Testiranje softvera

Industrijski softver u odnosu na "akademske" programe

- "Akademski" softver:
  - Uglavnom se pravi u svrhu učenja, istraživanja, demonstracije ili iz hobija, ne rešava neki realan problem
    - Sledi da prisustvo grešaka (bagova, defekata) ne zabrinjava
  - Upotrebljava ga uglavnom sam autor, tako da nije važno dokumentovanje, a bagove ispravlja sam autor ako na njih naiđe
  - Životni vek je kratak, najčešće za jednokratnu upotrebu

Industrijski softver

- Engl. industrial strength software
- Napravljen da rešava neki poslovni problem korisnika
- Važne aktivnosti mogu zavisiti od korektnog funkcionisanja sistema
  => loše funkcionisanje izaziva nezadovoljstvo korisnika i finansijske, materijalne gubitke, ili čak ljudske žrtve

Čuveni primeri softverskih otkaza (1)

- **Aerodrom u Denveru**: 1994, zakašnjenje u otvaranju od skoro godinu dana zbog nefunkcionisanja automatizovanog sistema za transport prtljaga (inicijalna cena tr.sistema 234 M$, troškovi zakašnjenja 1 M$ dnevno)
- **Deutsche Telekom**: pogrešan proračun cene telefonskih impulsa za 1.1.96 (softverska greška: bez praznične tarife – šteta: stotine miliona DEM)
- **Prvi internet crv** (program koji se sam umnožava i širi): 1988 zaraženo nekoliko hiljada računara, bug u virusu prouzrokovao zagušenje računara, šteta 10-100 M$. Koristio rupe u sendmailu, rsh/execu. autor R.T.Moris sin inženjera NSA uslovno i novčano osuđen, sada prof na MITu

Čuveni primeri softverskih otkaza (2)

- **Mariner I, prva svemirska raketa za Veneru:** 1962 usled softverskog otkaza automatizovanog sistema za navođenje morala biti uništena daljinskom komandom 5 min posle lansiranja (gubitak 18 M$).
- **Ariane 5 let 501**, 1996 raspala se 40s posle lansiranja usled prekoračenja u konverziji float->int, šteta 370 M$
- **Therac 25 – računarski kontrolisani uredjaj za terapijsku radijaciju:** izmedju juna 1985. i januara 1987. 6 ljudi je predozirano (5 od njih je kasnije umrlo) kao posledica nedostajuće softverske sigurnosne brave koja bi trebala da spreči predoziranje

Ima li šta gore od toga?

- **1980 kompjuterski sistem NORAD je javio da su SAD izložene raketnom napadu.** Problem je bio hardverski i nastao je od jednog kratkog spoja, ali softver za izveštavanje nije bio projektovan da uzme tu mogućnost u obzir.
- **1983 sovjetski satelit je lažno detektovao dolazak raketa iz SAD.** Usled greške u softveru, radarski odraz izlaska sunca detektovan je kao dolazak projektila, a pukovnik Stanislav Petrov sprečio potencijanu katastrofu samostano odlučivši, na bazi lične procene, da je u pitanju lažna uzbuna i da ne treba uraditi ništa.

Industrijski softver

- Posledice potreba za kvalitetom:
  - 30% do 50% ukupnog napora (troška) je na testiranje, za akademski softver ne prelazi 5%
  - Zahtevi za planiranim razvojem po fazama, dokumentovanjem, pridržavanjem raznih standarda, ispunjavanjem različitih nefunkcionalnih zahteva (sigurnost, portabilnost, performanse)
  - Ovakav softver zahteva 10x više napora za istu funkciju od “akademskog”
  - prosečna produktivnost po osobi u celokupnom ciklusu razvoja industrijskog softvera je 300 do 1000 LOC/mes (LOC = linija izvornog koda)

Veličina industrijskog softvera

- Mali projekti <= 10 KLOC
- Srednji <= 100 KLOC
- Veliki <= 1 MLOC
- Veoma veliki – više MLOC

Industrijski softver i važnost njegovog testiranja

- Veličina industrijskog softvera:
  - Windows 2003: ~50 MLOC (miliona linija izvornog koda)
  - Linux kernel: 2.6.32 >12 MLOC
- Američki nacionalni institut za standarde (NIST) procenjuje da su 2002. softverski bagovi izazvali 60 milijardi $ gubitaka u američkoj ekonomiji
- Istraživanja Univerziteta u Kembridžu daju procenu od 312 milijardi $ na globalnom nivou za 2012.

Ključne stvari pri testiranju

Sve

Svako

Sve vreme

(i pored toga se dešava da nedostaci prođu testiranje)

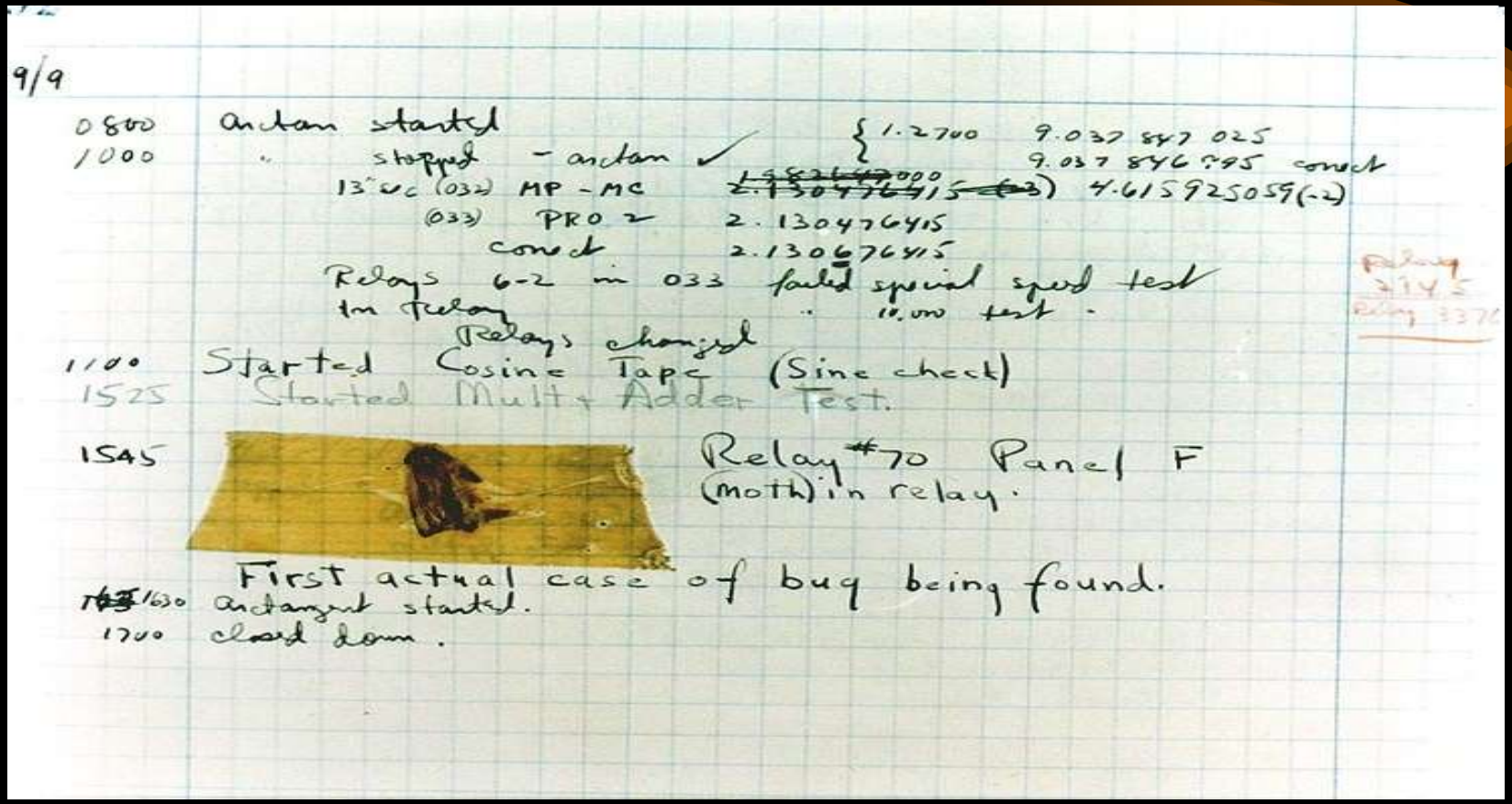
Razvoj softvera u odnosu na druge tehničke discipline

- Jedno ispitivanje u američkom ministarstvu odbrane pokazalo je da se čak 70% otkaza opreme može pripisati softveru, u sistemima punim električnih, mehaničkih i hidrauličnih komponenata
- Druge tehničke discipline su znatno zrelije, softver je često slaba tačka
- Otkazi fizičkih sistema javljaju se usled fizičkih i električnih promena uzrokovanih starenjem
- Softver ne stari, greške se nalaze u njemu od početka, a mogu se manifestovati u vidu otkaza i posle dužeg ispravnog rada

Terminologija u vezi sa greškama i testiranjem

- Izvor: Glossary of Software Engineering Terminology. ANSI/IEEE Std. 729–1983
- **Greška (Error)**
  - Napravi je čovek, na primer, prilikom specifikacije zahteva ili kodiranja programa
- **Mana, defekat (Fault)**
  - posledica greške (na primer, programu nešto nedostaje, ili ima funkciju ali neispravno - “bug”)
- **Otkaz (Failure)**
  - Nemogućnost sistema da obavi zahtevanu funkciju najčešće se javlja aktiviranjem (izvršavanjem) defektnog koda.

Prva kompjuterska "bubica"



Moljac pronađen u releju računara testiranog na Harvard univerzitetu septembra 1945 i prikačen u dnevnik

Programski nedostaci su analogni materijalnim nedostacima

- Svi (ne-trivijanlni) softverski sistemi imaju nedostatke
  - Kaže se da se dokumentacija ranijih softverskih sistema uglavnom sastojala od izveštaja o nedostacima
- Neki bivaju uočeni odmah
- Drugi su stalno prisutni i može se desiti da nikada ne budu otkriveni
- Neki se, pak, ukažu u najgorem mogućem ili čak kritičnom trenutku

Specifičnosti programskih nedostataka

- Najčešći nedostaci su u
  - Programskoj logici
  - Definisanju podataka ili struktura podataka
- Nedostaci su 'nasumično raspodeljeni'
- Teško ih je pronaći
- Najpogodnije je kada je nedostatak samo jednom u modulu
  - Ovo favorizuje modularnu ili objektnu strukturu programa

Validacija i verifikacija

Validacija = željeni proizvod

Verifikacija = proizvod je dobar

Etape testiranja

Jedinice (Unit)

Moduli (Module)

Podsistem (Subsystem)

Integracija (Integration)

Prijemno (Acceptance)

Testiranje postaje sve teže kako se ide kroz ovu listu!

Testiranje jedinica

Razumno napredovanje

ako je jedinica projektovana efikasno

i ako je dobro specificirana

Testiranje modula

Skup povezanih,

međusobno zavisnih delova

Testiranje podsistema

Problemi se često javljaju

zato što različiti softverski inženjeri

različito interpretiraju specifikacije.

Testiranje integracije

Validnost se prvi put proverava

U toku procesa

Pitanje performansi

Realni podaci

Prijemno testiranje

Javljaju se nedostaci
iz faze specifikacije zahteva

Ova faza se naziva etapa alfa testiranja

[Mogu biti sprovedena završna plaćanja]

Beta testiranje

Distribucione verzije

Poverljivim,

ponekad čak i nepoznatim,

potencijalnim korisnicima.

Mnogo su efikasniji od većine profesionalnih testera

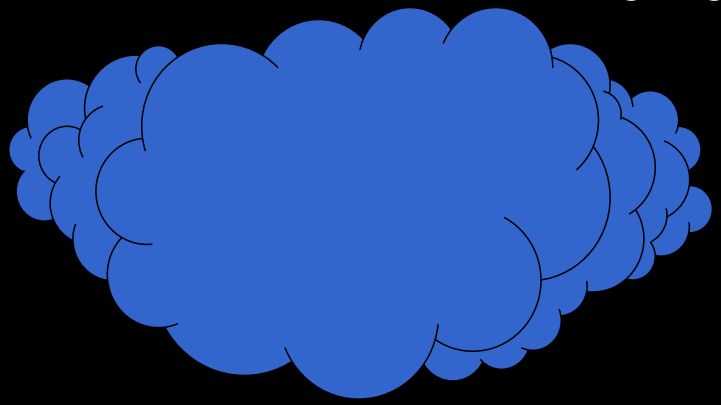
Testiranje

Metod "crne kutije"

Metod "bele kutije"

Crna kutija (Black Box)

Ništa ne znamo o
unutrašnjoj strukturi softvera



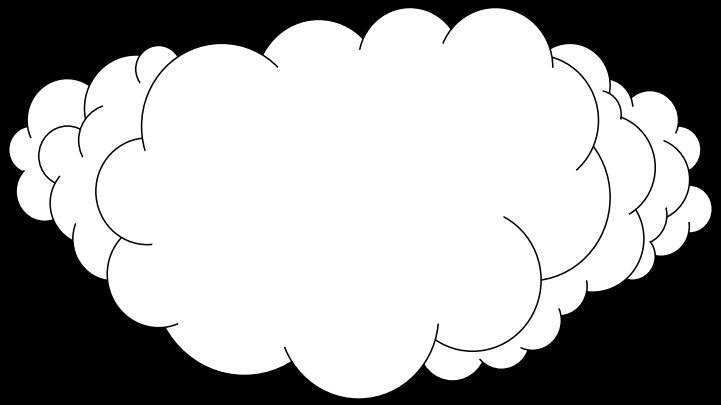
Oslanjamo se na
neku vrstu specifikacije

Tehnike crne kutije

Provera da li softver

ispunjava zahteve specifikacije

Bela kutija (White Box)



Kako sistem
treba da se ponaša

Koristiti ova znanja
da bi poboljšali testiranje

Tako možemo potvrditi da je većina

činjenica,

modula,

itd.

bila testirana.

Cilj testiranja

VALIDACIJA i VERIFIKACIJA
Pomaže nam da pronađemo nedostatke

Drugi razlozi za testiranje

Upoznavanje sa sistemom

Mogućnosti da se predlože poboljšanja

Sticanje poverenja u pogodnost za upotrebu

Pitanja pri testiranju

Koje podatke koristiti

Koliko podataka koristiti

Kada zaustaviti testiranje

Testiranje odozgo na dole i testiranje odozdo na gore

Neophodno je sprovesti
neka testiranja odozdo na gore

Ako se nedostaci ne izoluju teško ih je pronaći

Ipak je neophodno sprovesti dosta testiranja odozgo na dole

Jedini način za pronalaženje strategijskih nedostataka

Većina njih zahteva mnogo resursa za popravku

Testiranje po metodu crne kutije

Brojanje grešaka

Najjednostavnije

Najpouzdanije, mada možda ne i najkorisnije

Prijavljene greške - zavisno od vremena

Podela na klase ekvivalencije (Equivalence Partitioning)

- Razdvajanje "prostora" ulaznih veličina na ekvivalentne klase
- Podela se vrši tako da ce program ponaša na sličan način za sve ulazne vrednosti koje pripadaju istoj klasi ekvivalencije
- Značajno uvećanje efikasnosti testiranja
  - Program se testira po jednom reprezentativnom vrednošću ulaza iz svake klase ekvivalencije
  - Druge vrednosti iz te klase otkrile bi iste greške

Kako odrediti klase ekvivalencije?

- Posmatraju se svi uslovi vezani za ulaze programa koji proizilaze iz specifikacije
- Za svaki uslov se posmatraju dva grupe klasa prema zadovoljenosti uslova:
  - legalne klase obuhvataju dozvoljene situacije
  - nelegalne klase obuhvataju sve ostale situacije
- U svakoj od ovih grupa može biti više klasa

Podela na klase ekvivalencije: primeri

- Ulazna veličina programa je broj godina zaposlenog, koji može biti između 16 i 67
  - Definišu se jedna legalna ($16 \leq godine \leq 67$) i dve nelegalne klase ekvivalencije. ($godine < 16$) i ($godine > 67$)
- Matični broj građana ima 13 cifara
  - Definišu se jedna legalna ($mbr\_c = 13$) i dve nelegalne klase ekvivalencije. ($mbr\_c < 13$) i ($mbr\_c > 13$)
- Podatak uzima vrednosti iz nekog prebrojivog skupa, pri čemu se program različito ponaša za svaku vrednost, za slučaj ispitne ocene
  - Dozvoljenje klase su (1) nije došao, (2) nije položio, ocena 5, i (3) položio, ocena 6-10
  - Još jedna klasa za vrednosti van tog skupa, nedozvoljena

Formiranje test primera iz klasa

1) Dodeliti jedinstveni broj svakoj klasi
2) Napisati test primere za sve legalne klase
   - Ukoliko je moguće, uključujući u jedan test što više klasa
3) Napisati po jedan test primer za svaku nelegalnu klasu zasebno, da bi se
   - Izbeglo da jedan nelegalan ulaz maskira drugi
   - Lakše pronašao uzrok problema

Analiza graničnih vrednosti (Boundary Value Analysis)

Mnogo grešaka se pojavljuje na prelazima između naših klasa (ili karakterističnih slučajeva)

Uključujući vrednosti ulaza bliske graničnim i one na samoj granici

Preporuke za granične vrednosti

- Ako ulazni uslov precizira opseg vrednosti, napisati testove za same krajeve opsega i testove nevažećih ulaza za situacije odmah iza legalnih krajeva.
  - Npr. ako je ulaz u opsegu -1.0 do 1.0 testirati treba za -1.0, 1.0, -1.0001, 1.0001
- Ako ulazni uslov precizira broj vrednosti, napisati testove za minimalni i maksimalni broj vrednosti i jedan ispod i iznad ovih vrednosti
  - Ako ulazni fajl može da sadrži 1-255 zapisa, napisati testove za 0, 1, 255, i 256 zapisa.
- Primeniti preporuku 1. na izlazne uslove
- Primeniti preporuku 2. na izlazne uslove
  - Ako na stranu ispitnog spiska staje 1-65 redova, napisati testove za 64, 65 i 66 redova, a tome treba dodati i 0 odnosno 1 red
- Ako je ulaz (ili izlaz) programa uređen skup (npr. linearna lista ili tabela), fokusirati pažnju na prvi i poslednji element skupa
- Analizirati mogućnost nalaženja drugih, dodatnih graničnih uslova

Testiranje po modelu bele kutije

Poznajemo strukturu delova programa

Razmisliti o mogućim vrstama grešaka

Formirati testove za njih

Druge korisne ideje

- Foto opcija
  - Memoriše sve korisnikove interakcije
  - Reprizira problem kad se pojavi
  - Omogućena u nekim operativnim sistemima ili alatima
- Ispravka starih nedostataka (Past Bug Sets)
  - Napomene i primeri unošenih podataka koji su doveli do otkrivanja nedostataka
  - Dobra praksa
  - Nove verzije mogu biti testirane na stare nedostatke

Tehnike testiranja

- Regresiono testiranje
- Testiranje na opterećenje (Load testing)
- Testiranje na udare (Stress testing)
- Inspekcija
- Grupno testiranje (Group testing)
- Testiranje višestrukim selektivnim korišćenjem test primera (Selective reuse of test cases)

Pet zabluda o testiranju softvera

1) testiranje samo troši razvojne i finansijske resurse
2) debagovanje je zamena za testiranje softvera
3) testiranje softvera je zamena za debagovanje
4) alati za testiranje su slični; ozbiljno razmatranje koji kupiti je samo gubljenje vremena
5) jednom kada se instalira alat za testiranje, dovoljno je ubaciti svoj kod, uključiti ga i sačekati nekoliko minuta

Specifičnosti testiranja savremenih aplikacija

- kako se aplikacija ponaša u višekorisničkom radu?
- kako varijacije u hardverskoj konfiguraciji utiču na ponašanje aplikacije?
- kako se aplikacija ponaša u uslovima nedostatka memorije, prostora na disku i nedostupnosti kritičnih komponenti hardvera?
- kako se aplikacija ponaša u klijent-server / Web / mobilnom okruženju?
- koji je najveći stepen obezbeđenja kvaliteta koji se može dostići, sa postojećim ograničenjima vremena i budžeta?

Alati za testiranje softvera

- Mercury Interactive
- SQA
- Pure Software
- Test Complete
- ...