Procedūrinio programavimo pagrindai

Testavimas

lekt. Irmantas Radavičius
irmantas.radavicius@mif.vu.lt
Informatikos institutas, MIF, VU

Turinys

Klaidos

Klaidos – neatsiejama programinės dalis

Pokyčiai → klaidos → pokyčiai → klaidos ...

Klaidų tipai:

- kompiliavimo
- vykdymo
- loginės

Klaidos aplikacijoje

Vartotojo sukeltos klaidos:

- galimų klaidų prognozė ir tinkamas apdorojimas
- vyksta vykdymo metu

Programuotojo klaidos:

- klaidų prevencija vs klaidų taisymas
- testavimas su vartotojais ?!
- vienetų (angl. unit) testai

Klaidų vengimo principai

Klaidų prevencija vs klaidų taisymas

Rekomendacijos:

- “švarus” kodas
- inicializacija
- validacija
- logika
- resursai

Operandų įvertinimas

Operandų įvertinimo eiliškumas (angl. order of evaluation)

```
a[i++] = i++ + i;
```

Operatoriai su fiksuota įvertinimo tvarka:

- ||
- &&
- ? :
- ,

Kitais atvejais įvertinimo tvarka neapibrėžta:

"The moral is that writing code that depends on order of evaluation is a bad programming practice in any language. Naturally, it is necessary to know what things to avoid, but if you don't know *how* they are done on various machines, that innocence may help to protect you. " (K&R, 1ed.)

Pašaliniai efektai (angl. side effects)

Kintamojo reikšmė kinta reiškinio įvertinimo procese:

```
a = b = c;
a = i++;
a = scanf(“%d”, &a);
if (a = 0) { … }
```

Sekos taškai (angl. sequence points)

Taškas programos vykdymo eigoje, kuriame:

- visi ankstesni “pašaliniai efektai” jau neturi įtakos
- visi vėlesni “pašaliniai efektai” dar neturi įtakos

Sekos taškai:

- viso reiškinio įvertinimo pabaiga
- operandai operatoriuose && || ?: ,
- prieš pat funkcijos iškvietimą (įvertinus iškvietimo parametrus)

Dvi kertinės taisyklės (C99)

Tarp dviejų sekos taškų kintamajame saugoma reikšmė gali būti pakeista ne daugiau nei vieną kartą.

```
c = a[i++] + b[i++];
c = a[i++] && b[i++];
```

Prieš tai buvusi kintamojo reikšmė turi būti naudojama tik naujai reikšmei nustatyti.

```
a[i] = i++;
a[i++] = i;
```

Klaidų paieška

Klaidų lokalizacija:

- stebimosios reikšmės (angl. watches)
- stabdos taškai (angl. breakpoints)
- tarpiniai išvedimai

Kodo peržiūra ir kokybės užtikrinimas

Automatiniai ir neautomatiniai testai:

- vienetų testavimas (angl. unit testing)
- integracijos testavimas (angl. integration testing)
- sistemos testavimas (angl. system testing)

Sistemos testavimas

El.pašto programa...

1. Prisijungimas.
2. Pašto dėžutės atidarymas.
3. Gautų laiškų peržiūra.
4. Laiško rašymas, persiuntimas ir atsakymas.
5. Išsiųstų laiškų peržiūra.
6. Pašto dėžutės uždarymas.
7. Atsijungimas.

Realiame pasaulyje sudėtingas, tinka kai atskiros dalys jau veikia.

Vienetų testavimas

Klaidų paieška ir taisymas vienetų testavimo etape kainuoja žymiai mažiau nei sistemos testavimo etape.

Vienetų testai:

- izoliuoja programos dalį
- specifikuoja ir tikrina konkrečią elgseną, kurios tikimasi
- aiškiai identifikuoja klaidos priežastį
- veikia greitai

Testavimo rezultatai:

- „praėjęs“ (angl. passed) testas (klaidos rasti nepavyko)
- „nepraėjęs“ (angl. failed) testas (sėkmingai rasta klaida)

Vienetų testai

Tikrinama elgsena vs vienas testas vienai funkcijai

Nejmanoma tikrinti visko. Ką pasirinkti klaidų aptikimui?

Testų variantai:

- kraštutinės sąlygos
- nekorektiški parametrai
- pakartotinis iškvietimas
- invariantų užtikrinimas
- klaidų pranešimai

Ir kt.

Testavimas realiame pasaulyje

Testai kuriami jau sukūrus kodą.

Per mažai automatizuojamas.

Laikomas neprioritetine ir/ar antrarūše ir/ar naujokų veikla.

Prisiminimui:

- top-down vs bottom-up

„Vaikščiojantis skeletas“

Architektūra

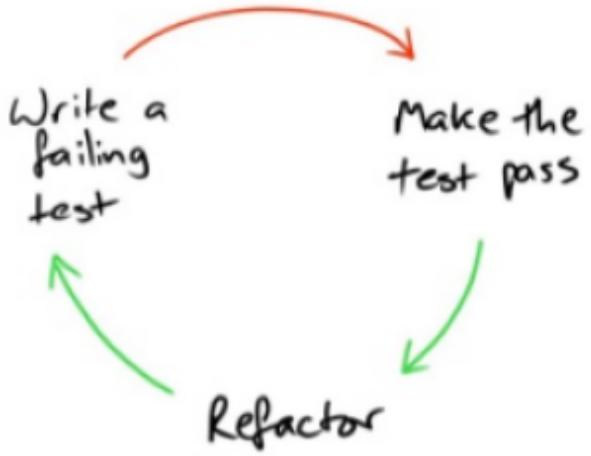
- automatizuotam kūrimui,
- testavimui ir
- pateikimui

```
BEGIN
    Temperature = ThermometerRead(Outside)
    IF Temperature > 40 THEN
         PRINT "It's HOT!"
    END IF
END
```

```
BEGIN ThermometerRead(Source insideOrOutside)
     RETURN 28
END ThermometerRead
```

Testavimu paremtas kūrimas/projektavimas

1. Testas kuriamas PRIEŠ rašant kodą.
2. Testas paleidžiamas PRIEŠ rašant kodą.
3. Testas PRIVALO nepraeiti.
4. Kodas rašomas, kad patenkinti testą.
5. Kodas rašomas TIK testams patenkinti.
6. Leidžiami visi testai.
7. VISI testai privalo praeiti.
8. Kodo pertvarkymas (angl. refactoring) ir kokybės užtikrinimas.
9. Užtikrinama, kad visi testai vis dar praeina.



Vykdoma (angl. executable) dokumentacija?

Assert

Funkcija assert yra skirta nurodyti sąlygoms,
kurias programuotojas mano esant visada teisingomis.

Neišpildyta sąlyga turėtų indikuoti klaidą programos kode.

```
int main(int argc, char *argv[]){
    assert(true);     // does nothing
    assert(false);    // always aborts
    system("pause"); // never reached
}
```

```
Assertion failed: false, file test.cpp, line 27
```

Assert panaudojimas

Tipinės panaudojimo vietos:

- pre-sąlygos
- post-sąlygos
- įvairūs invariantai

```
if (i % 3 == 0) {
    ...
} else if (i % 3 == 1) {
    ...
} else {
    assert (i % 3 == 2);
    ...
}
```

Turi pranešti apie klaidą kode!

Nenaudoti įvedimo tikrinimui.

Vengti pašalinių efektų.

```
assert (file = fopen("failas.txt", "r");
```

```
for (...){
   if (...)
       return;
}
// should be unreachable
assert (false);
```

#define NDEBUG

```
#define assert(ignore)((void) 0)
```

assert(end)