SVEUČILIŠTE U RIJECI

**TEHNIČKI FAKULTET**

Preddiplomski studij računarstva

Seminarski rad

**Defect classification and analysis**

(Klasifikacija i analiza pogrešaka)

Luka Šarlija

Luka Babić

Marin Markanjević

Borna Gilja

Kristijan Knežić

doc. dr. sc. Goran Mauša

Rijeka, svibanj, 2019

Sadržaj

[1. Uvod 4](#_Toc8683738)

[2. Ispravnost i defekti: definicije, svojstva i mjerenja 4](#_Toc8683739)

[2.1. Definicije : kvar, manjkavost/nedostatak, greška 4](#_Toc8683740)

[2.2. Svojstva i mjerenja ispravnosti 6](#_Toc8683741)

[2.3. Defekti u kontekstu održavanja i razvoja kvalitete 7](#_Toc8683742)

[3. Direktna mjerenja kvalitete: Rezultati te mjerenja defekata 7](#_Toc8683743)

[4. Ciklus procesa poboljšanja 8](#_Toc8683744)

[4.1. Identifikacija pogrešaka 8](#_Toc8683745)

[4.2. Klasifikacija pogrešaka 9](#_Toc8683746)

[4.2.1. Klasifikacija na temelju ozbiljnosti pogreške 9](#_Toc8683747)

[4.2.2. Klasifikacija na temelju vjerojatnosti 9](#_Toc8683748)

[4.2.3. Klasifikacija po prioritetu 10](#_Toc8683749)

[4.3. Analiza pogreške 10](#_Toc8683750)

[4.4. Preventivne akcije 10](#_Toc8683751)

[4.5. Unaprjeđenje proizvoda 11](#_Toc8683752)

[5. Osiguranje kvalitete 12](#_Toc8683753)

[5.1. Klasifikacija rješavanja pogrešaka 12](#_Toc8683754)

[5.2 . Rješavanje pogrešaka prije i nakon izdavanja proizvoda 13](#_Toc8683755)

[5.3 . Prevencija defekta 15](#_Toc8683756)

[5.4 . Redukcija defekta 16](#_Toc8683757)

[5.5 . Ograničavanje defekata 17](#_Toc8683758)

[6. Uspoređivanje tehnika za osiguranje kvalitete 18](#_Toc8683759)

[6.1. Usporedba prema okolini 18](#_Toc8683760)

[6.2. Usporedba prema efikasnosti 19](#_Toc8683761)

[6.3. Usporedba prema trošku 20](#_Toc8683762)

[6.4. Sažetak usporedbe i preporuka 20](#_Toc8683763)

[7. Generalni tipovi analize defekata 22](#_Toc8683764)

[7.1. Analiza raspodjele defekata 22](#_Toc8683765)

[7.1.1. Analiza smjera razvoja defekta i model kretanja defekta 24](#_Toc8683766)

[7.1.2. Analiza uzroka defekta 25](#_Toc8683767)

[8. Klasifikacija defekata i ODC 27](#_Toc8683768)

[8.1. ODC koncepti 27](#_Toc8683769)

[8.2. Klasifikacija defekata korištenjem ODC-a : primjer 28](#_Toc8683770)

[8.3. Korištenje ODC-a u analizi web grešaka 29](#_Toc8683771)

[9. Analiza defekata za klasificirane podatke 31](#_Toc8683772)

[9.1. Jednosmjerna analiza : analiziranje atributa defekta 31](#_Toc8683773)

[9.2. Dvosmjerna i višestruka analiza: ispitivanje unakrsnih interakcija 32](#_Toc8683774)

[Reference 34](#_Toc8683775)

# Uvod

Softverske pogreške mogu biti definirane kao nepravilnosti u razvoju programskog proizvoda zbog kojih proizvod ne zadovoljava dana očekivanja. Pogreške se događaju tijekom cijelog procesa razvoja, te je njihova korekcija postala bitna stavka u procesu poboljšanja kvalitete proizvoda. Svrha korekcija pogrešaka je identificiranje istih, te sprječavanje njihovog ponovnog pojavljivanja.

# Ispravnost i defekti: definicije, svojstva i mjerenja

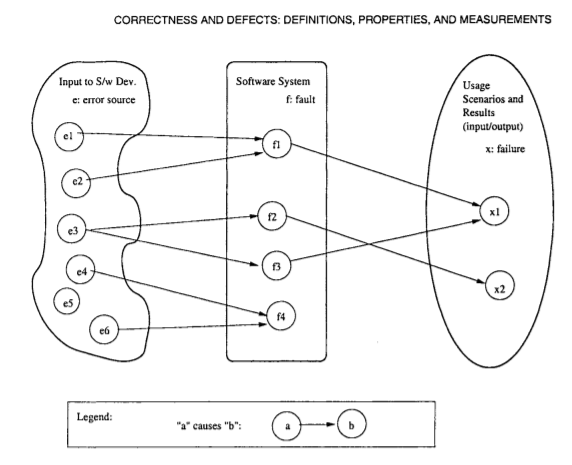
## Definicije : kvar, manjkavost/nedostatak, greška

Ljudi visoku kvalitetu software-a asociraju sa minimalnim brojem problema koje možemo očekivati prilikom rada, odnosno izvršavanja funkcionalnosti software-a. Štoviše, ukoliko do problema dođe, njihov učinak na rad software-a mora biti minimalan da bi taj software smatrali visoko kvalitetnim.

Defekt je problem sa softverom, bilo na njegovom vanjskom ponašanju ili sa unutarnjim karakteristikama.

Prema IEEE standardu 610.12 (IEEE, 1990) definiraju se termini povezani sa defektima:

* Kvar (Failure) – nemogućnost komponente ili sustava u izvođenju svojih funkcija unutar zadanih zahtjeva performansi
* Manjkavost/nedostatak (Fault) – neispavan korak, proces ili definiranje podataka u računalnom programu
* Greška (Error) – ljudska aktivnost koja dovodi do neispravnog rezultata



**Slika 1** Ilustracija povezanosti defekata

Iz slike možemo zaključiti kako postoji određena poveznica između pojmova koji se tiču aspekta defekta:

Errors → Faults → Failures

Greške (Errors) mogu uzrokovati pojavu nedostataka/manjkavosti (faults) u softveru, a te greške, promašaji opet mogu prouzročiti kvar (failure) u softveru kada je izvršen. Njihov odnos nije uvijek „jedan na jedan“ već jedna greška tipa error može uzrokovati više grešaka tipa failure npr. kada je krivi algoritam primijenjen na više različitih modula. Isto tako, više grešaka tipa faults može urokovati greške odnosno kvarove tipa failure. Ponekad nam pogreška tipa error neće uzrokovati pogrešku tipa fault (Slika 1 – Element e5), a pogreška tipa fault neće uzrokovati pogrešku tipa failure odnosno kvar (Slika 1 – Element f4). Takve greške tipa fault se nazivaju skrivene ili latentne pogreške tipa fault.

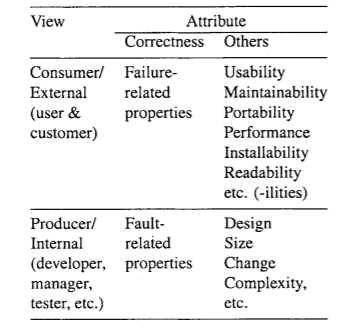
Defekt nije jednak pojmu „bug“ jer pojam „bug“ nije jasno definiran. Zato termin „debugging“ nazivamo otkrivanje i otklanjanje defekta(engl. defect detection and removal).

## Svojstva i mjerenja ispravnosti

Ukoliko ljude povezane sa software-om podijelimo binarno na korisnike i programere, možemo definirati kvalitetu te svojstva i atribute vezane uz te 2 različite perspektive (views).

Za korisnika je najbitnije da proizvod radi bez kvarova, odnosno sa što manje kvarova moguće te da utjecaj tih kvarova na funkcionalnost software-a bude minimalan te u odnosu na ta svojstva definira kvalitetu proizvoda.

Programer kvalitetu proizvoda definira kroz interakciju sa software-om s ciljem uklanjanja postojećih defekata te uklanjanja grešaka koje bi mogle prouzročiti kvarove prilikom rada programa.



**Slika 2** Pregled perspektiva (korisnik, programer) te atributa kvalitete

## Defekti u kontekstu održavanja i razvoja kvalitete

Za većinu poduzeća koja se bave razvitkom software-a, osiguranje kvalitete znači rješavanje i korekcija defekata. Postoje 3 osnovna načina za rješavanje defekata:

1. prevencija defekta
2. detekcija i uklanjanje defekta
3. obuzdavanje/ograničavanje defekta

Osim nabrojanih metoda i tehnika za rješavanje defekata, ključnu ulogu prilikom razvoja kvalitete proizvoda ima i mjerenje, analiza, testiranje te planiranje aktivnosti vezanih uz očekivane defekte da bi se stvorio multi-dimenzionalni prostor mjerenja kvalitete koji se naziva terminom “profil kvalitete”.

# Direktna mjerenja kvalitete: Rezultati te mjerenja defekata

Krucijalan element razvoja i održavanja kvalitete je analiza rezultata aktivnosti te lociranje i rješavanje problema. Naprimjer, rezultat testnog slučaja se mora provjeriti da bi se ustanovilo da li je sukladan sa očekivanjima korisnika te specifikacijom proizvoda. Ukoliko se pojavi znatno odstupanje od očekivanja možemo to odstupanje definirati kao kvar te se otvara zapis kvara pomoću kojeg ćemo ukloniti defekt.

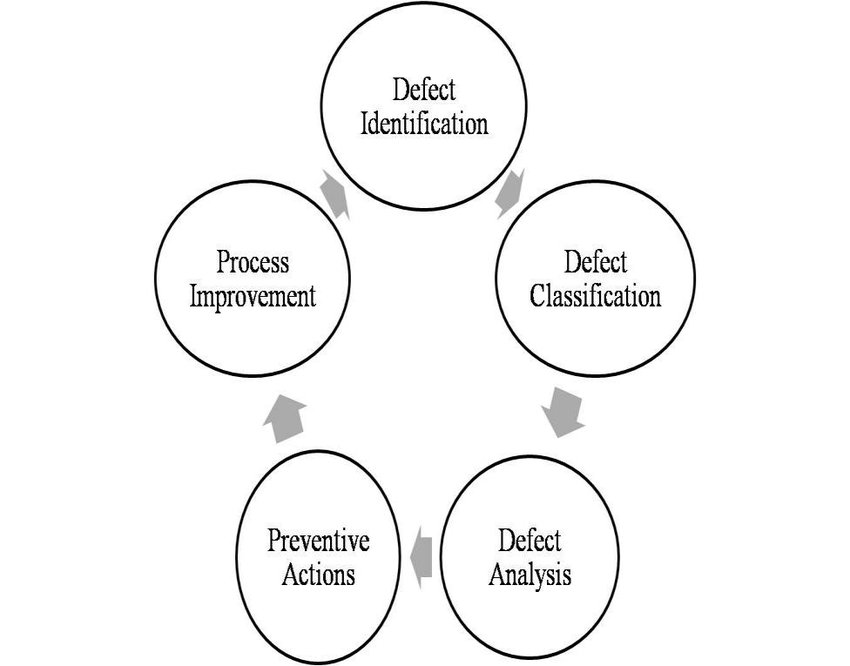
Zapis kvara mora sadržati:

* informacije o svim okolnostima prisutnim za vrijeme nastanka defekta, poput akcija koje je tester poduzimao prilikom kvara, takve informacije programerima mogu biti od velikog značaja prilikom popravka nedostatka u programu
* točnu lokaciju, odnosno područje programa u kojem je program nastao

Tokom izvršavanja metoda za razvoj kvalitete te ispravljanje pogrešaka prikupljaju se vrijedne informacije o tipu, značaju i veličini defekta te se dolazi do zaključka o tipu i načinu ispravljanja tog defekta. Sve te informacije su vrlo korisne u procesu održavanja i razvoja kvalitete.

# Ciklus procesa poboljšanja

Na Slici 1 prikazan je ciklus procesa poboljšanja programskog proizvoda. Pogreške se prvo identificiraju, te se zatim klasificiraju i analiziraju. Pronalazeći glavni uzrok, one se eliminiraju i uspostavlja se preventivni mehanizam da bi se spriječilo njihovo ponavljanje. Time se poboljšava programski proizvod.



Slika 3

## Identifikacija pogrešaka

Pogreške se pronalaze planiranim aktivnostima čiji je cilj isključivo razotkrivanje istih. Generalno, pogreške se identificiraju tijekom raznih stadija života programskog proizvoda pomoću aktivnosti kao što su *pregled dizajna, pregled GUI-a*, *pregled koda*… Jednom kada su identificirane, pogreške se klasificiraju jednom od metoda.

## Klasifikacija pogrešaka

### Klasifikacija na temelju ozbiljnosti pogreške

Ozbiljnost pogreške se definira po stupnju utjecaja iste na programski proizvod tj. koliko intenzivno će ona utjecati na proizvod. Može biti kategorizirana u jednu od četiri moguće kategorije:

1. Kritična pogreška – ove pogreške zahtijevaju neposrednu pažnju. Kritične pogreške utječu na esencijalne funkcionalnosti koje mobu rezultirati padom cijelog sustava.
2. Velika pogreška – pogreške koje utječu na osnovne funkcionalnosti proizvoda. Pogreška nije rezultirala padom cijelog sustava ali su bitne funkcionalnosti narušene.
3. Mala pogreška – pogreške koje nemaju značajan utjecaj na rad programskog proizvoda. One ne spriječavaju korisnika u izvršavanju zadataka.
4. Trivijalna pogreška – pogreške koje nemaju nikakav utjecaj na rad programskog proizvoda. Ponekad se čak i ignoriraju, kao npr. gramatičke greške

### Klasifikacija na temelju vjerojatnosti

Pogreške se klasificiraju ovisno o tome kolika je vjerojatnost da će se pogreška pojaviti te da će korisnik naići na nju.

1. Visoka vjerojatnost – postoji velika vjerojatnost da će većina korisnika doći do pogreške
2. Srednja vjerojatnost – polovica korisnika je u mogućnosti naići na pogrešku
3. Niska vjerojatnost – pogreške su detektirane kod nekoliko korisnika ili čak niti jednog

### Klasifikacija po prioritetu

Pogreške se mogu promatrati i iz poslovne perspektive. Neke pogreške treba ispraviti prije, dok se druge mogu ispraviti u kasnijim stadijima razvoja, ovisno o trenutnim potrebama i zahtjevima poslovnog sustava.

1. Visoki prioritet – označava najistaknutije potrebe sustava i pogreška se mora ispraviti što je prije moguće
2. Srednji prioritet – pogreške je moguće ispraviti u bilo kojoj slijedećoj verziji produkta
3. Nizak prioritet – pogreške se ne moraju zasebno ispravljati, već zajedno sa drugim pogreškama koje trebaju ispravak

## Analiza pogreške

Pitanja koja se postavljaju prilikom analize pogrešaka su:

Što? – identifikacija i klasifikacija

Gdje? – u kojem koraku

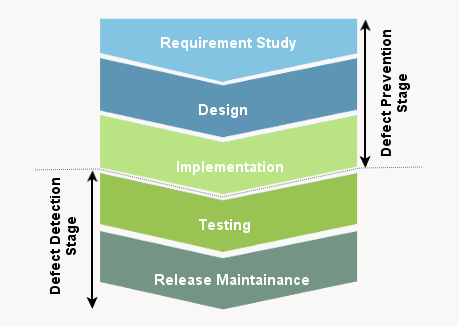
Kada? – kada se greška pojavila

Kako / Zašto?

## Preventivne akcije

Na Slici 2 se može vidjeti u kojim fazama se pogreške mogu prevenirati a u kojima detektirati. Postoje nekoliko metoda pomoću kojih se preveniraju pogreške. Neke od njih su:

* Pregled i inspekcija – ova metoda uključuje članove tima koji pregledavaju sve krajnje produkte.
* Uspoređivanje sa prototipovima
* Evidentiranje pogrešaka i dokumentacija – ova metoda pruža informacije i parametre pomoću kojih se pogreške analiziraju
* Analiza uzroka – o tome ćemo više kasnije



Slika 4

## Unaprjeđenje proizvoda

Zadnja faza u ciklusu je unaprjeđenje proizvoda. Provođenjem svih stadija u ciklusu programski proizvod postaje sve efikasnije i kvalitetniji te se tako unaprjeđuje.

# **5. Osiguranje kvalitete**

Cilj osiguranja kvalitete(*QA – quality assurance*) je osigurati da gotov programski proizvod bude dostavljen korisniku uz što je moguće manje ili potpuno bez pogreške. Štoviše, želimo osigurati da utjecaj preostalih pogrešaka bude što manji. Kroz sljedeće odlomke opisat će se nekoliko generalnih načina kako postupati s greškama, koje su rješenja te alternative. Kako osigurati prevenciju pogrešaka, te ako dođe do pogreške redukciju iste.

## **5.1. Klasifikacija rješavanja pogrešaka**

Da bi razumjeli kako se nositi sa pogreškama potrebno ih je klasificirati, odnosno podijeliti na kategorije. Podjela rada s pogreškama na kategorije omogućuje da lakše odredimo, lakše se prilagodimo te odredimo koju alternativu i tehniku otklanjanja pogreške koristiti. Sljedeća podjela pogrešaka je inicijalno predložena u Tianu 2001. godine.

* *Defect prevention through error blocking or error source removal*: ove metode služe za prevenciju unošenja određenih tipova grešaka u programski proizvod. Mogućnost nedostatka ili nepravilnih akcija čovjeka vode do unošenja pogrešaka u proizvod. Rješavaju se na način da se takve akcije blokiraju ili se otklanja uzrok tih akcija. Postoje dva načina za ispravak ovih pogrešaka:
  + Uklanjanje nejasnoća ili ispravak krivog shvaćanja odgovornog čovjeka
  + Direktno blokiranje ili ispravak neispravnih postupaka odgovornog čovjeka
* *Defect reduction through fault detection and removal*: ove metode zadužene su za detekciju i uklanjanje određenih grešaka koje su već dospjele u programski proizvod. U ovu kategoriju spadaju klasične QA aktivnosti a to su:
  + Inspekcija programskog koda, dizajna i ostalog dovodi do otkrivanja i uklanjanja greške
  + Promatranje i testiranje za vrijeme izvršavanja programa dovodi do otklanjanja grešaka

Također provjere temeljene na statičkoj analizi ili promatranje dinamičkog izvođenja mogu dovesti do smanjenja broja pogrešaka.

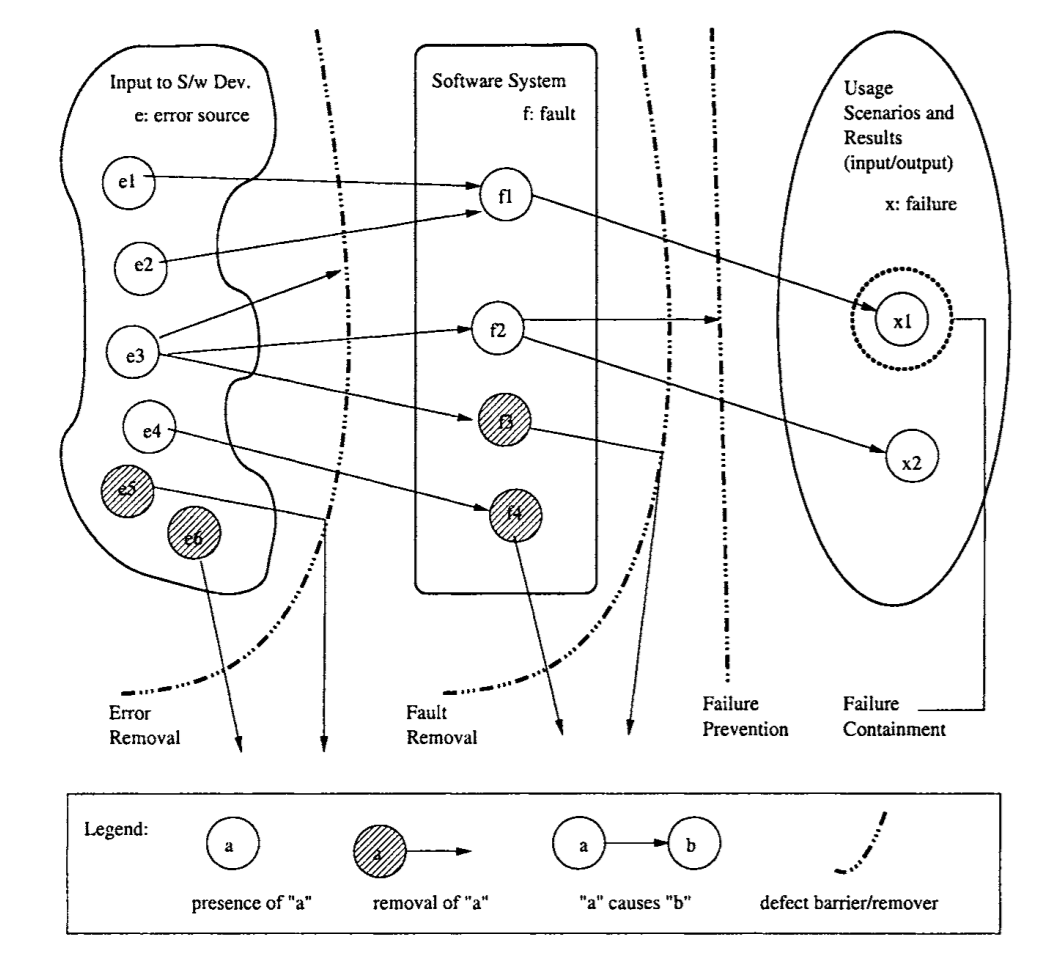
* *Defect containment through failure prevention and containment*: ove metode fokusiraju se na ograničavanje grešaka na lokalno područje tako da se ne bi pojavljivale globalne greške vidljive korisniku. Također, ograničavaju štetu uzrokovanu padom sustava. Ostvaruje se na dva načina:
  + Prekidanjem relacijskih veza tako da lokalne greške ne uzrokuju globalni kvar
  + Korištenjem mjera ograničenja kako bi se spriječile katastrofalne posljedice

## . Rješavanje pogrešaka prije i nakon izdavanja proizvoda

Aktivnosti prevencije pogreške i redukcije pogreške direktno sudjeluju u procesu pronalaska i uklanjanja grešaka za vrijeme razvojne faze proizvoda. Utječu na količinu pogrešnog sadržaja tj. broj grešaka u gotovom programskom proizvodu na način da uklone što je moguće više grešaka prije izdavanja proizvoda. Greške zaostale u gotovom proizvodu su često zvane „dormant defects[[1]](#footnote-1)“. Te greške mogu nastavit postojat određeno vrijeme, ali vrlo je moguće da će kupcu ili korisniku proizvoda zadavat probleme, odnosno situacije koje želimo izbjeći.

Nakon izdavanja proizvoda isti se promatra, bilježe se greške u radu. Kupci i korisnici također vraćaju povratne informacije o proizvodu, uključujući i greške koje je potrebno ispraviti. Ispravak pogrešaka nakon izdavanja proizvoda dovodi do boljeg proizvoda a time i kvalitetnijeg. Međutim, ne treba se oslanjati na informacije o radu nakon izdavanja proizvoda tj. popravak grešaka nakon izdavanja s ciljem podizanja kvalitete. Rješavanje grešaka nakon izdavanja proizvoda višestruko je skuplje od rješavanja grešaka za vrijeme razvoja, tj. prije izdavanja. Ovakvim postupcima moguće je ozbiljno narušiti reputacija izdavača proizvoda. Da bi se ispravak grešaka nakon izdavanja što je moguće više smanjio vrše se kontrolirana testiranja, takozvani „beta testing“.

Međutim, za smanjenje negativnog učinka neke od grešaka koriste se *defect containment activities,* aktivnosti koje ograničavaju doseg greške. Većina ovakvih metoda uključuju redundanciju i ponavljanja što zahtjeva znatno više truda za dizajn i implementaciju. Uobičajeno se koristi u situacijama gdje bi moglo biti izazvano znatno oštećenje proizvoda.



Slika 5[[2]](#footnote-2) prikazuje objašnjeno u gornjem odlomku.

- Prva isprekidana zakrivljena barijera između lijevog i srednjeg oblika predstavlja aktivnosti za prevenciju pogreške uzrokovane ljudskim faktorom

- Druga isprekidana zakrivljena barijera između srednjeg i desnog oblika predstavlja aktivnosti za uklanjanje pogreške putem inspekcije i testiranja.

- Ravna barijera između srednjeg i desnog oblika predstavlja aktivnosti za prevenciju od nemogućnosti pravilnog rada sustava

- Zadnja barijera unutar desnog oblika predstavlja aktivnost za ograničenje pogreške

## . Prevencija defekta

Metode prevencije pogreške mogu biti korištene na većini softverskih proizvoda s ciljem smanjenja mogućnosti unošenja pogreške tj. smanjenje troškova koji slijede ukoliko bi pogreška bila unesena u proizvod. Većina metoda pretpostavlja da je do pogreške došlo ljudskim faktorom tj. da je uzrok neobavljena ili netočna akcija čovjek. Neki od uzroka pogreške te prevencija:

* Ako je čovjekovo pogrešno shvaćanje uzrok pogreške, prevencija edukacijom i obukom čovjeka mogu riješiti problem
* Neprecizno definiran dizajn ili neispravna specifikacija proizvoda mogu dovesti do unošenja greške u sustav. Prevencija se vrši formalnim metodama (*eng. formal method*)
* Ako ne korištenje standarda dovodi do unošenja greške prevencija je korištenje istih
* Ako neki alat ili tehnologija smanjuju mogućnost pojave greške, prevencija je korištenje istih

**Edukacija i obuka** omogućuju ljudska rješenja za eliminaciju grešaka. Promatranjem je zaključeno da je ljudski faktor ključan te određuje kvalitetu tj. uspjeh ili neuspjeh projekta. Edukacija i obuka pojedinca pomoći će kontrolirati, upravljati i unaprijediti njihov rad. Eliminacija pogrešnog shvaćanja čovjeka pomoći će da prilikom izrade softvera ne dolazi do unošenja kvarova u proizvod.

**Formalne metode** (*eng. formal method)* uključuju formalnu specifikaciju i formalnu verifikaciju. Formalna specifikacija zadužena je za izradu smislene specifikacije proizvoda koji kupac zahtjeva. Zadaća formalne verifikacije je provjeriti slažu li se dizajn i implementacija s zadanom formalnom specifikacijom. Uz to provjerava da li je proizvod bez nekih nedostataka s obzirom što je definirano u formalnoj specifikaciji.

Druge tehnike prevencije pogreške, uglavnom su bazirane na tehnologiji, alatima, standardima. Korištenje odgovarajućih softverskih metoda i tehnologija pomaže smanjiti šanse unošenja nedostatka. Preporučeno je koristiti ispitane softverske tehnologije više kvalitete. Bolje upravljanje procesima može doprinijeti kvaliteti proizvoda. Potrebna je konzistencija u upravljanju kroz različite komponente. Korištenje odgovarajućih softverskih alata također pomaže kod prevencija pogreške. Na primjer alati koji automatski zatvaraju zagrade kod pisanja programskog koda.

## . Redukcija defekta

Ne možemo očekivati ni od jedne metode prevencije pogreške da će biti potpuno uspješna. Zbog toga postoje metode redukcije pogreške kojima je zadatak ukloniti što je više moguće greška koje su unesene.

**Inspekcija** je pregledavanje softverskog proizvoda od strane čovjeka s ciljem otkrivanja i popravka nedostataka koje sadrži. Inspekcija se vrši pod različitim okolnostima i u različitim okruženjima. Osnovne ideje inspekcije su čitanje i analiza programskog koda, te ostalih dokumentacija kao što su specifikacija proizvoda, plan testiranja i ostalo. Inspekcije se provode od strane više koordiniranih ljudi zaduženih za provjeru. Pronađeni nedostaci moraju biti uklonjeni te uklanjanje mora biti potvrđeno. Proces inspekcije je planiran.  
Iako se najčešće pregledava programski kod, vrlo dobra praksa je inspekciju vršiti i na ostalim dokumentima. Inspekcija se može provoditi još u razvojnoj fazi dok testiranje nije moguće.

**Testiranje** je jedna od najvažnijih i najčešće primjenjivanih metoda redukcija pogrešaka. Uključuje izvođenje programa te promatranja rada i izlaza programa. Ako je otkriven kvar, izvođenje programa je analizirano kako bi se pronašao i riješio nedostatak koji je izazvao kvar.   
Priprema za testiranje vrši se na način da se provjeravaju implementirane komponente, sustav na kojem će se program izvoditi. Testiranje komponenti sustava može početi već u ranijem stadiju razvijanja. Testiranje možemo podijeliti u fazu testiranja programskog koda te fazu testiranja nakon izdavanja proizvoda.

Postoje dvije vrste testiranja proizvoda. Funkcionalno testiranje koje provjerava ispravnost funkcija softvera, ponaša li se proizvod u skladu s očekivanjima korisnika i specifikacijom proizvoda. Prilikom testiranja kvarovi povezani s funkcijama programa se pregledavaju, otkrivaju se nedostaci te se uklanjaju. Cilj mu je smanjiti greške kako bi korisnici imali manje problema s funkcijom proizvoda. Strukturalno testiranje provjerava ispravnost implementacije građevnih cjelina, struktura i veza između njih. Kvarovi vezani na unutarnju implementaciju se razmatraju, pronalaze i uklanjaju. Cilj je redukcija unutarnjih grešaka koje bi vodile do kvarova u budućnosti.

Testiranje prekidamo kada smo sigurni da smo provjerili sve scenarije do kojih bi u normalnom radu korisnik mogao doći. Koriste se *check*-liste kako bi bili sigurni da smo testirali sve glavne funkcije i scenarije proizvoda prije izdavanja istog.

Također postoji još metoda kako bi reducirali pogreške. Neki do njih su analiza algoritama, analiza ograničenja vrijednosti, *finite-state machine*, *control and data flowI* analize. Kao dodatak na testiranje s ciljem pronalaska i uklanjanja pogreške postoje simulacije i prototipovi proizvoda.

## . Ograničavanje defekata

Zbog veličine i visoke složenosti današnjih softverskih proizvoda, ograničavanje defekata (pogrešaka) može umanjiti broj istih ali ne ih i potpuno ukloniti. Za određene softverske sustave gdje je utjecaj kvara kritičan, kao na primjer u medicini ili nuklearnoj industriji, ograničavanje defekata nije dovoljno te je potrebno koristiti dodatne alternative. Dok u drugu ruku, tamo gdje je manja mogućnost kvara i gdje on nije kritičan ograničavanje ima smisla koristiti. Koristi se na način da razbija relacije između dijelova sustava, to znači da će *tolerirati* određene nedostatke sustava ali će njihovim ograničavanjem spriječiti kvar koji vuku za sobom.

# **6. Uspoređivanje tehnika za osiguranje kvalitete**

Jedna od zadaća tehnika za osiguranje kvalitete je utvrđivanje i rješavanje kvarova u samom programu. Njih uspoređujemo prema cijeni, mogućnosti korištenja u raznim okolinama i fazama razvoja u kojima se mogu koristiti te prema efikasnosti rješavanja određenog problema.

Neke od tehnika za osiguranje kvalitete su:

* *testiranje*
* *prevencija kvarova*
* *inspekcija*
* *formalna verifikacija*
* *otpornost na kvarove*
* *zadržavanje kvarova*

|  |  |
| --- | --- |
| **QA tehnika** | **Objekt nad kojim se provodi** |
| Testiranje | Programski kod prilikom izvršavanja |
| Prevencija grešaka | Prilikom implementacije |
| Inspekcija | Dizajn, kod i sl. |
| Formalna verifikacija | Dizajn/kod s formalnom verifikacijom |
| Otpornost na kvarove | Operativni programski sustav |
| Zadržavanje kvara | Dio sustava s potencijalnim kvarovima |

## 6.1. Usporedba prema okolini

Podjela prema okolini se dijeli na:

* *okolinu razvoja*
* *okolinu održavanja*

Većina tehnika za održavanje kvalitete proizvoda je orijentirana razvojnoj okolini, ali neke od njih se jednako dobro mogu koristiti i u okolini održavanja. Tehnike se uspoređuju prema primjeni u korektivnoj okolini održavanja. Također postoje i adaptivna te perfektivna, ali primjena tehnike u tim okolinama je slična onoj u razvojnoj okolini.

|  |  |
| --- | --- |
| **QA tehnika** | **Faza korištenja** |
| Testiranje | Faza testiranja i sve ono nakon nje |
| Prevencija grešaka | Implementacija |
| Inspekcija | Uvijek |
| Formalna verifikacija | Dizajn/kodiranje |
| Otpornost na kvarove | in-field operation |
| Zadržavanje kvara | in-field operation |

Također QA tehnike se uspoređuju i prema njihovoj složenosti, tj. koliki stupanj edukacije osoblje mora posjedovati da bi mogli ispravno i efikasno koristiti određenu QA tehniku. Ukoliko je velika razlika između kvalifikacije osoblja te težini određene tehnike može doći do dodatnih problema te i samim time posao neće biti izveden na pravi način. To je dobar način za izračunavanje direktnog troška nekog kvara.

|  |  |  |
| --- | --- | --- |
| **QA tehnika** | **Razina stručnosti osoblja** | **Edukacija** |
| Testiranje | slaba – visoka | / |
| Prevencija grešaka | Srednja – visoka | / |
| Inspekcija | Slaba – srednja | / |
| Formalna verifikacija | Visoka | Formalni trening |
| Otpornost na kvarove | Visoka | Dinamički sustavi |
| Zadržavanje kvara | Visoka | Sigurnost, ugradbeni sustavi |

## 6.2. Usporedba prema efikasnosti

Drugačije tehnike za održavanje kvalitete pristupaju rješavanju problema kroz drugačije perspektive, bave se drugačijim problemima te se mogu koristiti prilikom drugačijih strategija za održavanje kvalitete proizvoda. Kako je svaka tehnika najefikasnija za pojedini slučaj, inače ne bi bilo smisla njezinog postojanja, efikasnost te tehnike se uspoređuje na generalnoj razini.

Zbog navedenih razloga tehnike uspoređujemo prema:

* *perspektivi,* koja se dijeli prema tome kako koja tehnika promatra i detektira kvar te tipu popratnih akcija
* *tipu problema*, neki od njih su: errors, faults, failures
* *razini kvara*
* *konstruktivnim informacijama i mjernicama za osiguranje kvalitete*

## 6.3. Usporedba prema trošku

Umjesto prema direktnom trošku, QA tehnike uspoređujemo:

* *jednostavnosti tehnike*
* *mogućnost korištenja alata*

Većina QA tehnika, kao što su testiranje i inspekcija, se orijentiraju na detekciju i uklanjanje kvarova. Trošak tada može biti izravno povezan s naporom u otkrivanju i rješavanju problema. Drugi dio troškova koji se ne smatra uobičajenim je trošak štete nastale uslijed neispravnih grešaka, to su greške koje nisu uočene od strane QA tehnika nego postaju vidljive tek u nekom specifičnom slučaju (npr. kršenje rubnog uvjeta). Međutim, ovaj trošak je negativno povezan s učinkovitošću različitih QA tehnika, a posebno o razinama kvalitete na izlazu nakon primjene kolekcije QA tehnika. Stoga se prilikom usporedbe različitih QA tehnika fokusiramo na troškove otkrivanja i uklanjanja kvarova. Generalno, što se duže kvar nalazi u sustavu, to će ukupni trošak biti veći (linearan rast).

|  |  |
| --- | --- |
| **QA tehnika** | **Trošak (cijena)** |
| Testiranje | Srednji |
| Prevencija grešaka | Niski |
| Inspekcija | Niski – srednji |
| Formalna verifikacija | Visoki |
| Otpornost na kvarove | Visoki |
| Zadržavanje kvara | Najviši |

## 6.4. Sažetak usporedbe i preporuka

Prema prethodno predstavljenim usporedbama dolazimo do zaključka:

* Da bi se osigurala kvaliteta proizvoda te da bi se kvarovi popravili ispravno i učinkovito, potrebno je koristiti više različitih QA tehnika.
* Tehnika prevencije kvarova poprilično smanjuje pojavu ubačenih kvarova (*fault injections*) te bi se trebala u svakom planu osiguranja kvalitete.
* Tehnike kao što su inspekcija i testiranje imaju mogućnost korištenja u različitim situacijama te su učinkovite za različite razine i tipove problema. Stoga se može prvo koristiti inspekcija koja će smanjit razinu grešaka tako što je direktno otkriti i ukloniti lokalne i statičke pogreške u sustavu, a tek onda QA tehniku testiranja koja će ukloniti preostale greške u sustavu vezane za dinamičke scenarije i interakciju.
* Tehnike kao što su otpornost na kvarove te formalna verifikacija imaju veliki trošak implementacije te se mogu koristiti kada je potencijalno visok trošak kvara.

|  |  |  |  |
| --- | --- | --- | --- |
| **QA tehnika** | **Primjena** | **Učinkovitost** | **Trošak** |
| Testiranje | Programski kod | Occasional failures | Srednji |
| Prevencija grešaka | Poznati izvori | Systematic problems | Niski |
| Inspekcija | / | Scattered faults | Niski – srednji |
| Formalna verifikacija | Formalna spec. | Fault absence | Visoki |
| Otpornost na kvarove | Duplikacija | Rare-cond. Failures | Visoki |
| Zadržavanje kvara | Poznate opasnosti | Rare-cond. accidents | Najviši |

# 7. Generalni tipovi analize defekata

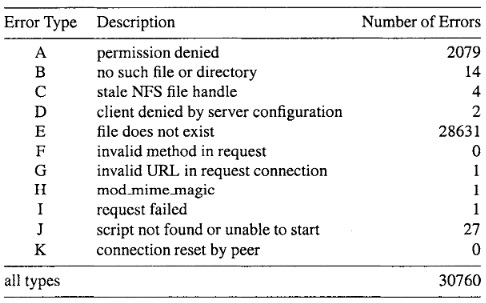
Kada je defekt otkriven može se izvesti mnogo pojedinačnih analiza. Jednom kada su prikupljeni podaci o defektu može se primijeniti kolektivna analiza. Iako ova dva tipa analize imaju različita gledišta oba postavljaju neka od pitanja, a to su:

* Što? – Određivanje i klasifikacija otkrivenih defekata – identificiranje što su ti defekti i stavljanje istih u razrede po određenoj konzistentnoj shemi.
* Gdje ? – Informacija gdje je otkriven defekt koja može biti iskorištena kao vrijedna povratna informacija u procesu razvoja kroz analizu raspodjele defekata.
* Kada? – Određivanje točnog vremena ili pripadajuće faze, podfaze razvoja kada je defekt umetnut i kada je otkriven što je važno jer pruža informaciju za analizu kretanja defekata i služi kao baza za predviđanje kvalitete u budućnosti.
* Preddistribucijski ili postdistribucijski defekt ? – Defekti unutar procesa razvoja koji se otkriju prije ili nakon korištenja softvera od strane potrošača, krajnjeg korisnika.
* Kako i zašto? – Kako se defekt dogodio i koji je uzrok tomu.

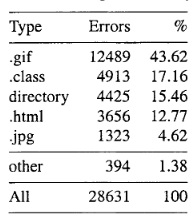
## 7.1. Analiza raspodjele defekata

Analiza raspodjele defekta pomaže nam odgovoriti na ključna pitanja što i gdje. Kod odgovaranja na pitanje „što“ možemo otkriti o kojem tipu defekta je riječ i kako je povezan sa drugim podskupinama u cjelovitom skupu defekata. Nakon što smo otkrili kojoj skupini defekt pripada možemo lakše odabrati postupke za bolje rješavanje problema. Kao konkretni primjer možemo navesti web aplikacije gdje su defekti pogreške tipa failure nazvane još i web pogreške koje se spremaju u dnevniku grešaka web servera. Ako pogledamo raspodjelu pogrešaka od web stranice [www.seas.smu.edu](http://www.seas.smu.edu) iz primjera navedenog u literaturi[[3]](#footnote-3), uočavamo neravnomjernu raspodjelu web grešaka odnosno defekata kroz različite tipove grešaka. Na slici 6 vidimo kako je najzastupljenija greška tipa E(„file does not exist“) u visokom postotku od 93.08% te su istraživanja bila usmjerena na tu grešku kako bi poboljšala rad web stranice.

Kod odgovaranja na pitanje „gdje“ možemo otkriti u koju skupinu defekt pripada preko različitih područja ili dijelova proizvoda otkriti je li povezan sa nekom od cjelokupnih podjela defekta. Na taj način se možemo lakše fokusirati na određeni dio gdje se javljaju defekti te tako poboljšati proizvod. Za primjer uzmimo već spomenuti primjer web stranice [www.seas.smu.edu](http://www.seas.smu.edu). Ako postavljamo pitanje „gdje“ možemo se pitati koji tipovi datoteka nedostaju. Od prikaza datoteka koje nedostaju na slici 7 vidimo da od svih mogućih datoteka one koje nedostaju se nalaze u top 5 u zastupljenosti od 98%. Identifikacijom tipova datoteka koje nedostaju te povezanosti sa njihovim pristupnim informacijama možemo otkloniti nastali problem i povećati pouzdanost web stranice.



Slika 6 - Tipovi i podjela pogrešaka za stranici SMU-SEAS[[4]](#footnote-4)



Slika 7 - Karakteristične web greške po tipu datoteke[[5]](#footnote-5)

Kod analize podjela defekta obično se susrećemo sa pogreškom tipa fault ili tzv. popravljenim defektima. Popravljeni defekti se javljaju kada popravljanje otkrivenog problema dolazi prije analize defekta, dok kod pogreške tipa fault može doći do popravljanja problema tek nakon što smo identificirali defekt.

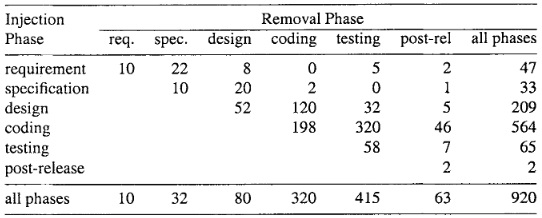
Problem kod ove vrste analize je što ne možemo uvijek čekati dok se defekti otkriju i dok neravnomjerna podjela defekata potvrdi poduzimanje određenih aktivnosti. Umjesto toga možemo pronaći način za pronalazak takvih tipova defekata za određeno područje temeljen na podacima iz prošlosti.

### 7.1.1. Analiza smjera razvoja defekta i model kretanja defekta

Većina podataka o defektima sadrži određene informacije o vremenu ili je minimalno defekt svrstan u preddistribucijski ili postdistribucijski. Kada se takva informacija koristi sa određenim modelima, ti nam podaci mogu poslužiti kao temeljna postavka za predviđanje u budućnosti.

Ponekad informacija o vremenu za pojedine defekte odgovara informacijama o pojedinim fazama ili podfazama razvoja u nekim zapisima o defektima.

Ako je informacija o unosu defekta dostupna, to nam može poslužiti kao argument za stvaranje modela kretanja defekata gdje su unos i otklanjanje defekata praćeni fazama razvoja. Ovaj model je često predstavljen kao matrica sa recima koji odgovaraju unosu defekta u svakoj fazi i stupcima koji odgovaraju uklanjanju objekata u svakoj fazi što je prikazano slikom 8. Unutarnja matrica je gornje trokutasta jer se otklanjanje defekta uvijek odvija nakon unosa defekta. Zadnji redak sumira sve defekte otklonjene u različitim fazama te nam daje model otklanjanja defekta. Zadnji stupac sumira sve defekte umetnute u različitim fazama te nam daje informaciju gdje su glavni izvori defekata u uvjetima kada su umetnuti.



Slika 8 - Uzorak dinamičkog modela defekta[[6]](#footnote-6)

Logično je za pretpostaviti kako svaki defekt može stvoriti određenu štetu, trošak zbog čega je nužno otkriti defekte što ranije i ispraviti ih na vrijeme.

Trošak defekta umetnutog u fazi X i maknutog u fazi Y nije isti jer se značajno povećava sa udaljenošću između X i Y odnosno brojem faza u kojem defekt ostaje neotkriven.[[7]](#footnote-7)

### 7.1.2. Analiza uzroka defekta

Analiza uzroka defekta može imati dva oblika: logička analiza i statistička analiza. Logička analiza je deterministička analiza koja ispituje logičku poveznicu između efekta i određenih uzroka i uspostavlja generalnu uzročnu vezu. Statistička analiza je probabilistička analiza koja ispituje statističku vezu između uzroka i efekta i daje zaključak o vjerojatnoj uzročnoj vezi između njih. Efekti su u ovoj vrsti analize ili greške tipa failures ili otkrivene(ili ispravljene) greške tipa faults, a odgovarajući uzroci su greške tipa faults koja uzrokuje grešku tipa failures ili greške tipa errors koje uzrokuju umetanje greške tipa faults. Uzročna veza između greške tipa fault i greške tipa failure određena je od strane developera ili vlasnika koda koji popravlja kod ili dizajn kao odgovor na grešku tipa failure koja se javila tijekom testiranja ili inspekcije kao dio normalno procesa razvijanja gdje se popravljaju defekti. Uzročna veza između greške tipa error i greške tipa fault određene su kroz posebnu analizu uzorka defekta iznad normalnog procesa razvoja te se ovakva vrsta u literaturi naziva korijenska analiza uzroka. Korijenska analiza uzroka provodi se od strane čovjeka sa temeljitim znanjem o proizvodu, procesu razvoja, domeni primjene i generalnom okruženju. Jedan od primjera ovakve analize je proces brainstorming.

Statistička analiza se bazira na empirijskim dokazima skupljenim lokalno ili u sličnim projektima. Ovi empirijski podaci se mogu obogatiti različitim modelima kako bi ostvarili predvidljive odnose između efekata i uzroka. Jednom kada je uzročni odnos uspostavljen određene aktivnosti za osiguranje kvalitete se mogu primijeniti za otklanjanje greški tipa fault ili tipa error.

# 8. Klasifikacija defekata i ODC

Kod susretanja sa problemima tijekom korištenja ili razvoja softvera mogu se skupiti i pohraniti različite informacije o defektima. Takve se informacije mogu sustavno organizirati za daljnje analize, a rezultati analize obećavaju vrjedniju i specifičniju povratnu informaciju koja ukazuje na specifična problematična područja za fokusirano rješavanje problema i poboljšanje kvalitete. Takve analize obično koriste statističke modele. Obično se uzročna analiza za pojedinačne defekte i statistička analiza za skupinu podataka o defektima koriste odvojeno jedna od druge . Sustavna klasifikacija i analiza podataka o greškama premošćuje jaz između uzročne analize i statističke analize te osiguravaju dragocjenu povratnu informaciju u procesu razvoja ili održavanja te pomažu osigurati i poboljšati kvalitetu proizvoda. Ortogonalna podjela defekata(ODC), prvotno razvijena u IBM-u, je najutjecajniji među općim okvirima za klasifikaciju i analizu defekta softvera.

## 8.1. ODC koncepti

ODC ima bogatu i opsežnu kategoriju atributa defekata, koji proizlaze iz pogleda na greške tipa failure i pogleda na greške tipa fault. Atribute povezane s prvim tipično ispunjavaju ispitivači softvera ili inspektori koji su u početku uočili probleme i otvorili izvješća o greškama, dok su oni povezani s potonjim tipično popunjeni od strane razvojnih inženjera ili osoblja za održavanje sustava koji su popravili prijavljene probleme i ažurirali odgovarajuće izvještaje o defektima. Atributi defekta organizirani su po sljedećoj hijerarhiji kako je navedeno u literaturi[[8]](#footnote-8):

* Ključni atributi defekta iz pogleda greške tipa failure i informacija sakupljenih kod otkrivanja defekta uključuju:
* Učinak defekta, s vrijednostima atributa koji pokrivaju funkcionalnost, pouzdanost itd.
* Okidač defekta s vrijednostima atributa koje odgovaraju određenim vrstama testiranja
* Ozbiljnost defekta, s uobičajenim vrijednostima atributa: kritična, velika, manja ili neka numerička skala
* Ključne informacije iz prikaza greške tipa fault prikupljene pri popravljanju defekata uključuju:
  + Tip defekta s vrijednostima atributa: funkcija, sučelje, algoritam, mjerenje vremena itd.
  + Broj redaka iskorišten za ispravljanje
* Neke dodatne uzročne analize se mogu provesti, a povezani rezultati pružaju informacije o nedostacima i povezane atribute, kao što su:
  + Izvor defekta, s vrijednostima atributa: kod dobavljača, novi kod, osnovni kod itd.
  + Gdje je unesen defekt, smješten u podsustave, module ili komponente.
  + Kada je unesen defekt, obično se identificira s razvojnom fazom.

## 8.2. Klasifikacija defekata korištenjem ODC-a : primjer[[9]](#footnote-9)

U ranijoj studiji o klasifikaciji i analizi defekata za neke IBM-ove proizvode relacijskih baza podataka, u fazi testiranja sustava prikupljeni su različiti podaci o defektima prema ODC-u. U ovom okruženju, nakon što se otkrije defekt, formalno izvješće (nazvano Izvješće o praćenju problema ili PTR u IBM-u) se bilježi i prati do konačne odluke. Različiti alati koji su se izvorno koristili za praćenje izvješća o defektima prošireni su kako bi se prikupile dodatne informacije o kvaru povezane s ODC-om.

Informacije koje pružaju ispitivači o otkrivanju defekata uključuju utjecaj defekta, okidač, ozbiljnost i vrijeme otkrivanja defekta izraženo u tjednima:

* Utjecaj defekta temelji se na odgovorima na pitanje: "Ako ovaj defekt nije ispravljen, kako će to utjecati na kupca?". Prethodno definirane kategorije utjecaja (mogući odgovori) uključuju izvedbu, pouzdanost itd.
* Kategorije okidača defekta vrlo su slične razredima testnog scenarija koji se koriste za upravljanje procesom testiranja za ovaj proizvod.
* Ozbiljnost defekta može biti 1 (kritični problem), 2 (glavni problem), 3 (manji problem) i 4 (manje poteškoće, neugodnosti).
* Tjedan u kojem je kvar otkriven, računa se od početka projekta.

Informacije prikupljene pri popravljanju defekata odnose se na radnje koje su programeri poduzeli kako bi pronašli, identificirali i ispravili greške tipa fault koje su uzrokovale otkrivene pogreške tipa failure:

* Tip popravka: popravak dizajna, koda, itd.
* Broj linija koda koji je promijenjen prilikom popravljanja.
* Radnja ispravljanja: dodavanje, brisanje ili promjena dizajna ili koda.

Neke jednostavne uzročne analize izvršili su programeri kada su ispravili prijavljene defekte, što je dovelo do sljedećih rezultata uzročne analize zabilježenih u podacima ODC-a:

* Izvor kvara: kod dobavljača, novi kod, osnovni kod itd.
* Faza razvoja kada je defekt umetnut: prethodna izdanja ili faze razvoja slične slapovima u trenutnom izdanju.

## 8.3. Korištenje ODC-a u analizi web grešaka

Za web aplikacije, može se definirati ODC-slična klasifikacija grešaka i relevantni podaci o defektima mogu se izdvojiti iz postojećih dnevnika web poslužitelja za analizu. Dostupnost takvih informacija u web zapisima je značajna prednost za ovu situaciju u odnosu na tradicionalne primjene ODC-a, gdje je prikupljanje podataka uvijek velika prepreka koja zahtijeva od programera i testera da posvete značajno vrijeme za analizu defekata i izvješćivanje o nalazima. Među različitim atributima ODC-a, slijedeće se može prilagoditi za analizu web problema:

* Učinak defekta odgovara vrsti web pogreške, što ukazuje na problem koji su iskusili web korisnici. Može se analizirati izravno na temelju informacija koje su izvađene iz zapisnika o pogreškama ili iz koda koji se koristi u dnevnicima web pristupa.
* Pokretač defekta odgovara specifičnim sljedovima upotrebe ili uputa koje dovode do problema zabilježenih u zapisnicima pogrešaka. Može se analizirati ispitivanjem informacija o paru uputa koje se mogu izdvojiti iz dnevnika pristupa.
* Izvor defekta odgovara određenim datotekama ili vrstama datoteka koje je potrebno izmijeniti, dodati ili ukloniti da bi se riješili problemi zabilježeni u zapisnicima pogrešaka. Može se analizirati ispitivanjem specifičnih pogrešaka i parova uputa.

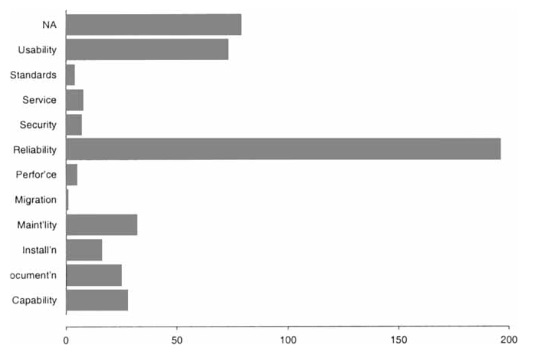
Različiti drugi atributi također se mogu usvojiti ili prilagoditi od izvornih ODC atributa kroz pomno ispitivanje web okruženja i dostupnosti podataka. Takvo prilagođavanje različitim okruženjima može pomoći ljudima kod analiziranja problema ili pitanja koja ih zanimaju i ispunjava različite namjene. Kao dodatni primjer, ODC je prilagođen kako bi pomogao u trajnom otkrivanju zahtjeva u sustavima visokog integriteta.

# 9. Analiza defekata za klasificirane podatke

Za analizu klasificiranih podataka mogu se primijeniti različite tehnike. Najočitije i najizravnije analize su primjena podjele defekta i analiza smjera razvoja defekta za neklasificirane podatke o nedostacima izravno na klasificiranim podacima. U ODC terminologiji, to se naziva jednosmjernom analizom, jer ispituje jedan atribut u isto vrijeme, ili njegovu ukupnu distribuciju ili njegov smjer razvoja(trend) tijekom vremena. Dvosmjerna analiza može se koristiti za ispitivanje unakrsne interakcije dvaju atributa. Moguća je i analiza višeg reda, kao što je modeliranje temeljeno na stablu na svim ODC atributima. Jedna od temeljnih pretpostavki u svim tim analizama je da postoji očekivani profil defekta. Stvarni rezultati analize se uspoređuju s tim profilom, rangirani prema njihovim razlikama kako bi se identificirale anomalije ili one s najvećim razlikama. Navedene anomalije analizira osoblje za razvoj kako bi se vidjelo jesu li očekivane, ako ne, potrebno je pokrenuti neke korektivne radnje kako bi se riješili problemi. Međutim, u mnogim aplikacijskim okruženjima takav profil ne postoji. Ono što se predlaže je da se kao početna točka koristi ravnomjerna podjela i postupno se gradi takav profil defekta za buduću uporabu.

## 9.1. Jednosmjerna analiza : analiziranje atributa defekta

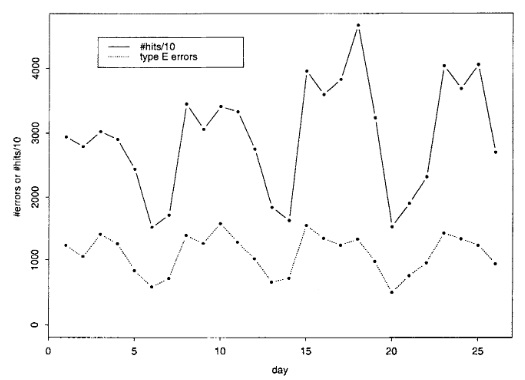
Za svaki atribut defekta može se ispitati ukupna distribucija njegovih vrijednosti. Na primjer, na slici 9 prikazana je distribucija atributa defekta za IBM proizvod. Analiza pokazuje da je raspodjela defekata među različitim područjima oštećenja vrlo nehomogena. Očekivano, najveći udio defekata jesu defekti u pouzdanosti. Glavni razlog je to što se testiranje sustava uglavnom bavi ukupnim radom i robusnošću proizvoda.



Slika 9 - Jednosmjerna analiza utjecaja defekta za IBM-ov proizvod[[10]](#footnote-10)

## 9.2. Dvosmjerna i višestruka analiza: ispitivanje unakrsnih interakcija

Dvosmjerna analiza ispituje interakciju između dva atributa i može se primijeniti na sve atribute u paru. Najjednostavniji oblik dvosmjerne analize je uvjetna analiza pojedinačnog atributa pod uvjetom da neki drugi atribut uzima određenu vrijednost. Na primjer, u gornjem primjeru analize web-pogrešaka za web-mjesto SMU/SEAS, slika 10 daje distribuciju pogrešaka tipa E prema vrsti datoteke. To se može tumačiti kao dvosmjerna analiza za dva atributa: vrsta pogreške (tip E kao uvjet ovdje) i vrsta datoteke koja nedostaje.



Slika 10 - Pogreška (tip E) i pogodak profila za SMU/SEAS[[11]](#footnote-11)

Nakon općeg napredovanja od jednosmjerne analize do dvosmjerne analize, možemo imati opću višestruku analizu. Međutim, takvi rezultati analize su previše kompleksni za praćenje zbog kombinatorne eksplozije višestrukih atributa, zbog čega je gotovo nemoguće interpretirati i koristiti rezultate u praktične svrhe.

# Reference

[1] Jeff Tian: Software quality engineering (testingquality assurance and quantifiable improvement).

[2] <http://www.professionalqa.com/types-of-defects-in-software-testing>

[3] <https://melsatar.blog/2018/06/26/the-software-process-improvement-spi-reward-or-risk/>

[4] <https://www.softwaretestinghelp.com/defect-prevention-methods/>

[5]<https://s3.amazonaws.com/academia.edu.documents/33650399/IJCA_Publication.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1556839235&Signature=vpns4I%2FeKkYPAaavz5SSCygzZoE%3D&response-content-disposition=inline%3B%20filename%3DDefect_Analysis_and_Prevention_for_Softw.pdf>

1. Preuzeto iz [1] str. 29 [↑](#footnote-ref-1)
2. Preuzeto iz [1] str. 30 [↑](#footnote-ref-2)
3. [1] Jeff Tian: Software quality engineering (testing quality assurance and quantifiable improvement), str. 341, 342; [↑](#footnote-ref-3)
4. Preuzeto iz [1], str. 341; [↑](#footnote-ref-4)
5. Preuzeto iz [1], str. 342; [↑](#footnote-ref-5)
6. Preuzeto iz [1], str. 344; [↑](#footnote-ref-6)
7. [1] Jeff Tian: Software quality engineering (testing quality assurance and quantifiable improvement), str. 343; [↑](#footnote-ref-7)
8. [1] Jeff Tian: Software quality engineering (testing quality assurance and quantifiable improvement), str. 345, 345, 347; [↑](#footnote-ref-8)
9. Preuzeto iz [1] – str. 346, 347; [↑](#footnote-ref-9)
10. Preuzeto iz [1] – str. 349; [↑](#footnote-ref-10)
11. Preuzeto iz [1] – str. 350; [↑](#footnote-ref-11)