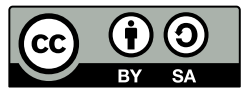
Programų testavimo scenarijai

Saulius Gražulis

2009 ruduo

Vilniaus universitetas, Matematikos ir informatikos fakultetas
Informatikos institutas



Testų naudojimo scenarijai

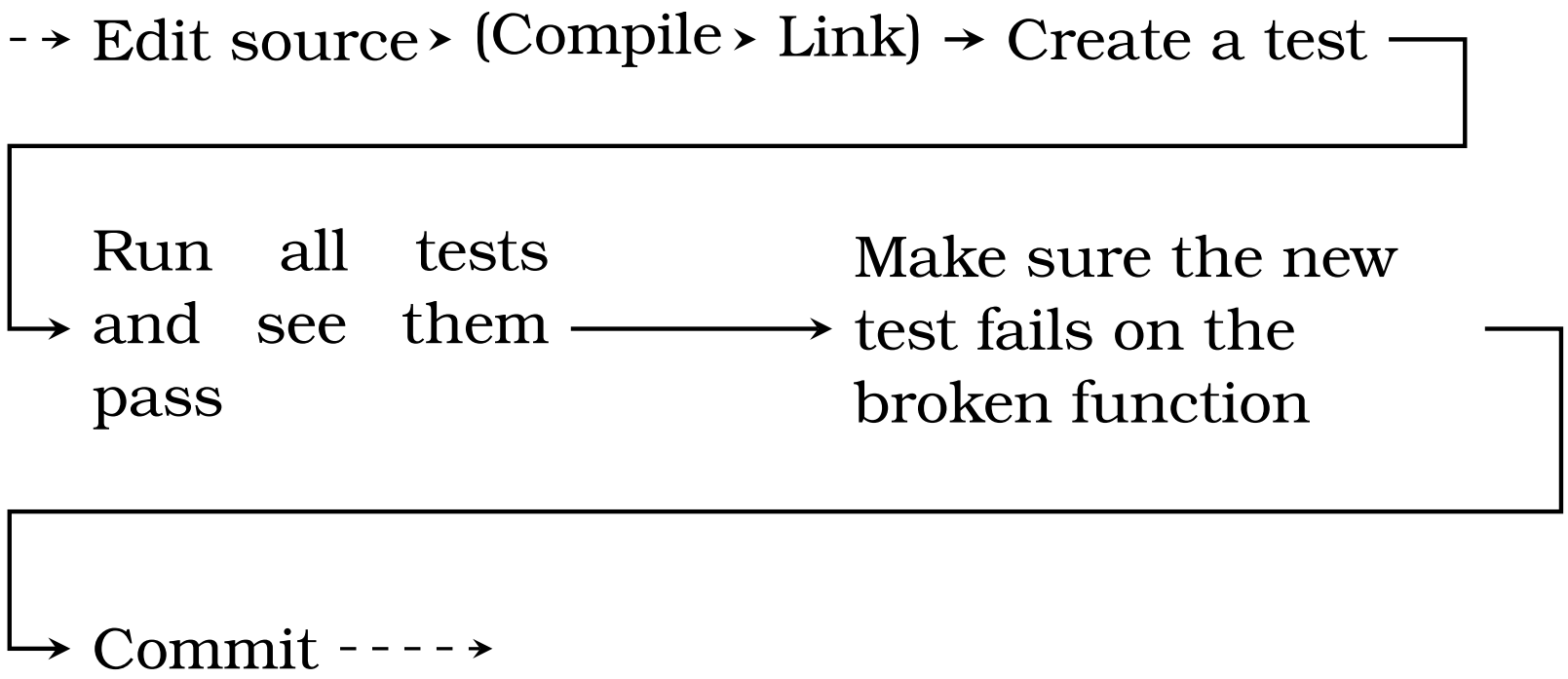
testus galime naudoti įvairiais būdais

- Jau įgyvendintų (realizuotų) funkcijų testavimas
  - sukuriame automatinį testą kiekvienai realizuotai funkcijai.
- Testais paremtas vystymas (Test driven development):
  - sukuriame testą ***prieš*** realizuodami naują funkciją;
- „Ekstremalus programavimas“ (Extreme programming, XP), „Lankstusis programavimas“ (Agile software development):
  - programa nuolatos automatiškai testuojama, kad ir po mažiausių pakeitimų (testų serveryje);
  - naudojame TDD; „ekstremalus“ požiūris – be testo programos pakeitimas į repozitoriją nepriimamas.

Paprasčiausias testavimo scenarijus

Kiekvinai realizuotai funkcijai parašome testą

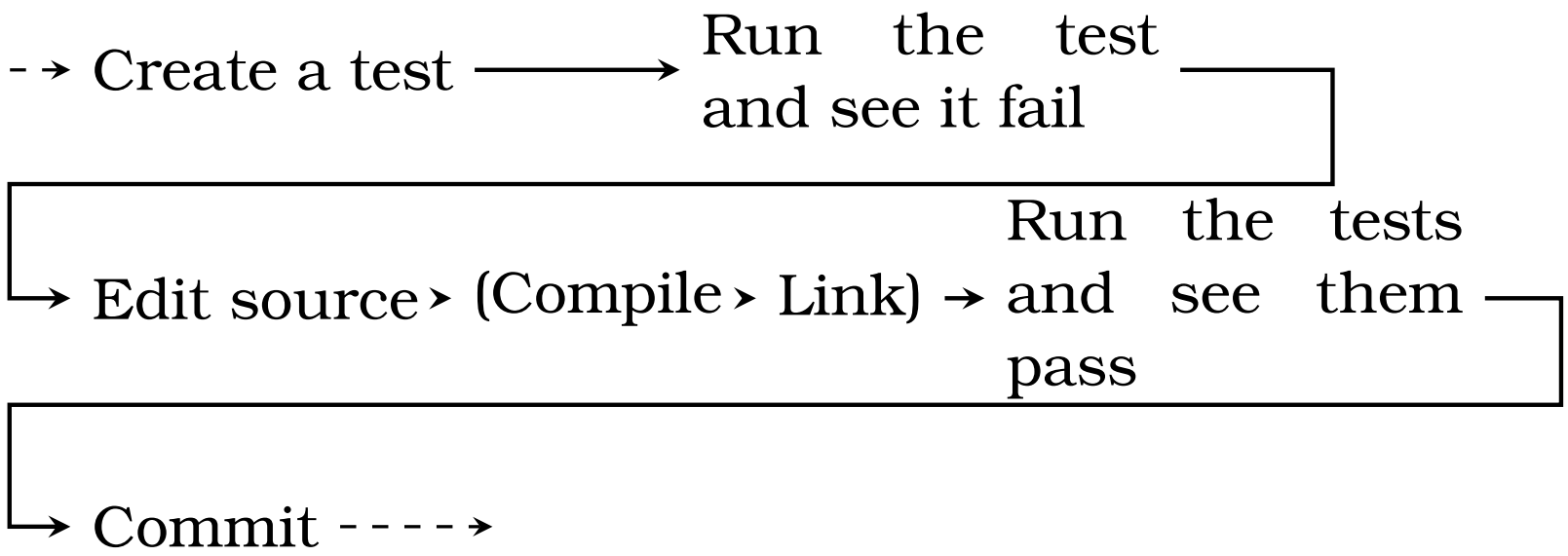
- Programos vystymo ciklas papildomas testo kūrimo ir testavimo operacijomis:



Testais paremtas vystymas (Test driven development)

Parašykime testą ***prieš*** rašydami kodą

- Programos vystymo ciklas papildomas testo kūrimo ir testavimo operacijomis:



Etaloninių rezultatų paruošimas

Kur gauti „teisingus“ programos rezultatų pavyzdžius?

- Suskaičiuoti „rankiniu“ būdu;
- privalumai:
  - rezultatas bus teisingas – juk mes žinome, ką darome :)
  - rezultato neįtakoja galimos klaidos testuojamoje programoje.
- trūkumai:
  - skaičiuodami galime privelti klaidų – klysti žmogiška...
  - sudetingi skaičiavimai gali būti neįgyvendinam (juk tam ir rašome programą, kad juos atlikti)

Etaloninių rezultatų paruošimas (2)

Kur gauti „teisingus“ programos rezultatų pavyzdžius?

- Paruošti kita programa, gal būt kitu algoritmu; paprastas (kad ir lėtas algoritmas) gali būti realizuojamas daug paprasčiau.
- privalumai:
  - rezultato neįtakoja galimos klaidos testuojamoje programoje.
  - rezultatas bus teisingas – juk sugeneruojame jį teisinga programa :)
  - rezultatas generuojamas automatiškai, galima sukurti daug testų
- trūkumai:
  - reikia sugalvoti ir realizuoti atskirą algoritmą tam pačiam uždaviniui spręsti;
  - *kas* mums sakė, kad testus generuojantis algoritmas yra teisingas? ...
  - lėtas testus generuojantis algoritmas gali pareikalauti daug laiko.

Etaloninių rezultatų paruošimas (3)

Kur gauti „teisingus“ programos rezultatų pavyzdžius?

- Sugeneruoti pačia testuojama programa... (!)
- privalumai:
  - testai įtvirtina (*teisingus?*) programos darbo pavyzdžius, todėl gerai tinka regresijos testams;
  - etaloninius rezultatus galime paruošti automatiškai (pvz. komanda `make outputs`)
- trūkumai:
  - būtina ***atidžiai patikrinti***, geriausiai formaliai *įrodyti*, kad testo rezultatas teisingas,
  - neteisinga programa duos neteisingus rezultatus (o ju programa dar tik testuojama...)
  - prieš generuojant testų rezultatus testuojama funkcija jau turi būti realizuota, todėl šis metodas (formaliai) netinka testais paremtam vystymui...

Kodo padengimas (padengtis?)

Gerai būtų, jei kiekviena programos eilutė būtų įvykdyta bent kartą...

- Kodo padengimas testais (code coverage) yra parametras, kuris parodo, kokią programos dalį iš teoriškai įmanomų testai patikrina
- Galime nagrinėti skirtingus kodo padengimo aspektus:
  - Funkcijų (paprogramių) padengimas
    - ar kiekvina paprogramė testuojant buvo pakviesta, ir kiek kartų?
  - Operatorių (eilučių, programos sakinių) padengimas
    - ar kiekvienas operatorius buvo aktyvuotas, ir kiek kartų?
  - Sąlygų padengimas, predikatų padengimas, sprendimų padengimas
    - ar kiekviena sąlyga buvo patikrinta abiem galimais atvejais (teisinga, klaidinga)?
  - Nuoseklių kodo fragmentų su perėjimais (Linear Code Sequence and Jump) padengimas, pagrindinių blokų (basic block)
    - ar kiekvienas nesišakojantis kodo fragmentas atidirbo bent kartą?

Pradiniai duomenys testams

- Geri tokie testai, kurie
  - padeda aptikti kuo daugiau klaidų
  - patikrina kuo daugiau programos funkcijų.
- Kad tai pasiekti, galima naudoti štai tokius metodus:
  - Duomenų ruožų analizė (Equivalence partitioning)
    - Išnagrinėjame, kokie duomenų ruožai svarbūs programos darbui, ir pasirenkame bent po vieną vertę iš kiekvieno ruožo.
  - Kraštutinių reikšmių analizė (Boundary value analysis)
    - Iš kiekvieno ekvivalentiško duomenų ruožo testuojame programos elgesį su ruožo kraštinėmis reikšmemis – viena prieš kraštą, su pačiu kraštu, viena po krašto.

Pradinių duomenų šaltiniai

- „Sintetiniai testai“
  - Sukuriami „dirbtiniai“ duomenys, kurie testuoja kokią nors programos savybę
  - Privalumai:
    - greitai įvykdomi,
    - testuoja visas savybes atskirai, todėl gali gerai lokalizuoti klaidas;
    - leidžia sukurti „egzotiškas“, retai sutinkamas sąlygas ir pagerinti padengimą testais.
  - Trūkumai:
    - Reikia didelio tokių testų kiekio;
    - Gali būti nepatikrintos svarbios sąveikos tarp sistemos dalių.

Pradinių duomenų šaltiniai (2)

- Atsitiktiniai duomenų srautai
  - Programai paduodamas atsitiktinis bitų srautas, arba atsitiktinis bet teisingai suformatuotas duomenų srautas
  - Privalumai:
    - leidžia patikrinti programos reakciją į nelauktas klaidas.
  - Trūkumai:
    - Reikia didelio tokių testų kiekio;
    - Negarantuojamas padengimas testais.

Pradinių duomenų šaltiniai (3)

- „Realaus gyvenimo“ testai
  - Imami tipiški, realiame gyvenime sutinkami duomenys
  - Privalumai:
    - Atspindi realias programos naudojimo situacijas; parodo, kiek programa tinka praktiniam naudojimui;
    - Patikrina programą visapusiškai, patikrina reliai reikalingas sąveikas tarp programos dalių.
  - Trūkumai:
    - gali ilgai užtrukti;
    - sunku užtinkrinti gerą padengimą testais.

Testų rūšys

- Regresijos testai (angl. *regression tests*):
  - tikriname, ar senos funkcijos vis dar veikia taip, kaip anksčiau,
  - tikriname, ar nepasireiškia senos programos klaidos
- Modulių testai (angl. *unit tests*):
  - tikriname, ar atskiros programos dalys (bibliotekos, moduliai, klasės, procedūros, metodai) veikia taip, kaip turi veikti.
- Sistemos testai (angl. *system tests*):
  - tikriname, ar surinkta pagrindinė programa veikia taip, kaip turi veikti, ir ar teisinga sąveika tarp jos dalių (modulių, klasių).
- Integracijos testai (angl. *integration tests*):
  - tikriname, kaip sistema veikia savo tikslinėje aplinkoje.
- Atsiskaitymo testai (angl. *acceptance tests*):
  - tikriname ir demonstruojame, ar programa atitinka specifikacijoje apibrėžtus reikalavimus.

Testavimo strategijos

Testavimo strategijos iš esmės atspindi programų kūrimo strategijas

- Iš viršaus žemyn (top-down)
  - pradžioje parašome pagrindinę programą; vietoj nerealizuotų funkcijų įdedame „tuščias“ paprogrames (objektus),
  - tetstuojame pagrindinę programą.
  - paruošiame po testą kiekvienai pridėtai funkcijai.
- Iš apačios į viršų (bottom-up)
  - rašome ir testuojame atskirus modulius; vietoj pagrindinės programos naudojame testinius „draiverius“
  - kai paruošta pakankama modulių dalis, apjungiame juos į bendrą programą ir paruošiame sisteminius testus.
- Kombinuotos

Egzotiški testavimo būdai

Dirbtinai įveskime klaidas į programą...

- Dirbtinis klaidų įvedimas („fault injection“)
  - Dirbtinai įvedame klaidas į programą;
  - testuojame ir kitais būdais ieškome klaidų;
  - žiūrime, koks dirbtinai įvestų klaidų procentas surastas...
- Klaidų modeliavimas
  - laikinai pakeičiame modulį tokiu, kuris žinomais atvejais generuoja išimtinę situaciją;
  - testuojame, kaip likusios programos dalys reaguos į tokias išimtines situacijas
- Stresas programai (stress testing)
  - testuojame programą su labai didleiu duomenų kiekiu, dirbtinai apribojė prieinamą atmintis kiekį, tinklo pralaidumą, su dirbtinai ilgais atsako laikais ir pan.

Programų testavimo standartai

- ISO 9126 – an international standard for the evaluation of software quality.
- DO-178B – Software Considerations in Airborne Systems and Equipment Certification.

Testavimo galimybių ribos

- Testavimas gali *įrodyti*, kad programa yra klaidinga...
- ...bet jis negali įrodyti, kad programa yra *teisinga*