineering, 2024).

Proces uklanjanja pogrešaka može biti zahtjevan i naporan, a razumijevanje različitih metoda uklanjanja može učiniti proces lakšim. Postoje nekoliko standardnih metoda i tehnika koje se primjenjuju u procesu uklanjanja pogrešaka, a to su: metoda provjere koda, alati za ispravljanje pogrešaka, jedinično testiranje, testiranje integracije, testiranje sustava, praćenje i bilježenje (Geeksforgeeks.org: What is Debugging in Software Engineering, 2024). Metoda provjere koda je metoda koja uključuje ručno pregledavanje koda softvera radi identifikacije mogućih pogrešaka i nedostataka. Alati za ispravljanje grešaka su poput programa za ispravljanje pogrešaka, praćenja problema, identificiranje i rješavanje problema u kodu. Jedinično testiranje je tehnika koja uključuje testiranje pojedinačnih dijelova koda poput funkcija i komponenti kako bi se otkrile

pogreške ili propusti. Testiranje integracije je tehnika koja provjerava međusobno funkcioniranje različitih komponenti softvera kako bi se otkrili propusti u njihovoj interakciji. Testiranje sustava obuhvaća testiranje cjelokupnog softvera kako bi se otkrile pogreške koje se mogu pojaviti u integriranom okruženju. Praćenje se odnosi na kontinuirano praćenje ponašanja softvera radi identifikacije neuobičajenih pojava ili problema u performansama koje mogu ukazivati na prisutnost pogrešaka. Bilježenje događaja i poruka povezanih sa softverom omogućuje analizu i identifikaciju pogrešaka (Geeksforgeeks.org: What is Debugging in Software Engineering, 2024).

3.1 Proces uklanjanja pogrešaka

Proces uklanjanja pogrešaka sastoji se od prepoznavanje pogreške, pronalaženja mjesta pogreške, analize pogreške, dokazivanja pogreške, popravka pogreške i provjere valjanosti nakon što je pogreška uklonjena (Edcuba.com: What is Debugging?, 2023) (Slika 8).



Slika 8. Debugging proces (Edcuba.com: What is Debugging?, 2023)

Prvi korak u procesu uklanjanja pogrešaka je prepoznavanje (identifikacija) pogreške. Ovaj proces može biti dugotrajan jer zahtijeva puno vremena za prepoznavanje što je zapravo greška, a što nije. Nakon što se identificira pogreška, slijedi drugi korak koji uključuje temeljito lociranje pogreške. Pronalaženje mjesta pogreške također zahtijeva vremenski angažman, budući da je potrebno detaljno istražiti kod kako bi se pronašlo točno mjesto gdje je pogreška nastala. Stoga se prvi korak

može opisati kao proces uočavanja pogreške, dok se drugi korak odnosi na njeno pronalaženje (Edcuba.com: What is Debugging?, 2023).

Analiza pogreške, kao treći korak, uključuje sustavan pristup odozdo prema gore, počevši od mjesta gdje se pogreška pojavljuje, nakon čega slijedi detaljna analiza koda. Ovaj korak je ključan jer olakšava razumijevanje uzroka pogrešaka. Glavni ciljevi analize pogrešaka su ponovna procjena grešaka kako bi se pronašli postojeći „bugovi“ te predviđanje mogućih neizvjesnosti i kolateralnih šteta koje mogu nastati tijekom popravka (Edcuba.com: What is Debugging?, 2023).

Četvrti korak je dokazivanje pogreške gdje se mogu provjeriti i potražiti dodatne pogreške koje se mogu pojaviti u kodu. Uključuje pisanje automatiziranih testova koji će provjeriti ispravnost ispravaka pogrešaka i spriječiti njihovo ponovno pojavljivanje (Edcuba.com: What is Debugging?, 2023).

Peti korak je popravak pogreške gdje se radi izmjena koda kako bi se ispravila pogreška i time se osigurava da se pogreška ne pojavljuje ponovno (Edcuba.com: What is Debugging?, 2023).

Šesti korak je provjera valjanosti gdje se pokreću testovi kako bi se osiguralo da su sve pogreške ispravno popravljene i da softver ispravno funkcionira. Ovaj proces može se ponavljati više puta, pogotovo u kompleksnim programima gdje se pogreške mogu otkrivati i popravljati u više navrata (Edcuba.com: What is Debugging?, 2023).

3.2 Alati za uklanjanje pogrešaka

Alati za uklanjanje pogrešaka ili tzv. „debuggeri“, računalni su programi koji se koriste za testiranje i uklanjanje pogrešaka drugih programa. Ti alati koriste se sistematičnim pristupom pri dijagnosticiranju i uklanjanju pogrešaka u softveru te nude detaljne informacije o izvršenju programa, korištenju memorije, varijablama i druge relevantne podatke (Geeksforgeeks.org: What is Debugging in Software Engineering?, 2024). Funkcioniraju na način da se postavi prijelomna točka u kodu koja se obično označava retkom u kodu, zatim se aplikacija pokrene u načinu rada za ispravljanje pogrešaka, a program za ispravljanje pogrešaka presreće program i zaustavlja izvođenje. Nakon što je izvođenje zaustavljeno, provjerava se vrijednost varijabli u toj točki i zatim slijedi nadziranje programa red po red dok se ne pronađe pogreška. Neki od poznatih komercijalnih alata su GNU Debugger (GDB), Windows Debugger (WinDbg), IntelliJ IDEA Debugger, PyCharm Debugger itd.

GNU Debbuger je alat za uklanjanje pogrešaka koji daje na uvid što se događa unutar programa dok se izvršava, odnosno što je program radio kada se srušio (sourceware.org/gdb: GDB: The GNU Project Debugger, bez dat.). GNU Debbuger omogućuje pokretanje programa, zaustavljanje programa pod određenim uvjetima, ispitivanje onoga što se dogodilo u trenutku kada se program zaustavio i izmjenu programa dok je zaustavljen.

3.3 Pristupi uklanjanju pogrešaka

Postoje razni pristupi uklanjanju pogrešaka poput povratka (eng. backtracking), uklanjanje uzroka, „divide et impera“ (podijeli pa vladaj), uklanjanje pogrešaka pomoću ispisa/zapisa, uklanjanje pogrešaka gumenom patkom, automatizirano uklanjanje pogrešaka i ispravljanje pogrešaka grubom silom, tj. „brute force“ pristup (IBM.com: What is debugging? bez dat.). Treba spomenuti još dva pristupa, a to su udaljeno uklanjanje pogrešaka i post mortem uklanjanje pogrešaka (Sonarsource.com: Debugging developer's guide, bez dat.).

Povratak prati korake koji su doveli do pogreške i pomaže pri identificiranju izvora problema. Ovaj pristup funkcionira na način da se izvorni kod izvodi unatrag dok se pogreška ne otkrije. Međutim, to vraćanja unatrag može stvoriti niz drugih problema jer se vraćanjem raznolikost koraka povećava i količina samog koda postaje prevelika što bitno otežava korištenje ovog pristupa.

Uklanjanje uzroka je pristup koji zahtijeva duboko razumijevanje koda i okolnosti vezanih uz pogrešku jer se vodi nagađanjem što je uzrokovalo pogrešku i neovisno o tome da se testira svaka mogućnost kako bi se uklonio uzorak. Razvija se popis uzroka i na temelju toga se provodi testiranje kako bi se uklonila svaka pogreška.

Pristup „divide et impera“ je pristup dijeljenja koda, i koristi se pri uklanjanju pogrešaka u velikim bazama koda gdje timovi mogu podijeliti retke koda u segmente, odnosno na funkcije, module, klase ili druge logičke podjele koje se mogu testirati, a testira se svaki zasebno kako bi se locirala pogreška. Nakon što se identificira dio koji sadrži problem, može se dalje podijeliti i testirati dok se ne identificira izvor pogreške.

Uklanjanje pogrešaka pomoću ispisa/zapisa odnosi se na dodavanje naredbi ispisa ili zapisa kodu za prikaz vrijednosti varijabli, skupova poziva, tijeka izvršenja i drugih relevantnih informacija te može biti brz način za praćenje ponašanja programa i identifikacije nepravilnosti. U kod se umeću

naredbe ispisa ili zapisa (eng. print i log) kako bi se prikazalo stanje programa u određenim točkama njegovog izvođenja (Sonarsource.com: Debugging developer's guide, bez dat.).

Uklanjanje pogrešaka gumenom patkom je pristup gdje programeri objašnjavaju kod „red po red“ gumenoj patkici (ili nekom drugom neživom objektu). Ideja je objasniti kod nekom drugom, kako bi programeri bolje shvatili logiku koda i lakše uočili pogreške i nedostatke.

Automatizirano uklanjanje pogrešaka koristi se automatizacijom kako bi se ubrzao proces uklanjanja pogrešaka i smanjio ljudski angažman.

Uklanjanje pogrešaka grubom silom koristi se kada ranije navedeni pristupi nisu uspjeli, a funkcionira tako da se prolazi kroz svaki red koda u programu kako bi se identificirala pogreška. Koristi se najčešće iako je daleko od najboljeg pristupa za uklanjanje pogrešaka. Ovaj pristup nije prikladan za velike količine koda jer oduzima puno vremena.

Udaljeno uklanjanje pogrešaka je proces gdje se uklanjanje pogrešaka izvodi na sustavu različitom od mjesta na kojem se nalazi program za ispravljanje pogrešaka. Ovo se izvodi kada se pogreška može dogoditi samo na udaljenom računalu ili određenom okruženju. Kako bi pokrenuli ovaj način uklanjanja pogrešaka, potrebno je koristiti debugger koji se treba instalirati na oba računala, pri čemu se na jednom računalu izvodi program, a na drugom računalu alat za uklanjanje. Kada se udaljeno uklanjanje omogući, debugger se može koristiti na programu kao da se izvodi na istom uređaju.

Post mortem uklanjanje pogrešaka odnosi se na uklanjanje pogrešaka kada se dogodi pad programa. Koristi se mnogim metodama za praćenje poput pregledavanja bilješki, analize memorije itd.

4. Zaključak

Testiranje softvera važan je aspekt razvoja softverskog proizvoda jer pomaže pri dostavljanju pouzdanog i kvalitetnog proizvoda koji zadovoljava očekivanja korisnika. Kako bi softver funkcionirao onako kako se očekuje, potrebno je iscrpno testirati sve njegove komponente kako bi se izbjegle pojave pogrešaka. Životni ciklus testiranja osmišljen je s ciljem postizanja svih kvalitativnih ciljeva te obuhvaća šest ključnih faza koje glase: analiza zahtjeva, planiranje testiranja, razvoj testnog slučaja, postavljanje testnog okruženja, izvođenje testa i zatvaranje ciklusa ispitivanja. Različite vrste testiranja, poput automatskog i ručnog testiranja, te podvrste kao što su testiranje bijele i crne kutije, omogućuju sveobuhvatan pregled i analizu softvera. Važno je naglasiti verifikaciju i validaciju prije i nakon izvođenja koda kako bi se osigurala maksimalna kvaliteta. Različiti modeli testiranja, poput vodopada, V-modela, agilnog modela, spiralnog modela i iterativnog modela, pružaju različite pristupe i metodologije testiranja. Testiranje softvera se provodi radi otkrivanja nedostataka i pogrešaka te kako bi se oni što prije otkrili i otklonili. Kada se pogreške otkriju, nastupa debugging, tj. proces uklanjanja pogrešaka. Bitno je razlikovati testiranje softvera od uklanjanja pogrešaka jer se testiranje fokusira na identificiranje pogreške, a uklanjanje nastupa tek kada je pogreška identificirana. Sam proces uklanjanja pogrešaka može biti mukotrpan i vremenski zahtjevan. Koristeći se različitim metodama i pristupima pri uklanjanju pogrešaka daleko se olakšava cijeli proces. Navedeni pristupi u uklanjanju pogreški su povratak, uklanjanje uzroka, divide et impera, uklanjanje pogrešaka pomoću ispisa/zapisa, uklanjanje pogrešaka gumenom patkom, automatizirano uklanjanje pogrešaka, ispravljanje pogrešaka grubom silom, udaljeno uklanjanje pogrešaka i post mortem uklanjanje pogrešaka. Opisan je proces uklanjanja pogrešaka koji se dijeli na šest faza: prepoznavanje, lociranje, analiza, dokazivanje, popravak pogreške i provjera valjanosti nakon što je pogreška uklonjena. Na samom kraju objašnjeni su alati za uklanjanje pogrešaka i navedeni su neki od primjera debuggera.

5. Literatura

1. Builtin.com [Slika] (2023). *What Is the V-Model in Software Development?*. https://builtin.com/software-engineering-perspectives/v-model
2. Edcuba.com (2023). *What is Debugging?*. https://www.educba.com/what-is-debugging/
3. Education.nationalgeographic.org (2023). *Sep 9, 1947 CE: World's First Computer Bug*. https://education.nationalgeographic.org/resource/worlds-first-computer-bug/
4. Edureka.co (2020). *What are the Types of Software Testing Models?* https://www.edureka.co/blog/software-testing-models/
5. *GDB: The GNU Project Debugger* (bez dat.). https://www.sourceware.org/gdb/
6. Geek Culture (2023). *Software Development Framework — Iterative Model*. https://medium.com/geekculture/software-development-framework-iterative-model-68584bfad773
7. Geeksforgeeks.org (2023). *Software and its types*. https://www.geeksforgeeks.org/software-and-its-types/
8. Geeksforgeeks.org (2024). *Differences between Black Box Testing and White Box Testing*. https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing/
9. Geeksforgeeks.org (2024). *Differences between Verification and Validation*. https://www.geeksforgeeks.org/differences-between-verification-and-validation/
10. Geeksforgeeks.org (2024). *Manual Testing – Software Testing*. https://www.geeksforgeeks.org/software-testing-manual-testing/
11. Geeksforgeeks.org (2024). *SDLC V-Model – Software Engineering*. https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/
12. Geeksforgeeks.org (2024). *Software Characteristics – Software Engineering*. https://www.geeksforgeeks.org/software-engineering-software-characteristics/
13. Geeksforgeeks.org (2024). *Software testing life cycle*. https://www.geeksforgeeks.org/software-testing-life-cycle-stlc/
14. Geeksforgeeks.org (2024). *Types of software testing*. https://www.geeksforgeeks.org/types-software-testing/
15. Geeksforgeeks.org (2024). *What is Debugging in Software Engineering?*. https://www.geeksforgeeks.org/software-engineering-debugging/

16. Geeksforgeeks.org (2024). *What is Software Testing?* https://www.geeksforgeeks.org/software-testing-basics/
17. Graham, D., Veeneendaal, E. V., Evans, I. & Black, R. (2008). Foundations of Software Testing: ISTQB Certification. https://www.utcluj.ro/media/page_document/78/Foundations%20of%20software%20testing%20-%20ISTQB%20Certification.pdf
18. Guru99.com (2024). *STLC (Software Testing Life Cycle)*. https://www.guru99.com/software-testing-life-cycle.html
19. IBM (bez dat.). *What is debugging?* https://www.ibm.com/topics/debugging
20. ISTQB Glossary (bez dat.). Testing. https://glossary.istqb.org/en_US/term/testing-9
21. Javatpoint (bez dat.). Agile Model. https://www.javatpoint.com/software-engineering-agile-model
22. Javatpoint (bez dat.). *Verification and Validation Testing*. https://www.javatpoint.com/verification-and-validation-testing
23. Javatpoint.com [Slika] (bez dat.). *Types of Software Testing*. https://www.javatpoint.com/types-of-software-testing
24. Katalon (bez dat.). *What is Software Testing? Definition, Types, and Tools*. https://katalon.com/resources-center/blog/software-testing
25. Lakra, G. (2022). *Types of Software Testing*. https://analyticsdrift.com/types-of-software-testing/?amp=
26. MoldStud.com (2024). *Exploring the benefits of unit testing*. https://moldstud.com/articles/p-exploring-the-benefits-of-unit-testing
27. Prepinsta.com [Slika] (bez dat.). Spiral Model in SDLC. https://prepinsta.com/software-engineering/spiral-model/
28. Rosencrance, L. (2021). *Software*. https://www.techtarget.com/searchapparchitecture/definition/software
29. Sonarsource (bez dat.). *Debugging developer's guide*. https://www.sonarsource.com/learn/debugging/#why-is-debugging-important
30. Zaptest (bez dat.). *Alati za testiranje softvera – 30 najboljih proizvoda za testiranje softvera na tržištu u 2024*. https://www.zaptest.com/hr/alati-za-testiranje-softvera-30-najboljih-proizvoda-za-testiranje-softvera-na-trzistu-u-2024

31. Zaptest (bez dat.). *Nefunkcionalno testiranje: što je to, vrste, pristupi, alati i više*. https://www.zaptest.com/hr/nefunkcionalno-testiranje-sto-je-to-vrste-pristupi-alati-i-vise
32. Zaptest (bez dat.). *Što je funkcionalno testiranje? Vrste, primjeri, kontrolni popis i implementacija*. https://www.zaptest.com/hr/sto-je-funkcionalno-testiranje-vrste-primjeri-kontrolni-popis-i-implementacija
33. Zaptest (bez dat.). *Što je integracijsko testiranje? Duboko zaronite u vrste, procese i implementaciju*. https://www.zaptest.com/hr/sto-je-integracijsko-testiranje-duboko-zaronite-u-vrste-procese-i-implementaciju
34. Zaptest (bez dat.). *Testiranje bijele kutije: što je to, kako funkcionira, izazovi, metrika, alati i više*. https://www.zaptesbijelet.com/hr/testiranje-bijele-kutije-sto-je-to-kako-funkcionira-izazovi-metrika-alati-i-vise
35. Zaptest (bez dat.). *Testiranje crne kutije: što je to, vrste, procesi, pristupu, alati i više* https://www.zaptest.com/hr/testiranje-crne-kutije-sto-je-to-vrste-procesi-pristupi-alati-i-vise

Popis slika

Uklanjanje pogrešaka i testiranje softvera

Sažetak

Testiranje je proces kojim se provjerava odgovaraju li dobiveni rezultati očekivanim rezultatima te se identificiraju pogreške u programu ako postoje. Kada dođe do problema, nastupa proces uklanjanja pogrešaka koju obavlja programer kada dobije izvješće o testiranju softvera. Uklanjanje pogrešaka je proces pronalaska i rješavanja problema unutar programa koji sprječava njegov rad. Testiranje se može obavljati ručno pomoću ljudskog tima, ali i pomoću programa koji služe za testiranje određenih softvera. Uklanjanje pogrešaka obavlja se isključivo ručno. Glavna razlika između testiranja i uklanjanja pogrešaka jest to što je cilj testiranja pronaći sve pogreške, a uklanjanje pogrešaka se temelji isključivo na ispravljanju pronađenih pogrešaka.

Ključne riječi: testiranje softvera, softver, debugging, uklanjanje pogrešaka

Debugging and software testing

Summary

Testing is a process that checks whether the obtained results correspond to the expected results and identifies errors in the program if they exist. When a problem occurs, the debugging process is performed by the developer when he receives the software test report. Debugging is the process of finding and solving problems within a program that prevent its operation. Testing can be done manually by a human team, but also by using programs that are used to test certain software. Error removal is done exclusively manually. The main difference between testing and debugging is that the goal of testing is to find all errors, while debugging is based solely on correcting the errors found.

Keywords: software testing, software, debugging, removing errors