PĮ testavimas

Pagal I.Sommerville "Software Engineering", 9 leidimo 8 dalį

Temos

- ✧ Programos testavimas
- ✧ Į testavimą orientuotas programavimas (Test-driven development)
- ✧ Release testavimas
- ✧ Vartotojo testavimas

Programos testavimas

- Testavimo metu siekiama parodyti, kad programa daro tą, ko iš jos tikimasi, bei surasti klaidas anksčiau, negu programą pradeda naudoti jos galutiniai vartotojai.
- Kai testuojame programą, jos vykdymui naudojame specialiai testavimui sugalvotus duomenų rinkinius.
- Tikriname testavimo ataskaitų rezultatus, pastebėtas anomalijas ir nefunkcinių reikalavimų tenkinimą.
- Galime išvengti pernelyg dažno klaidų pasikartojimo, bet ne jų nebuvimo.
- Testavimas yra dalis platesnio patikros ir atestacijos proceso, į kurį dar įeina ir statinė programos patikra.

Programos testavimo tikslai

- Pademonstruoti programuotojams ir užsakovams, kad PĮ atitinka jai iškeltus reikalavimus:
  - Tai reiškia, kad užsakomojo pobūdžio PĮ testo plane turi būti bent po vieną testą kiekvienam reikalavimui.
  - Bendro pobūdžio PĮ turi turėti po testą kiekvienai sistemos savybei bei tų savybių kombinacijoms.
- Surasti tokias situacijas, kada programa veikia neteisingai, nepageidaujamai ar neatitinka reikalavimų specifikacijos:
  - Defektų testavimas susideda iš sistemos lūžimų (crash), nepageidaujamos sąveikos su kitomis sistemomis, neteisingo skaičiavimo ar duomenų praradimo paieškos.

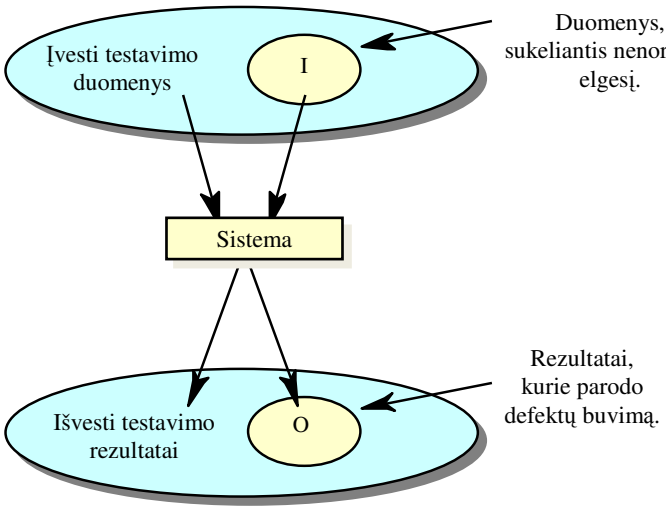
Atestavimas ir defektų radimas

- Pirmas tikslas yra atlikti/praeiti atestacijos testą
  - Jūs tikitės, kad sistema veiks teisingai su duotais testavimo duomenimis, parinktais pagal tai, kaip mes tikimės, kad programa turi veikti.
- Antras tikslas yra atlikti defektų radimo testą
  - Testus projektuojame taip, kad rastume kaip galima daugiau defektų (t.y. "laužiame" programą). Defektų radimo testuose duomenis tyčia parenkame "įtartinus" bei neatitinkančius normaliame sistemos darbe naudojamų duomenų.

Testavimo procesų tikslas

- Atestacijos testas
  - Pademonstruoti programuotojams ir užsakovams, kad PĮ atitinka jai iškeltus reikalavimus
  - Testas bus sėkmingas, jeigu parodysime, kad sistema veikia taip, kaip ir tikėjomės
- Defektų radimo testas
  - Reikia rasti klaidas ir defektus, rodančius, kad sistemos veikimas yra klaidingas arba neatitinka reikalavimų specifikacijos dokumento
  - Testas bus sėkmingas, jeigu pavyks priversti sistemą dirbti neteisingai ar ne pagal specifikaciją

Įvedimo-išvedimo duomenų modelis programos testavimui



Patikra ir atestavimas (Verification & Validation)

- Patikra (verification):
  "Ar mes gerai kuriame produktą"
- PĮ turi atitikti specifikaciją
- Atestavimas (validation):
  "Ar mes kuriame gerą produktą"
- PĮ turi daryti tą, ko iš tikrųjų reikia PĮ vartotojui

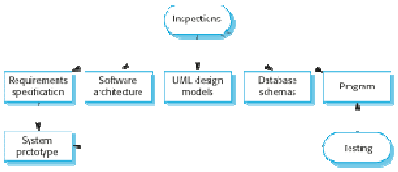
Patikros ir atestavimo svarbumo laipsnis

- ✧ P & A tikslas yra parodyti, kad sistema atitinka iškeltus tikslus
- ✧ Priklauso nuo sistemos tikslų, vartototjo lūkesčių bei verslo aplinkos
  - PĮ tikslas
    - Svarbumo laipsnis priklauso nuo to, kiek ta kuriama PĮ yra svarbi mūsų įmonei.
  - Vartotojo lūkesčiai
    - Vartotojai gali nedėti didelių vilčių kada nors sulaukti tam tikros rūšies programinės įrangos.
  - Verslo aplinka
    - Produkto atidavimas rinkai gali būti daug svarbiau, negu visų defektų radimas ir ištaisymas.

Peržiūros ir testavimas

- ✧ PĮ peržiūros susijusios su statine programos kodo peržiūra siekiant rasti klaidas kode (statinė patikra)
  - Gali būti naudojami tam tikri įrankiai ir procedūros
- ✧ PĮ testavimas atliekamas su pačia programa (dinaminė patikra)
  - Sistema tikrinama su iš anksto paruoštais testavimo duomenimis ir stebima, kaip sistema veikia.

Peržiūros ir testavimas



PĮ peržiūros

- ✧ Žmonės peržiūri PĮ kodą ir bando surasti anomalijas ir klaidas.
- ✧ Peržiūroms nereikia veikiančios programos, tik jos kodo, todėl galima atlikti dar prieš sukuriant pilnai veikiančią programą.
- ✧ Peržiūros gali būti atliekamos bet kuriai programos tadijai(reikalavimų dokumentas, projektavimo dokumentas, testavimo dokumentas ir t.t.).
- ✧ Įrodyta, kad tai labai efektyvi klaidų radimo technika.

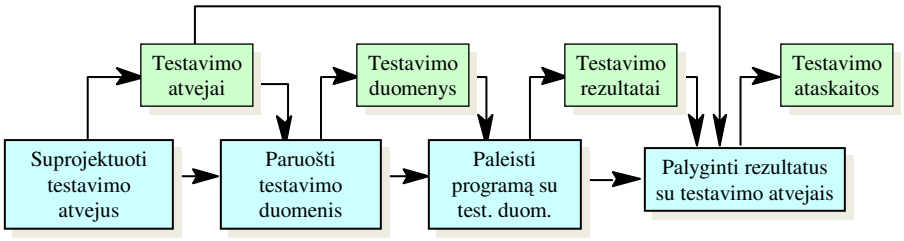
Peržiūrų privalumai

- Testuojant veikiančią programą, vienos klaidos gali paslėpti kitas klaidas. Kadangi peržiūra yra statinė, tai nėra jokio ryšio tarp dviejų klaidų ir jos viena kitos nepaslėps.
- Galima tikrinti dar nebaigtą programą be papildomų išlaidų. Jei programa nepilna, tai ruošiame testus tik toms programos dalims, kurios jau yra prieinamos.
- Be to, kad peržiūra padeda rasti klaidas, tai tuo pačiu galima patikrinti, ar programos kodas atitinka įmonės iškeltus programavimo standartus, galimybę panaudoti programos kodą pakartotinai bei būsimus palaikymo kaštus.

Peržiūros ir testavimas

- Peržiūra ir testavimas yra viena kitą paildančios patikros technikos.
- Patikros ir atestavimo procese būtina naudoti abi technikas.
- Peržiūros leidžia patikrinti PĮ atitikimą specifikacijai, bet neleidžia patikrinti, ar PĮ atitinka tikrus vartotojo reikalavimus.
- Peržiūros nepatikrina tokių nefunkcinių charakteristikų, kaip veikimo greitis, panaudojamumas ir pan.

PĮ testavimo modelis



PĮ testavimo etapai

- ✧ Kūrimo/programavimo testavimas, kurio metu sistema testuojama siekiant rasti klaidas ir defektus.
- ✧ Versijos (Release) testavimas, kurio metu nepriklausoma testavimo komanda testuoja pilnai užbaigtą sistemą prieš ją atiduodant galutiniam vartotojui.
- ✧ Vartotojo testavimas, kurio metu vartotojai (arba potencialūs vartotojai) testuoja sistemą savo darbinėje aplinkoje.

Kūrimo testavimas

- Apima visas testavimo veiklas, kurias atlieka programos kūrėjai (programuotojai).
  - Vieneto testavimas: atskirai testuojami programos moduliai, funkcijos ar kalsės. Vieneto testavimas fokusuojasi į testuojamų objektų funkcionalumo patikrinimą.
  - Komponentų testavimas: tikrinama, kaip veikia keli tarpusavyje tarpusavyje apjungti moduliai. Tikrinama, ar gerai veikia apjungtų komponentų sąsaja.
  - Sistemos testavimas: tikrinama, ar gerai veikia jau pilnai baigta sistema, t.y. Kai visi jos moduliai jau pilnai integruoti į sistemą. Testavimo metu akcentuojama apjungtų modulių tarpusavio sąveika.

Vieneto testavimas

- Vieneto testavimo proceso metu atskirai tikrinami individualūs programos komponentai.
- Tai yra defektų radimo procesas.
- Vienetais gali būti laikomi:
  - Atskiros funkcijos ir metodai
  - Objektų klasės, atributai ir metodai
  - Atskiri moduliai su apibrėžta sąsaja, leidžiančia prieiti prie komponento atliekamų funkcijų.

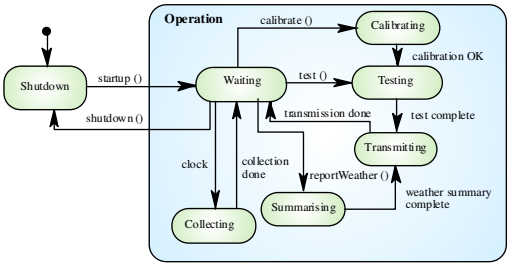
Objektų klasių testavimas

- Pilnas klasės testas apima
  - Visų su objektu susietų operacijų testavimą
  - Visų objekto atributų reikšmių pakeitimą ir patikrinimą
  - Visų galimų objekto būsenų keitimą ir tikrinimą
- Paveldimumas gali apsunkinti klasių testavimą, nes ne visa informacija mums yra prieinama.

Meteorologinės stoties objektas

WeatherStation
identifier
reportWeather () calibrate (instruments) test () startup (instruments) shutdown (instruments)

Meteorologinės stoties būsenų kaitos diagrama



Meteorologinės stoties objekto testavimas

- Testiniai atvejai reikalingi visoms operacijoms
- Naudojantis sistemos būvio modeliu nustatyti:
  - kuriuos būsenos pasikeitimus reikia testuoti
  - kokia įvykių seka sukelia tuos pasikeitimus
  - Testavimo sekų pavyzdžiai:

  Shutdown -> Waiting -> Shutdown

  Waiting -> Calibrating -> Testing -> Transmitting -> Waiting

Automatinis testavimas

- Visur, kur tik įmanoma, testavimą reikia automatizuoti.
- Vieneto testavimo automatizavimui galima naudoti testų automatizavimo karkasą (pvz. JUnit)
- Tokie vieneto testavimo automatizavimo karkasai sukuria testines klases, kurias jūs vėliau užpildote specialiu testavimui skirtu kodu. Tokiu būdu jūs iš karto dar tik rašydami klasę sukuriate jai testą, kurį galite leisti kiek tik norite kartų ir atlikti gautų klaidų analizę.

Automatinio testo komponentai

- Inicializacijos dalis, kur jūs nurodote įėjimo ir laukiamus išėjimo duomenis.
- Vykdymo dalis, kur iškviečiamas testuojamas objektas ar metodas.
- Lyginimo dalis, kur jūs lyginate gautus rezultatus su planuotais rezultatais. Jei jie sutampa, testas laikomas sėkmingai praėjusiu, jei ne – testas laikomas nepraėjusiu ir reikia ieškoti, dėl ko atsirado klaidos.

Vieneto testo efektyvumas

- Testo metu reikia parodyti, kad jei komponentas naudojamas taip, kaip tikimasi, tai ir jo duoti rezultatai yra tokie, kokių tikimasi.
- Jei komponentas turi klaidų, jos būtinai tūri būti surandamos mūsų paruoštais testais
- Galima išskirti du vieneto testavimo atvejų tipus:
  - Pirmasis turi atspindėti normalų sistemos legesį ir parodyti, kad testuojamas komponentas elgiasi taip, kaip ir tikimasi.
  - Antras testų tipas remiasi programuotojo patirtimi rasti tipines klaidas ir ruošiamas naudojant nenormalius įvedimo duomenis siekiant patikrinti, ar programa sugeba korektiškai apdoroti klaidas.

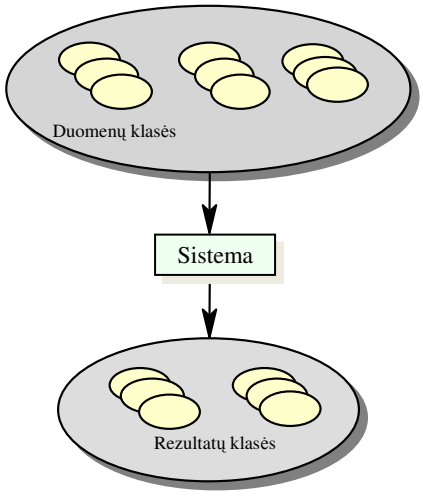
Testavimo strategijos

- Ekvivalentinis sudalinimas, kurio metu jūs nustatote tas pačias charakteristikas turinčių įvedimo duomenų grupes, kurias programa turėtų apdoroti taip pat.
  - Testus ruošiame kiekvienai tokių ekvivalentinių įvedimo duomenų grupei.
- Tam tikromis gairėmis paremtas testavimas, pagal kurį jūs ruošiate savo testinius atvejus.
  - Gairės remiasi ankstesne programuotojų patirtimi ir anksčiau padarytų klaidų sąrašu. Galima sudaryti tipinius klaidų sąrašus ir po to juos naudoti programos kodo tikrinimui.

Ekvivalentinis sudalinimas

- Įvedami duomenys ir išvedami rezultatai paskirstomi į atskiras klases, kuriose esantys nariai yra panašūs.
- Kiekviena iš šių klasių yra ekvivalentinio sudalinimo rezultatas, kur programos elgesys su kiekvienu klasės nariu yra toks pat (ekvivalentiškas).
- Testavimo atvejai ruošiami kiekvienai ekvivalentinio sudalijimo aibei.

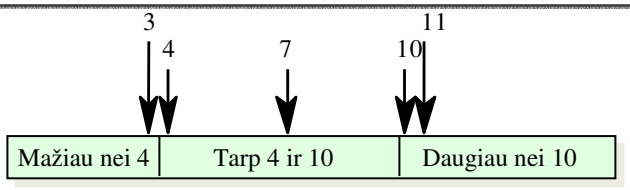
Ekvivalentinis sudalijimas



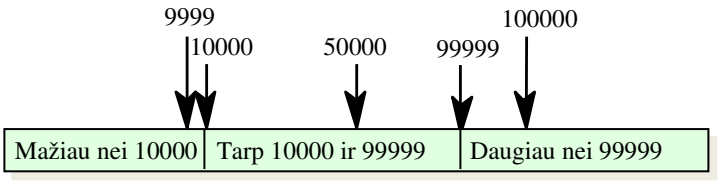
Ekvivalentinis sudalijimas ir ribinės reikšmės

- Padalinti sistemos įėjimų ir išėjimų reikšmes į “ekvivalentiškas aibes” ir iš jų paimti ribines.
- Jei įvedamas 5 skaitmenų sveikas skaičius tarp 10000 ir 99999, ekvivalentinio sudalinimo aibės yra <10000, 10000-99999 ir > 99999.
- Išrinkti aibėms testinius atvejus 09999, 10000, 99999, 100000.

Ekvivalentinis sudalijimas



Įvedamas 5-10 skaitmenų skaičius



Įvedamas 5 skaitmenų skaičius tarp 10000 ir 99999

Testavimo gairės testavimo duomenų sekoms

- ✧ Testuoti programą su seka, kuri turi tik vieną elementą
- ✧ Naudoti skirtingo dydžio duomenų sekas.
- ✧ Paruošti testus taip, kad būtų apdorojamas pirmas, paskutinis ir vidurinis testavimo sekos elementas.
- ✧ Testuoti su nulinio ilgio seka.

Bendros testavimo taisyklės

- ✧ Paruošti testavimo duomenis taip, kad būtų aporojami visi klaidų pranešimai
- ✧ Paruošti testavimo duomenis taip, kad visi fiksuoto ilgio masyvai persipildytų
- ✧ Kartoti testus su tais pačiais duomenimis keletą kartų
- ✧ Sugalvoti, kaip gauti blogus rezultatus
- ✧ Sugalvoti tokius testavimo duomenis, kad skaičiavimai vyktų labai ilgai arba labai trumpai.

Komponentų testavimas

- PĮ komponentai paprastai yra sudaryti iš kelių tarpusavyje sąveikaujančių objektų.
- Kiekvieno objekto funkcionalumas yra pasiekiams per tam tikrą to komponento sąsają.
- Tikrinant keletą komponentų reikėtų akcentuotis į tai, ar tie komponentai tarpusavyje sąveikauja pagal specifikaciją.
  - Daroma prielaida, kad į komponentą įeinatys vienetai jau pilnai ištestuoti.

Pavyzdys – paieškos procedūra

```
procedure Search (Key : ELEM ; T: ELEM_ARRAY;
      Found : in out BOOLEAN; L: in out ELEM_INDEX) ;

Pre-condition
      -- the array has at least one element
      T'FIRST <= T'LAST
Post-condition
      -- the element is found and is referenced by L
      ( Found and T (L) = Key)
or
      -- the element is not in the array
      ( not Found and
      not (exists i, T'FIRST >= i <= T'LAST, T (i) = Key ))
```

Paieškos procedūra – įvedimo duomenys

- ✧Duomenys, atitinkantys pradines sąlygas
- ✧Duomenys, neatitinkantys pradinių sąlygų
- ✧Duomenys, į kuriuos įeina raktinis elementas
- ✧Duomenys, į kuriuos neįeina raktinis elementas

Paieškos procedūra – rekomendacijos masyvams

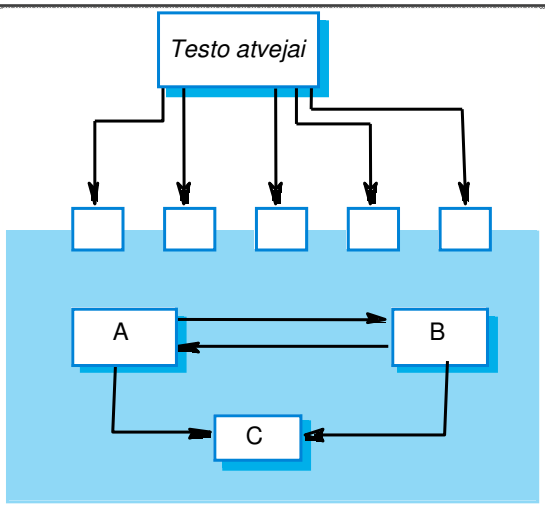
- ✧Testuoti programinę įrangą kai masyvas turi tik vieną elementą.
- ✧Skirtingiems testams naudoti skirtingo dydžio masyvus.
- ✧Parinkti testus taip, kad būtų nagrinėjamas pirmas, vidurinis ir paskutinis masyvo elementas.
- ✧Testuoti nulinio ilgio masyvus.

Testai paieškos procedūrai

Array	Element
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Input sequence (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Sąsajos testavimas



Sąsajos testavimas

✧ Sąsajų tipai

- Parametrų sąsajos: duomenys perduodami iš vienos procedūros ar metodo į kitą.
- Bendrai naudojamos atminties sąsaja: tam tikri atminties blokai yra bendri kelioms funkcijoms ar procedūroms..
- Procedūrinės sąsajos: tam tikros posistemės kviečia kitas procedūras, naudojamas dar kitose posistemėse.
- Žinučių perdavimo sąsajos: vienos posistemės prašo tam tikrų paslaugų iš kitos posistemės ir tam naudoja žinučių perdavimą

Sąsajos klaidos

✧Klaidingai naudojama sąsaja

- pakviestas komponentas iškviečia kitą komponentą ir daro klaidas jo naudojamoje sąsajoje, pvz.: neteisinga parametrų tvarka

✧Neteisingas sąsajos interpretavimas:

- kviečiantis komponentas remiasi prielaida apie kiekvieno komponento elgesį, kuris yra neteisingas.

✧Sinchronizacijos klaidos:

- pakviestas ir kviečiantysis komponentai dirba skirtingais greičiais ir nebegaliojanti informacija vis dar būna prieinama.

Sąsajos testavimo gairės

- Visada testuokite nuorodos parametrus su nuline rodykle.
- Sukurkite testus, kurie priverstų komponentą suklysti.
- Naudokite stresinį testavimą (stress) žinučių perdavimo sistemoje.
- Bendrai naudojamos atminties sistemose keiskite tvarką, kuria komponentai yra aktyvuojami.

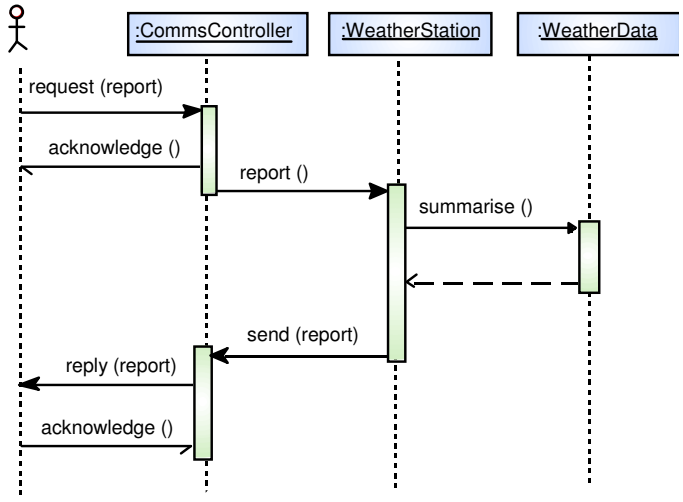
Sistemos testavimas

- Sistemos testavimo metu tikrinama iš atskirų komponentų sudaryta sistema.
- Tikrinama, kaip tie integruoti komponentai veikia tarpusavyje.
- Sistemos testavimo metu nustatoma, ar komponentai yra suderinami, ar jie sąveikauja korektiškai, ar jie atiduoda vienas kitam teisingus duomenis ir tada, kada jų reikia
- Sistemos testavimo testai turi patikrinti visus nenumatytus sistemos veikimo atvejus.

Vartotojo scenarijų testavimas

- Vartotojo scenarijai gali būti naudojami kaip pagrindas sistemos testavimui
- Kiekvienas vartotojo scenarijus paprastai apima keletą sistemos komponentų, todėl vartotojo scenarijų testavimas priverčia tuos komponentus suklysti, jei jie apjungti nekorektiškai.
- Sekos diagramos parodo, kokia seka reikėtų tikrinti sąveiką tarp atskirų komponentų

Meteorologinių duomenų surinkimo sekos diagrama



Testavimo politika

- ✧ Pilnas sistemos testavimas iš principo nėra įmanomas, todėl paprastai stengiamasi ruošti tokius testus, kad su kaip galima mažesniu testų kiekiu galėtume patikrinti kaip galiam didesnę programos dalį.
- ✧ Pavyzdžiai:
  - Patikrinti visas sitemos funkcijas, kurios yra pasiekiamos per meniu.
  - Funkcijų, kurios pasiekiamos per tuos pačius meniu, kombinacijos taip pat turi būti tikrinamos (pvz. format text)
  - Visas funkcijas, į kurias vartotojas paduoda savo duomenis, turi būti testuojamos tiek su teisingais, tiek su klaidingais duomenimis.

Į testus orientuotas programavimas

- ✧ Į testus orientuotas programavimas (Test-driven development (TDD)) yra programų kūrimo strategija, apjungianti ir programavimą, ir žemiausio lygio testavimą.
- ✧ Testai ruošiami dar prieš pradedant programavimo darbus.
- ✧ Jūs kuriate kodo gabaliuką ir testuojate pagal paruoštą testą, nesiimate programuoti kito kodo gabalo tol, kol jūsų šiuo metu kuriamas kodas nepraeina iš anksto paruošto testo.
- ✧ TDD yra ekstremalaus programavimo dalis.

Į testus orientuotas programavimas

Į testus orientuoto programavimo proceso veiklos

- ✧ Pirmiausia identifikuojame nedidelę funkcionalumo dalį, kurią turėsime suprogramuoti, o vėliau ir testuoti. Pvz. kelias kodo eilutes.
- ✧ Paruošiame testą toms kodo eilutėms ir įtraukiame į automatizuotus testus.
- ✧ Paleidžiame testą kartu su kitais anksčiau paruoštais testais. Kadangi pats programos kodas dar neparašytas, tai šis testas duos klaidą.
- ✧ Parašome programos kodą ir iš naujo paleidžiame testą.
- ✧ Jei testas praėjo, tai ruošiame testus kitoms funkcijoms programuoti ir testuoti, jei ne – ieškome klaidų.

Į testus orientuoto programavimo privalumai

- ✧ Kodo perdengimas
  - Kiekvienas kodo segmentas, kurį mes rašome, turi bent vieną su juos susietą testą, o visas mūsų parašytas kodas yra praktiškai pilnai pratestuotas.
- ✧ Regresinis testavimas
  - Periodiškai testas papildomas kartu su programos kūrimu – įdėjus naują kodo gabalą, kartu įdedamas ir jį atitinkantis testas.
- ✧ Supaprastintas debugging'as
  - Jei testas nepraeina, tai iš karto yra aišku, kurioje vietoje yra klaida – tikriname naujai parašytą kodą.
- ✧ Sistemos dokumentacija
  - Tokiu būdu paruošti testai kartu yra ir sistemos dokumentacija, parodanti, ką mūsų sistema turi daryti.

Regresinis testavimas

- ✧ Regresijos testas tikrina, ar nauji pakeitimai nesugadino anksčiau parašyto ir patikrinto programos kodo.
- ✧ Testuojant rankiniu būdu, regresinis testas yra brangus, bet jį automatizavus, jis yra paprastas ir labai veiksmingas. Visi testai atliekami po kiekvieno programos pakeitimo.
- ✧ Visi testai turi praeiti sėkmingai prieš patvirtinant kątik padarytus pakeitimus.

Versijos (Release) testavimas

- ✧ Versijos testavimas yra procesas, kurio metu tikrinama programa yra beveik paruošta išoriniam naudojimui.
- ✧ Pirminis versijos testavimo tikslas yra parodyti vartotojui, kad sistema yra pakankamai gera, kad galėtume ją pradėti naudoti.
  - Versijos testavimo metu parodome, kad sistema turi reikiamą funkcionalumą, veikimo greitį ir nelūžta atlikdama savo pagrindines funkcijas.
- ✧ Versijos testavimas yra juodos dėžės tipo, kai testus ruošiame tik pagal reikalavimų specifikaciją, o ne pagal programos logiką.

Versijos ir sistemos testavimas

- ✧ Versijos testavimas yra tam tikra prasme ir sistemos testavimas.
- ✧ Skirtumai:
  - Sistemos testavimą atlieka ne programavimo komanda, o visai su sistemos kūrimu nesusijusi komanda, pvz. testuotojai.
  - Sistemos testavimo metu programuotojų komanda siekia užtikrinti, kad programoje liktų kuo mažiau nesurastų klaidų. Versijos testavimo metu pagrindinis uždavinys yra parodyti, kad sistema atlieka savo funkcijas ir yra tinkama išoriniam vartojimui.

Reikalavimais paremtas testavimas

- Reikalavimais paremtas testavimas apima kiekvieno reikalavimo testavimą ir testų ruošimą tam reikalavimui.
- MHC-PMS reikalavimai:
  - Jei žinome, kad pacientas yra alergiškas tam tikrai medžiagai, tai tą medžiagą turinčių vaistų skyrimas turėtų vartotojui išmesti atskirą įspėjimą.
  - Jei vaistus skiriantis asmuo pranešimą ignoruoja, tai jis turi įvesti priežastį, kodėl jis taip daro.

Reikalavimų testas

- Suformuokite kliento įrašą be atžymų apie alergiją. Priskirkite jam visus galimus alergenus. Patikrinkite, ar sistema tikrai išmeta pranešimus kiekvienam alergenui.
- Suformuokite kliento įrašą su kokiu nors žinomu alergenu. Paskirkite tam klientui vaistų, turinčių tą alergeną. Patikrinkite, ar sistema išmeta pranešimą.
- Suformuokite kliento įrašą su bent dviem ar dugiau alergenų. Paskirkite jam vaistų iš pradžių su vienu, o po to su kitu alergenu, pažiūrėkite, ar sistema išmes pranešimus abiem atvejais.
- Sukurkite kliento įrašą su dviem alergenais, patikrinkite, ar sistema teisingai apdoroja abu alergenus.
- Klientui, turinčiam įrašą apie alergiją, skirkite vaistų, kurie turi tą medžaigą. Pranešimą ignoruokite, patikrinkite, ar sistema teisingai apdorojo paaiškinimą, kodėl klientuo buvo skirti tokie vaistai.

Savybės, tikrinamos remiantis vartotojo scenarijais

- ✧ Autorizacija jungiantis prie sistemos
- ✧ Tam tikrų duomenų atsisiuntimas arba įkėlimas į kompiuterį
- ✧ Paciento lankymo namuose tavrkaraščio sudarymas
- ✧ Kliento duomenų perkėlimas į mobilią priemonę
- ✧ Priėjimas prie įrašo ir jo modifikavimas
- ✧ Sąryšis su alergenų duomenų baze
- ✧ Kaip sistema apdoroja skambučius

Vartotojo scenarijus MHC-PMS sistemai

Kate is a nurse who specializes in mental health care. One of her responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side -effects.
On a day for home visits, Kate logs into the MHC-PMS and uses it to print her schedule of home visits for that day, along with summary information about the patients to be visited. She requests that the records for these patients be downloaded to her laptop. She is prompted for her key phrase to encrypt the records on the laptop.

One of the patients that she visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side -effect of keeping him awake at night. Kate looks up Jim's record and is prompted for her key phrase to decrypt the record. She checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so she notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. He agrees so Kate enters a prompt to call him when she gets back to the clinic to make an appointment with a physician. She ends the consultation and the system re-encrypts Jim's record.

After, finishing her consultations, Kate returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for Kate of those patients who she has to contact for follow-up information and make clinic appointments.

Našumo testavimas

- Versijos testavimo metu testuojamas sistemos našumas ir patikimumas.
- Testai turi atititkti realią vartotojo aplinką
- Našumas (greitis) paprastai testuojamas palaipsniui didinant sistemos apkrovimą tol, kol jos veikimas pasidaro nebepriimtinas.
- Sistemos testavimas virš jos maksimaliai suprojektuoto krūvio yra vadinamas stresiniu testavimu.

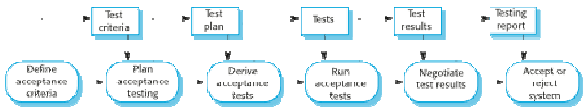
Vartotojo testavimas

- Vartotojo ar užsakovo testavimo metu sistemą tikrina jos galutiniai vartotojai.
- Vartotojo testavimas atliekamas tik tada, kai sistemos ir versijos testavimai yra pilnai baigti.

Vartotjo testavimo tipai

- ✧ Alpha testas
  - Vartotojai dirba kartu su programuotojais ir padeda testuoti programą iš programuotojų pozicijų.
- ✧ Beta testas
  - Beta versija leidžia vartotojams eksperimentuoti su sistema ir rasti klaidas, kurias pražiopsojo alfa testuotojai.
- ✧ Priėmimo testas
  - Vartotojai testuoja, ar sistema jau paruošta darbui ir ar tinkama kliento funkcijoms atlikti. Papastai šis testas aliekamas užsakomojo pobūdžio PĮ.

Priėmimo testo procesas



Priėmimo testo proceso žingsniai

- ✧ Priėmimo kriterijaus nustatymas
- ✧ Priėmimo testo planavimas
- ✧ Priėmimo testų ruošimas
- ✧ Priėmimo testų leidimas
- ✧ Derybos dėl gautų rezultatų
- ✧ Sistemos priėmimas ar atmetimas