TESTIRANJE SOFTVERA

SANJA MIJALKOVIĆ

1061/2013

- Development testing – testovi u toku razvoja
- Test-driven development – razvoj vođen testovima
- Release testing
- User testing

TESTIRANJE PROGRAMA

- Testiranje služi kao provera da li program ima predvidjenu funkcionalnost i da otkrije greške u programu pre nego što je pušten u upotrebu.
- Kada testiramo program treba da koristimo i nespecifične ulazne podatke.
- Testiranjem možemo da otkrijemo da postoje greške u kodu ali ne i da dokazemo njihovo odsustvo.
- Testiranje predstavlja deo šireg procesa – Validacije i Verifikacije softvera (V&V).

ZAŠTO TESTIRAMO PROGRAM?

- Da bi lakše pokazali korisnicima i razvijaocima da softver zadovoljava željene zahteve i funkcionalnosti.
  - Koristimo **testove validacije.**
  - Tada očekujemo od sistema da za skup test primera radi korektno.
  - Uspešni testovi pokazuju da sistem pruža željene funkcionalnosti.
- Da bi programeri lakse primetili i locirali greške i probleme u programu.
  - Koristimo **defect testiranje**.
  - Test primeri korišćeni pri ovakvom testiranju su pravljeni kako bi izazvali pojavljivanje grešaka (ukoliko one postoje) i pokazali eventualno neželjeno ili neočekivano ponašanja sistema.
    - Ovi testovi ne ilustruju nužno funkcije i ulogu sistema.
    - Dobro osmišljeni testovi čine da se pojave i skrivene greške, one koje bi se retko pojavile pri normalnom radu sistema.

VERIFIKACIJA I VALIDACIJA SOFTVERA (V & V)

Verifikacija:

- ✓ “Are we building the product right?”
- ✓ Provera da li softver zadovoljava zeljenu funkcionalnost i postavljene zahteve.

Validacija:

- ✓ “Are we building the right product?”
- ✓ Provera da li softver zadovoljava stvarne potrebe korisnika (ili ulagača).
- ✓ Validacija je važna jer postavljeni zahtevi ne definišu uvek precizno stvarne potrebe i želje korisnika softvera.

Cilj V & V je da sa određenom sigurnošću pokaže da softver služi svrsi. Potreban stepen sigurnosti V & V zaljučaka zavisi od:

- Svrhe softvera - što je bitnije da softver bude pouzdan to je potrebno da nivo sigurnosti V&V zaključaka bude viši.
- Očekivanja korisnika – korisnicima ne mora biti od presudne važnosti pouzdanost nekog softvera.
- Ekonomskog okruženja – Nekada je bitno pojaviti se prvi na tržištu i po cenu manje pouzdanosti softvera.

PROVERA SOFTVERA (*SOFTWARE INSPECTIONS*)

- Odnosi se na „statičku“ proveru, pre implrementacije kompletnog softvera, za razliku od testiranja koje podrazumeva „dinamičku“proveru prilikom izvršavanja.
- Uključuje ljude koji pregledaju izvorni kod kako bi našli anomalije, greške i delove koda koji bi mogli izazvati neželjeno ponašanje prilikom izvršavanja.
- Može biti primenjeno na bilo koji segment sistema (očekivanja, zahteve, dizajn softvera)
- Koristi se znanje o čitavom sistemu, UML modeli, šeme baze podataka, domen aplikacije.
- Ipak ne moze zameniti testiranje softvera.

PREDNOSTI I NEDOSTACI ‘INSPEKCIJE’ SOFTVERA

- Tokom testiranja greške mogu lako sakriti druge greške. Kako je inspekcija staticki proces, nema razloga za brigu o „vezanim“ greškama.
- Moze se primeniti i na nepotpunim - nedovršenim sistemima, pre konačne implementacije softvera.
- Pored pronalaženja grešaka, tokom ‘inspekcije’ mogu biti sagledane i druge važne karakteristike sistema, kao sto su saglasnost, prenosivost i održivost.
- Inspekcija ne može da proveri nefunkcionalne karakteristike sistema kao sto su performance, upotrebljivost i druge.

FAZE TESTIRANJA:

- **Development testing** – testiranje u fazama razvoja kako bi greške bile otkrivene i otklonjene iz programa.
- **Release testing** – testiranje čitavog sistema od strane posebnog tima za testiranje pre nego sto se sistem konačno pusti u upotrebu.
- **User testing** – kada korisnici ili potencijalni korisnici testiraju sistem u svom okruženju.

DEVELOPMENT TESTING

Predstavlja sva testiranja izvršena od strane razvojnog tima na sistemu.

Dele se na :

- **Unit tests** – testiranje jedinica koda, gde je svaka jedinica koda zasebno testirana, pri čemu jedinica koda može biti funkcija, metod, klasa, operacija, struktura podataka...
- **Component testing** – testiranje komponenata, pri čemu komponenta predstavlja integrisane jedinice koda koje zajedno pružaju neku funkcionalnost, tako da se ovo testiranje uglavnom bavi testiranjem interfejsa te komponente.
- **System testing** – testiranje sistema kao celine. Kako su sve njegove komponente integrisane, ovo testiranje se bavi interakcijom izmedju različitih komponenata.

TESTOVI JEDINICA KODA

- ✓ Uloga je testiranje jedne izolovane jedinice koda u cilju provere njene funkcionalnosti.
- ✓ Proverava da li u jedinici postoji neka greška ili neželjeno ponšanje.
- ✓ Spada u ‘defect testing’.
- ✓ Ispitivanje robusnosti (da li se jedinica ponaša ispravno i za neispravne ulazne podatke)
- ✓ Jedinica koda moze biti:
  - funkcija
  - metod
  - struktura podataka
  - klasa koja sadrži atribute i metode
  - operacija

TESTIRANJE KLASE

- Celokupno testiranje klase obuhvata:
  - Testiranje svih operacija povezanih sa objektima klase.
  - Postavljanje i ispitivanje svih atributa objekata.
  - Dovodjenje objekta u sva moguća stanja.
    - Stimulisati sve dogadjaje koji izazivaju promenu stanja objekta.
- Nasledjivanje čini testiranje klase teže, jer jedinica koda koja se testira nije u potpunosti lokalizovana.
  - U tom slučaju moramo testirati nasledjenu operaciju u svim kontekstima u kojima se koristi.

AUTOMATSKO TESTIRANJE

- Kada god je moguće, poželjno je automatizovati testiranje jedinica koda, tako da se testovi pokreću i njihovi rezultati proveravaju automatski.
- Pri automatskom testiranju može se koristiti framework za automatizaciju testiranja (kao sto je JUnit) za izradu i pokretanje testova za program.
- Framework za testiranje jedinica koda pruža generičke klase testova koje treba naslediti za izradu specifičnih testova. Postoji mogućnost pokretanja svih implementiranih testova i prikazivanja njihovih rezultata (najčešće kroz neki GUI). Na ovaj način ceo skup testova može biti pokrenut u par sekundi, što omogućava izvršavanje svih testova pri svakoj promeni programa.
- Komponente automatskih testova:
  - **Setup part** – gde inicijalizujemo test, pre svega ulaz i očekivani izlaz.
  - **Call part** – kada pozivamo objekat ili metod kako bi bili testirani.
  - **Assertion part** – poređenje rezultata poziva sa očekivanim rezultatima. Ukoliko se izlazi poklapaju, test prolazi, u suprotnom test ne prolazi.

SVRHA TESTIRANJA JEDINICA KODA

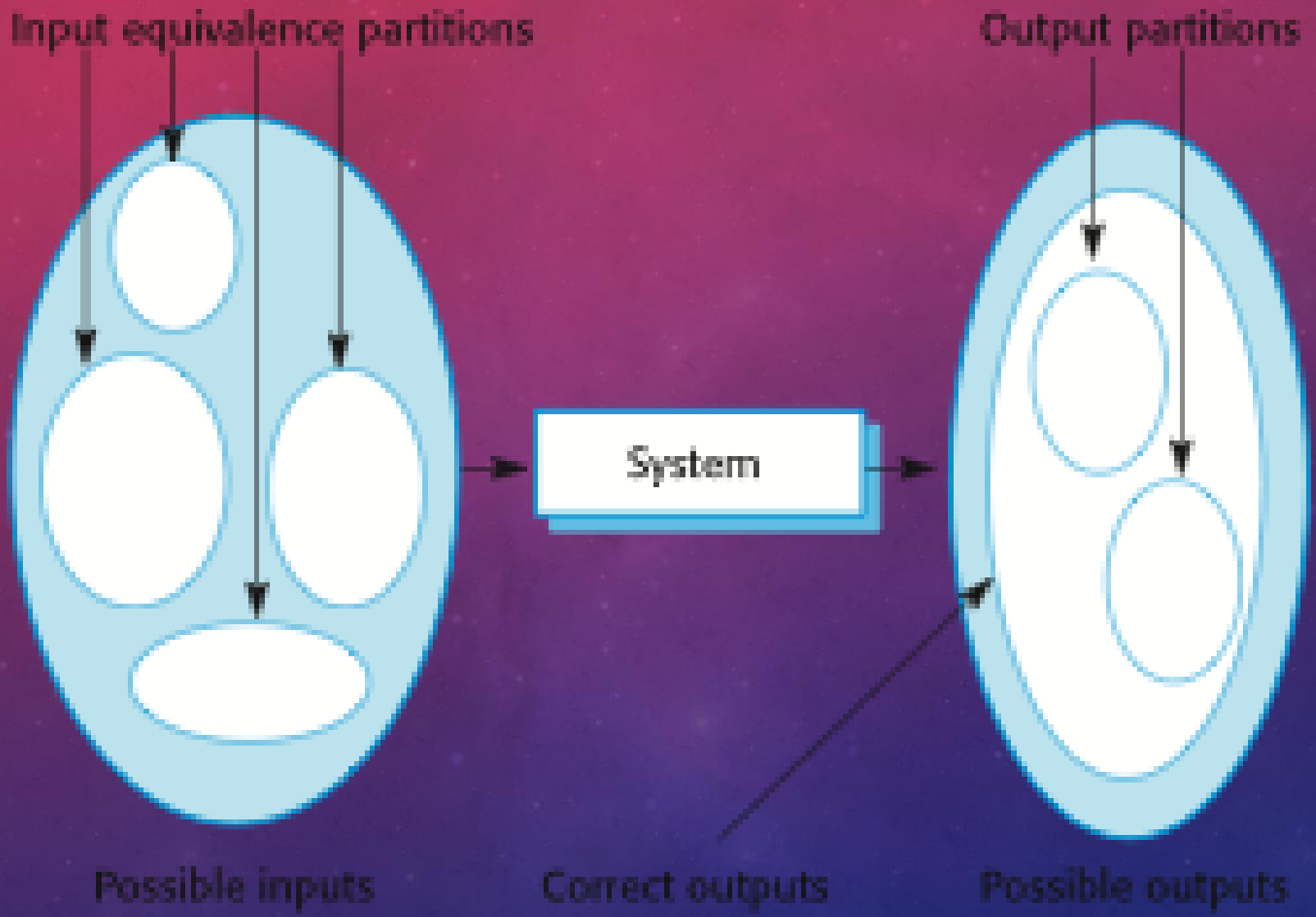
- Test primeri bi trebalo da pokazu da komponenta ima predvidjenu funkcionalnost i nema neželjeno ponašanje.
- Ukoliko postoje greške u komponenti, one bi trebalo da budu otkrivene u fazi testiranja (ili inspekcije) te komponente.
- Odavde imamo dva tipa testiranja jedinica koda
  - Prvi bi trebalo da pokaze da jedinica radi u skladu sa planovima.
  - Drugi bi trebalo da budu zasnovani na dosadasnjem iskustvu u testiranju. Treba koristiti speijalne slučajeve ulaza, isprobati granice domena, kao i nekorektan ulaz kako bi obezbedili da ne dolazi do pada sistema pri ovim situacijama.

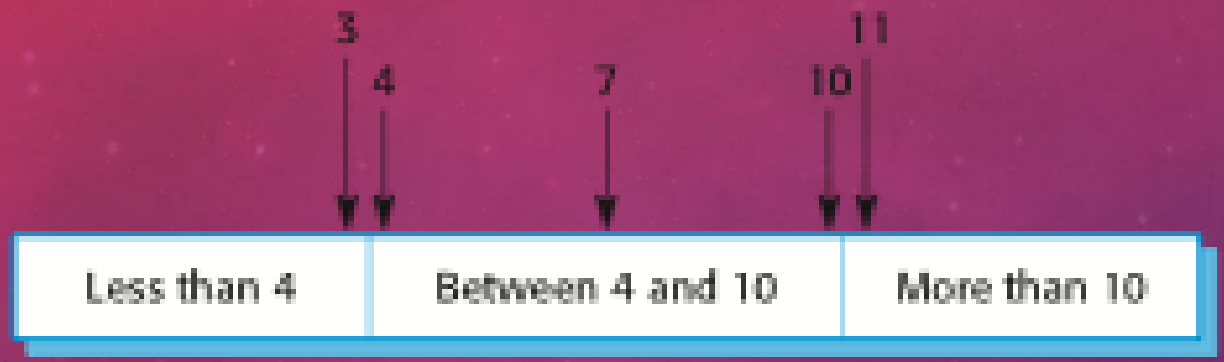
STRATEGIJE TESTIRANJA JEDINICE KODA

- **Partition testing** - particionisano testiranje
  - Ovde identifikujemo klase ulaza koje su sličnog karaktera i koje bi trebalo da se ponašaju na sličan način.
- **Guideline – based testing**
  - U ovoj vrsti testiranja koristimo uputstva za biranje test slucajeva koja su napravljena na osnovu prethodnih iskustava. Uputstva su napravljena po ugledu na greške koje su se često javljale u programima i na kojim mestima bi trebalo obratiti pažnju.

PARTITION TESTING

- Ulazni i izlazni podaci se često mogu grupisati u različite klase gde se svi članovi jedne klase ponašaju na isti nacin. Te klase se nazivaju *equivalence partition* – particije ekvivalencije ili domen, gde se program ekvivalentno ponasa sa svim članovima iste klase.
- Test primeri bi trebalo da budu odabrani iz svake klase. Trebalo bi odabrati test primere koji su granični slucajevi te klase i midpoint klase (srednju vrednost). Razlog za ovakvo biranje test primera je da bi razvijaoci testirali sve specifične slučajeve ulaza za svaku klasu.


Input equivalence partitions
Output partitions
System
Possible inputs
Correct outputs
Possible outputs



Number of input values

Input values

VRSTE TESTIRANJA

- **Black-box testing** - kada koristimo specifikacije sistema kako bismo identifikovali klase ulaznih podataka. Nije nam potrebno gledanje izvornog koda niti znanje o tome kako sistem radi, vec šta radi.
- **White-box testing** – kada gledamo u izvorni kod programa kako bi našli potencijalne test primere.
    - Naš kod može sadržati izuzetke za nepravilne ulazne podatke, pa ih tu možemo lako pronaći.

UPUTSTVA ZA TESTIRANJE PROGRAMA SA NIZOVIMA

- Testirajte softver nizom koji sadrži samo jedan element.
- Testirajte softver nizovima različite dužine.
- Pokrenite test tako da prvi, posledji i srednji element niza budu uključeni u testiranje.
- Testirajte program korišćenjem praznog niza.

UOPŠTENA UPUTSTVA ZA TESTIRANJE

- Birajte ulaz koji navodi sistem da generiše sve poruke o greškama.
- Napravite ulaz koji prepuni bafere za ulaz.
- Ponovite isti ulaz ili niz ulaza veliki broj puta.
- Naterajte program da generiše neispravan izlaz.
- Naterajte program da rezultati izračunavanja budu preveliki ili premali.

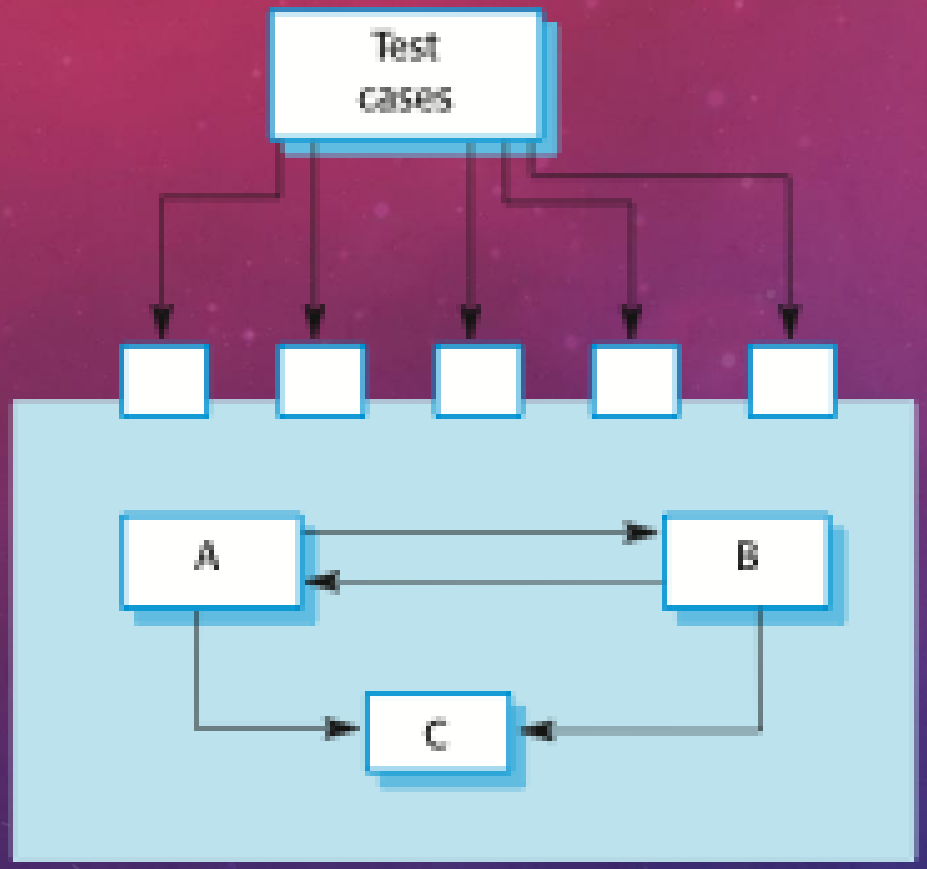
KLJUČNE STAVKE

- Testiranje može samo da nam otkrije samo greške u kodu ali ne i da dokaže njihovo odsustvo.
- Testiranje u fazi razvoja (Development testing) je zadatak članova ravojnog tima. Odvojen tim testera bi trebalo da uradi testiranje sistema (System testing) pre nego što bude pušten u upotrebu.
- Testiranje u fazi razvoja (Development testing) obuhvata testiranje jedinica koda (gde se testiraju metode i atributi), testiranje komponenti (gde se testira interfejs te komponente, odnosno njena funkcionalnost) i testiranje sistema ili samo nekih delova sistema.

TESTIRANJE KOMPONENTI

- Softverske komponente su sastavljene od nekoliko jedinica koda koje međusobrno interaguju.
- Funkcionalnost koju jedna takva komponenta pruža se može videti kroz njen interfejs.
- Testiranje konponente bi, shodno tome, trebalo da se fokusira na pokazivanje da komponenta ima odgovarajuću funkcionalnost i ponaša se kako je predviđeno u zahtevima.
    - Može se pretpostaviti da su izvršeni testovi jedinica koda od kojih je komponenta sačinjena.

PRIMER



- Jedinice koda A, B i C su integrisane u komponentu koja pruža neku funkcionalnost.
- Test primeri ne komunicariju direktno sa A , B ili C , već testiraju interfejs cele komponente.

TIPOVI INTERFEJSA

- ***Parameter interfaces*** - Podaci se prosleđuju iz jednog metoda u drugi.
- ***Shared memory interfaces*** – Blok memorije je deljen između procedura ili funkcija.
- ***Procedural interfaces*** – Komponenta (podsistem) enkapsulira skup procedura koje mogu biti pozvane iz druge komponente (podsistema).
- ***Message passing interfaces*** – Jedna komponenta zahteva usluge druge komponente.

GRESKE U INTERFEJSU

- *Interface misuse*
  - Kada komponenta poziva drugu komponentu i pravi greske u upotrebi njenog interfejsa.
    - Pošalje parameter u pogrešnom redosledu, ili slično.
- *Interface misunderstanding*
  - Kada komponenta ima netačne pretpostavke o ponašanju komponente koju poziva.
- *Timing errors*
  - Kada komponente koje komuniciraju rade drugačijim brzinama i nisu lepo usklađene, pa se zbog toga koriste podaci koji nisu vise validni.

SMERNICE ZA TESTIRANJE INTERFEJSA

- Dizajnirajte test primere tako da parametri procedura budu na granicama njihovih opsega.
- Uvek testirajte parametre koji su pokazivači sa NULL vrednostima.
- Dizajnirajte testove koji uzrokuju greške u komponentama.
- Naterajte komponentu da ispiše sve moguce poruke, vise nego sto bi se desilo u normalnim uslovima rada (*stress testing in message passing systems*).
- Kada ima deljenja memorije, varirajte redosled u kome su komponente aktivne.

TESTIRANJE SISTEMA

- Testiranje sistema u toku razvoja uključuje integrisanje komponenti kako bi dobili neku verziju sistema koju bismo onda testirali.
- Težište u testiranju sistema je testiranje međusobnih veza između komponenti sistema.
- Testiranje sistema proverava da li su komponente kompatibilne, da li pravilno sarađuju i prenose prave podatke u odgovarajućem trenutku putem interfejsa.
- Testiranje sistema testira ponašanje sistema koje postaje očigledno tek kada spojimo sve komponente.
    - Moze se desiti i da dođe do pojave novih, neželjenih funkcionalnosti kada se komponente povežu, kao rezultat povezivanja komponenti. Iz tog razloga, trebalo bi proveriti da li sistem ima samo potrebne i planirane funkcionalnosti.

TESTIRANJE SISTEMA I TESTIRANJE KOMPONENTI

- Komponente mogu razvijati i različiti članova tima ili podtimova.
- Svaki put kada se razvije nova komponenta sistema, dodaje se u sistem a zatim se opet vrši testiranje sistema kao celine.
- Zbog toga kažemo da je testiranje sistema kolektivni proces pre nego individualni proces.
  - U nekim firmama testiranje sistema radi odvojen tim testera u koji nisu uključeni dizajneri ili programeri koji su radili na tom sistemu.

TESTIRANJE SLUČAJEVA UPOTREBE

- Svaki slučaj upotrebe obično je implementiran pomoću vise komponenata i testiranje svakog slučaja upotrebe će naterati te komponente da međusobno interaguju na određeni nacin.
- Pomoću dijagrama sekvenci svakog slučaja upotrebe možemo pratiti da li se naš sistem ponaša u skladu sa zahtevima, da li su svi objekti aktivni u trenucima u kojima je naglašeno da treba da budu aktivni i obrnuto, kao i da li razmenjuju sve odgovarajuće podatke.
- Na osnovu dijagrama sekvenci možemo napraviti specifične test primere koji bi demonstrirali koji ulazi su potrebni a koji izlazi su očekivani.

POLITIKA TESTIRANJA

- Teško je znati kada stati sa testiranjem i koliko testiranja je dovoljno za koji sistem.
- Nemoguće je testirati baš sve okolnosti u kojima sistem može biti pokenut, sa svim mogućim kombinacijama ulaznih parametara.
- Moguće je testirati sistem samo nekim podskupom svih kombinacijam ulaznih parametara.
- Primeri politika testiranja:
  - Sve funkcije sistema kojima se može pristupiti iz menija treba da budu testirane.
  - Treba testirati i kombinacije funkcija kojima se pristupa kroz isti meni.
    - Na primer, formatiranje teksta
  - Treba testirati sve funkcije koje očekuju neki ulaz koristeći i pravilne i nepravilne ulazne podatke.

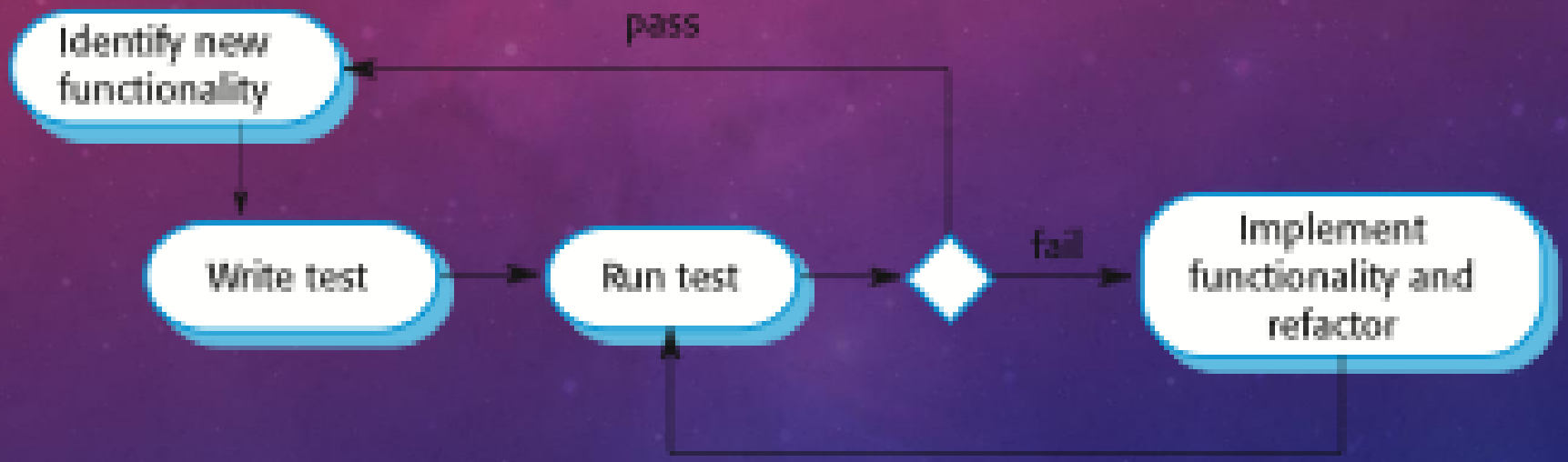
TEST-DRIVEN DEVELOPMENT

- Razvoj vođen testovima je jedan od pristupa u razvoju softvera, gde se neprestano smenjuju testiranje i razvijanje koda.
- Testovi se izrađuju pre kodiranja i prolaženje testova je kritična tačka u razvoju.
- Kod se razvija inkrementalno, zajedno sa testovima za celinu koja se trenutno razvija. Ne prelazi se na sledeci inkrementalni korak sve dok kod tekuće inkrementalne celine ne prodje sve testove napisane za nju.
- Razvoj vođen testovima je deo Ekstremnog programiranje, jedne od najpoznatijih metodologija Agilnog razvoja. Može se koristiti i u planskom razvoju procesa (*plan-driven development process*) .
- Pokazalo se kao izuzetno dobro za male i projekte srednje veličine.
  - Programeri koji su usvojili ovakav način programiranja su zadovoljni i tvrde da je produktivniji način za razvoj softvera.

KORACI U PROCESU

1. Uočavanje inkrementalnih celina procesa.
   - Inkrementalne celine ne bi trebalo da budu mnogo velike.
2. Izrada i implementiranje automatizovanih testova za funkcionalnost inkrementalne celine.
3. Pokretanje napisanog testa zajedno sa svim ostalim testovima koji su već implementirani.
   - Inicijalno testovi nece prolaziti, jer se prvo pišu testovi za jedinicu koda pa se tek onda implementira kod koji prolazi taj test.
4. Implementiramo jedinicu koda i opet pokrenemo testove.
5. Prelazimo na novu inkrementalnu celinu tek kada svi napisani testovi prođu uspešno.

KORACI U PROCESU



PREDNOSTI RAZVOJA VOĐENOG TESTOVIMA

- Pokrivenost koda
  - Svaki segment koga koji napišemo ima barem jedan test koji testira njegovu funkcionalnost.
- Regression testing
  - Testovi se razvijaju uporedo sa kodom. Uvek možemo pokrenuti regresione testove kako bismo proverili da promene nisu izazvale pojavu novih grešaka.
- Jednostavno debagovanje
  - Kada test ne prodje očigledno je gde je greška u kodu. Novonapisani kod treba da se proveri i modifikuje.
- Dokumentacija
  - Testovi predstavljaju temeljan vid dokumentacije u kojoj je opisano šta bi kod trebalo da radi.

REGRESSION TESTING

- Regresiono testiranje predstavlja pokretanje svih skupova testova koji su prethodno bili uspešno izvršeni na kodu.
- Ovi testovi omogućavaju proveru da li su promene u kodu dovele do nastanka novih grešaka i omogućavaju njihovo otkrivanje.
- Ukoliko se radi o automatizovanom testiranju, kao što je ovde reč, regresiono testiranje je relativno jednostavno izvršiti, za razliku od ručnog testiranja. Jednostavno se svi testovi ponovo pokrenu svaki put kada su napravljene neke promene u kodu.
- Svi testovi moraju biti uspešni kako bi promene bile potvrdjene.
- Zahvaljujuci regresionom testiranju možemo biti sigurni da nova funkcionalnost nije proizvela probleme u dosadašnjem kodu.

RELEASE TESTING

- *Release testing* predstavlja testiranje neke konkretne verzije sistema za koju se planira da se preda na korišćenje korisnicima van članova razvojnog tima.
- Primarni cilj ovog testiranja je da pokaže ulagačima da je dovoljno dobar za upotrebu.
  - Ovim testiranjem se pokazuje da sistem pruza funkcionalnost i performanse naglasene u zahtevima, kao i da ne dolazi do pada sistema pri radu u realnim uslovima.
- Obično je *black-box testing* proces, pošto se testovi uglavnom razvijaju na osnovu postavljenih zahteva (specifikacije sistema).

RELEASE TESTING - SYSTEM TESTING

- *Release testing* je jedan vid testiranja sistema.
- Važne razlike:
  - Poseban tim ljudi koji nema veze sa razvojnim timom sistema bi trebalo da izvrši *release testing*.
  - Testiranja sistema koja vrši razvojni tim imaju za cilj pronalaženje grešaka u sistemu, spadaju u *defect testing*.
  - *Release testing* proverava da li sistem ispunjava sve što je navedeno u njegovoj specifikaciji i da li odgovara zahtevima klijenta, dakle spada u *validation testing*.

TESTIRANJE PERFORMANSI

- Deo *release testing*-a, kao sto je već napomenuto, se fokusira da testiranje performansi sistema.
- Test bi trebalo da uzme u obzir način upotrebe sistema, kako bismo u realnom vremenu mogli da izmerimo performanse od značaja za taj način upotrebe.
- Testovi performansi obično uključuju planiranje niza tastova kojima se opterećenje stalno povećava, sve dok performanse ne postanu neprihvatljive.
- *Stress testing* je jedan od načina da testiramo performanse jer je tada sistem preopterećen. Na taj način testiramo i da li dolazi do grešaka sistema pri ovakvom radu i koliko je sistem izdržljiv.

USER TESTING

- Testiranje korisnika je faza testiranja u kojoj korisnici ili ulagači obezbeđuju ulaze i instrukcije za testiranje sistema.
  - Na ovaj način se preciznije može proveriti da li sistem udoban za korišćenje korisnicima.
  - Korisnici proveravaju da li je obezbedjena zahtevana funkcionalnost sistema.
- Testiranje korisnika je od velike važnosti, čak i kada su prethodno izvršena temeljna testiranja sistema.
  - Pokazuju kako sistem radi u okruženjima koja koriste korisnici.
  - Mnogo govori o performansama, upotrebljivosti i robusnosti softvera.
  - Takvi uslovi se ne mogu obezbediti kod razvijaoca, pa čak ni u posebnom timu testera.

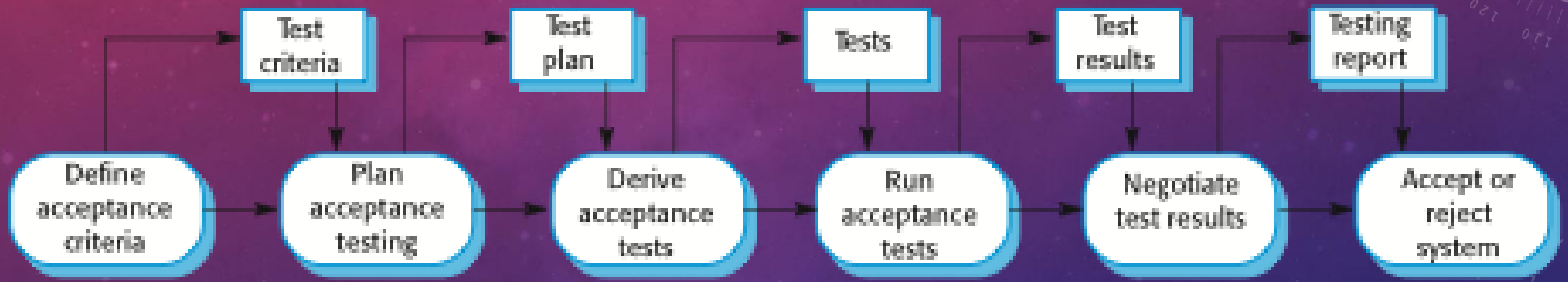
VRSTE *USER TESTING*-A

- ***Alpha testing***
  - Korisnici softvera sarađuju sa razvojnim timom kako bi testirali softver iz ugla razvijaoca.
- ***Beta testing***
  - Kada je softver pušten u upotrebu korisnicima kako bi eksperimentisali i prijavili razvijaocima probleme koji se javljaju pri upotrebi u realnim uslovima.
- ***Acceptance testing***
  - Korisnici testiraju softver i odlučuju da li je dovoljno dobar da se pusti u upotrebu.

FAZE PROCESA TESTIRANJA PRIHVATLJIVOSTI

- Definisanje kriterijuma prihvatanja
- Planiranje testova prihvatljivosti
- Razvijanje testova prihvatljivosti
- Pokretanje testova prihvatljivosti
- Diskutovanje o rezultatima testova prihvatljivosti
- Prihvatanje ili odbacivanje sistema

PROCES TESTIRANJA PRIHVATLJIVOSTI



AGILNE METODE I TESTOVI PRIHVATLJIVOSTI

- U Agilnim metodologijama, korisnik/ulagač je deo razvojnog tima i odgovoran je za pravljenje odluka o prihvatljivosti sistema.
- Testove definiše korisnik/ulagač, oni se integrišu sa ostalim testovima i pokreću se automatski svaki put kada su napravljene neke izmene u sistemu.
- Ne postoji odvojen proces testiranja kojim se testira prihvatljivost sistema.
- Glavni problem sa ovakvim pristupom je što ne možemo biti sigurni da je korisnik/ulagač dobar predstavnik ciljne grupe korisnika sistema.

Hvala na pažnji!