22. Verifikáció és validáció

Kérdések

- Mi a szoftver verifikáció és validáció, mi a különbség köztük?
- Mi a program-vizsgálati eljárás, mi a szerepe a verifikációban és validációban?
- Mi a statikus analízis, hogyan használható ez, mint verifikációs technika?
- Mi a *Cleanroom* szoftverfejlesztési eljárás?

Tartalom

- A verifikáció és validáció tervezése
- Szoftver vizsgálatok
- Automatizált statikus analízis
- Cleanroom szoftverfejlesztés

A verifikáció és a validáció

- **Verifikáció:**
  „Jó minőségű terméket fejlesztünk?”
  (*jól fejlesztünk*?)
- A szoftver teljesítse a specifikációt.
- **Validáció:**
  „A megfelelő terméket fejlesztjük?”
  (*jót fejlesztünk*?)
- A szoftver azt csinálja, amit a felhasználó tényleg akar.

A V & V eljárás

- Az egész életciklusra jellemző. A V & V a szoftver fejlesztés minden állomásán alkalmazni kell.
- Két fő cél:
  - A rendszerbeli hibák felfedezése;
  - Annak felmérése, hogy a rendszer hasznos-e és használható-e a felhasználási környezetben.

A V & V célja

- A verifikáció és validáció a szoftver iránti bizalmi alapot teremt: a szoftver el tudja látni a feladatát.
- NEM jelenti, hogy teljesen hibamentes.
- Azt jelenti, hogy *elég jó* ahhoz, hogy ellássa feladatát.
- A feladat típusa határozza meg, milyen mértékű bizalom kell.

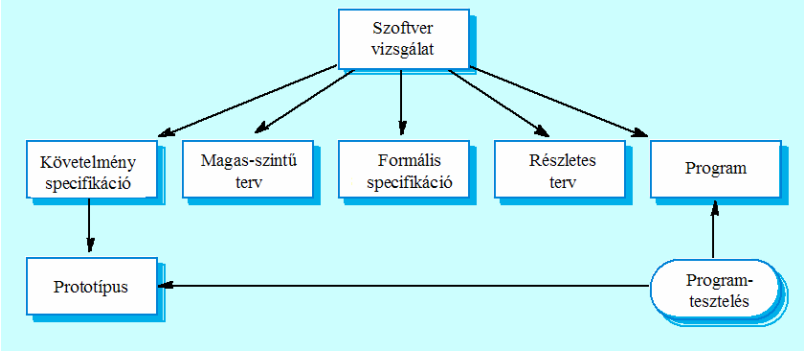
A bizalmi szint

- A rendszer céljától, a felhasználók elvárásaitól, valamint a piaci viszonyoktól függ.
  - A szoftver célja
    - A bizalmi szint függ attól, hogy mennyire kritikus a szoftver a szervezet számára.
  - Felhasználó elvárások
    - A felhasználóknak bizonyos szoftverekkel szemben nagyon alacsony elvárásaik vannak.
  - Piaci környezet
    - A gyors piacra dobás fontosabb lehet, mint a hibák megtalálása.

Statikus és dinamikus verifikáció

- Szoftver vizsgálatok. Problémák feltárása a rendszer statikus reprezentációjának analízise segítségével (statikus verifikáció).
  - Kiegészíthető eszköz-alapú dokumentum- és kód-analízissel.
- Szoftver tesztelés. Kísérletezés és a termék viselkedésének megfigyelése (dinamikus verifikáció)
  - A rendszert teszt-adatokkal futtatva működés közben figyeljük a viselkedését.

Statikus és dinamikus V&V



Programtesztelés

- A hibák jelenlétét és NEM hiányát jelezheti.
- Az egyetlen validációs technika nem-funkcionális követelmények ellenőrzésére, hiszen a szoftvert végre kell hajtani ahhoz, hogy lássuk, miként viselkedik.
- Statikus verifikációval együtt célszerű használni, hogy teljes V&V lefedettséget kapjunk.

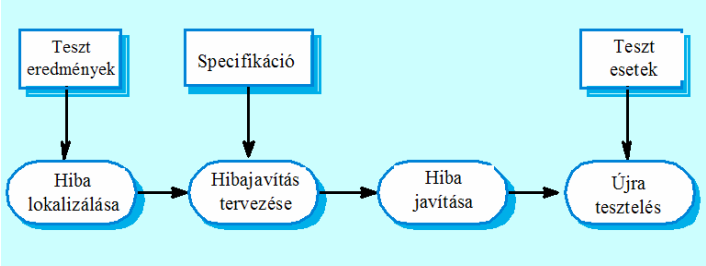
A tesztelés típusai

- Hibatesztelés
  - A tesztek rendszerhibák feltárására.
  - A jó teszt feltárja a rendszerben lévő hibák jelenlétét.
- Validációs tesztelés
  - Célja annak bizonyítása, hogy a szoftver teljesíti a követelményeket.
  - A jó teszt megmutatja, hogy a rendszert helyesen implementálták.

Tesztelés és hibakeresés

- A hibatesztelés és a hibakeresés különböző eljárások.
- A verifikáció és validáció feladata a programhibák jelenlétének feltárása.
- A hibakeresés ezen hibák lokalizálásával és javításával foglalkozik.
- A hibák megtalálása érdekében a hibakeresés során a program viselkedéséről hipotéziseket állítunk fel, amiket ellenőrzünk.

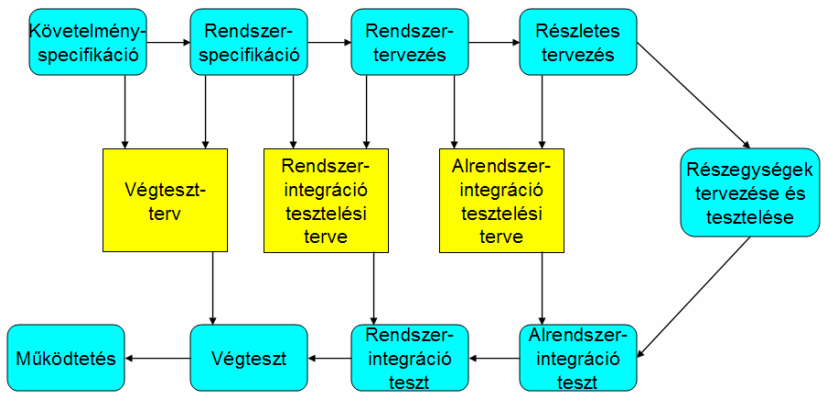
A hibakeresés folyamata



A V & V tervezése

- A tesztelési és vizsgálati eljárások sikere érdekében körültekintő tervezésre van szükség.
- A tervezést már a fejlesztés korai fázisában el kell kezdeni.
- A terv határozza meg a statikus verifikálás és a tesztelés helyes egyensúlyát.
- A teszt tervezése a tesztelési eljárás irányelveit fogalmazza meg, nem kell a termék tesztelését itt leírni.

A fejlesztés V-modellje



A szoftvertesztelési terv struktúrája

- A tesztelő eljárás.
- Követelmények követhetősége.
- Tesztelt elemek.
- A tesztelés menetrendje.
- A tesztek rögzítésének eljárása.
- Hardver és szoftver szükségletek.
- Kényszerek.

A szoftvertesztelési terv

A tesztelő eljárás.
A tesztelési eljárás főbb fázisainak leírása.

Követelmények követhetősége.
A teszteket úgy kell megtervezni, hogy minden követelményt külön lehessen tesztelni.

Tesztelt elemek.
A fejlesztési eljárás azon elemeit specifikáljuk, amelyeket tesztelni kell.

A tesztelés menetrendje.
A tesztelés menetrendje a szükséges erőforrások foglalásával. Természetesen szorosan kapcsolódik a teljes projekt ütemezéséhez.

A tesztek rögzítésének eljárása.
A teszteket nemcsak futtatni kell, hanem az eredményeket szisztematikusan rögzíteni is. A tesztelési eljárásnak felülvizsgálhatónak kell lenni.

Hardver és szoftver szükségletek.
A szükséges szoftver eszközök listája a becsült hardver használattal együtt.

Kényszerek.
A tesztelési eljárást befolyásoló kényszerek, pl. munkaerő hiány.

Szoftver vizsgálatok

- Emberek vizsgálják a forrás valamilyen reprezentációját anomáliák és hibák után kutatva.
- A vizsgálathoz nem kell a rendszert futtatni, így implementáció előtt is megtehető.
- A rendszer bármely reprezentációja vizsgálható: követelmények, terv, konfigurációs adatok, teszt adatok, stb.
- A programhibák feltárásának hatékony eszköze.

A vizsgálat hatékonysága

- Egyetlen vizsgálat több hibát is feltárhat. A tesztelés során egy hiba elfedhet más hibákat, így ott többszöri végrehajtás kell.
- Az újrafelhasználás és a programozói tapasztalat miatt a felülvizsgálók valószínűleg találkoztak már a gyakran előforduló hibákkal.

Vizsgálat és tesztelés

- A vizsgálatok és a tesztelés egymást kiegészítő verifikációs technikák.
- A V & V eljárás alatt mindkettő használata ajánlatos.
- A vizsgálat ellenőrzi, hogy a specifikációnak megfelel-e, de azt nem, hogy a valós felhasználói igényeket kielégíti-e.
- A vizsgálatok nem tudják ellenőrizni a nem-funkcionális jellemzőket, pl. teljesítmény, használhatóság, stb.

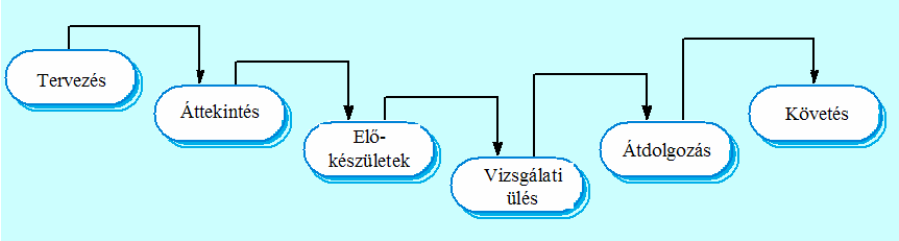
Program vizsgálatok

- A dokumentáció felülvizsgálatának formális módszere
- Célja kizárólag a hibák jelenlétének *jelzése*, (nem pedig javítása).
- A hibák lehetnek logikai hibák, anomáliák a kódban, amik hibás állapotot jelezhetnek (pl. nem inicializált változó), vagy egyes szabványok nem teljesítése.

A vizsgálat előfeltételei

- Precíz, teljes specifikáció.
- A csoport tagjainak ismerni kell a szervezet működési szabályait.
- Szintaktikailag helyes kód, vagy valamilyen más rendszer-reprezentáció.
- Egy hiba-ellenőrző listát kell készíteni.
- A menedzsmentnek el kell fogadni, hogy a szoftver vizsgálat növeli a költségeket.
- A menedzsment ne használja a szoftver vizsgálatot a dolgozók értékelésére (pl. ki hibázott).

A vizsgálat folyamata



- A rendszer ismertetése a felülvizsgáló csoport számára.
- A csoport tagjai megkapják a kódot és az egyéb kapcsolódó dokumentumokat.
- A vizsgálat megtörténik, a felfedezett hibákat feljegyzik.
- A felfedezett hibák javítása érdekében a szükséges módosítások elvégzése.
- Újabb vizsgálat lehet szükséges vagy szükségtelen.

Szerepek a vizsgálat során

Szerző vagy tulajdonos	A programozó vagy tervező, aki a program vagy egyéb dokumentum létrehozásáért felelős. Az ő felelőssége a vizsgálat során feltárt hibák javítása is.
Vizsgáló	Hibák, hiányosságok, inkonzisztenciák keresése a programokban és dokumentációban. Esetleg a vizsgáló bizottság feladatkörén kívül álló kérdéseket is felvethet.
Felolvasó	A vizsgálati ülésen bemutatja a kódot vagy dokumentumot.
Írnok	A vizsgálati ülés eredményeit rögzíti.
Elnök vagy moderátor	Menedzseli és segíti a vizsgálat menetét. Az eredményeket a fő moderátornak jelenti.
Fő moderátor	A vizsgálati folyamat javításáért, a hiba-ellenőrző lista frissítéséért, eszközökért, stb. felelős.

Ellenőrző listák

- A gyakori hibákat tartalmazó ellenőrző lista használandó a vizsgálat levezetésére.
- A hiba-ellenőrző listák programnyelv-specifikusak és az adott programnyelv karakterisztikus hibáit tartalmazzák.
- Általában minél gyengébb a típus-ellenőrzés, annál hosszabb az ellenőrző lista.
- Példák: inicializálás, konstansok elnevezése, kilépés hurokból, tömbhatár túllépés, stb.

Ellenőrző lista

Adathibák	Minden változó inicializálva van, mielőtt használnánk? Az összes konstansnak van neve? A tömbök felső indexe a tömb méretével egyenlő vagy ennél eggyel kisebb kell legyen? Karakter-tömbök használata esetén a delimiter egyértelműen definiálva van? Előfordulhat-e buffer overflow?
Vezérlési hibák	Minden feltételes utasításra: helyes a feltétel? Minden ciklus biztosan befejeződik? Az utasítás-blokkokat helyesen zárójeleztük? Case utasításnál minden lehetőséget kimerítettünk? Ha minden case utasítás után break kell, akkor ezek jelen vannak?
I/O hibák	Minden bemenő változót használunk? Minden kimeneti változónak adunk értéket visszatérés előtt? Váratlan bementi adatok okozhatnak-e hibát?
Interfész hibák	Minden függvény- és metódus-hívásnak megfelelő számú paramétere van? A paraméter-típusok megfelelők? A paraméterek sorrendje megfelelő? Ha több komponens osztott memóriát használ, akkor ugyanolyan struktúrájú memória-modellt használnak?
Tárolás-menedzsment hibák	Láncolt szerkezetek módosítása esetén minden mutató megfelelően módosítva van? Dinamikus memóriahasználat esetén helyes-e az allokáció? A nem használt memória explicit módon fel van-e szabadítva?
Kivétel-kezelési hibák	Minden lehetséges hibalehetőség figyelembe lett véve?

A vizsgálat sebessége

- 500 utasítás/óra az áttekintés során.
- 125 forrás utasítás/óra az egyéni előkészületek alatt.
- 90-125 utasítás/óra vizsgálható az ülésen.
- A vizsgálat drága!
- Pl.: 500 sor megvizsgálása kb. 40 emberóra igényű, ami kb. £2800 (1MFt).

Automatikus statikus analízis

- A statikus analizátorok a forrás kódok feldolgozására szolgáló szoftver eszközök.
- A program szövegének elemzésével potenciális hibalehetőségek felfedezésére szolgálnak, amelyeket a V & V csoporttal tudatnak.
- Hasznos segédeszközök a vizsgálathoz – csak kiegészítő eszközök, nem helyettesítik a vizsgálatot.

Statikus analízis ellenőrző lista

Hiba-osztály	Statikus analízis vizsgálat
Adat-hiba	Változók inicializálása használatuk előtt
	Deklarált, de nem használt változó
	Két értékadás közben nem használt változó
	Lehetséges tömbhatár-túllépés
	Nem deklarált változók
Vezérlési hiba	Nem elérhető kódrészlet
	Ugrás hurokba
I/O hiba	Kimeneti változónak két kimenet nincs értékadása
Interfész hiba	Paraméter típus konfliktus
	Paraméterek száma nem megfelelő
	Nem használt függvény-értékek
	Nem hívott függvények és eljárások
Tárolás-menedzsment hiba	Mutató értékadás hiánya
	Mutató aritmetika

A statikus analízis lépései

- Vezérlés analízis. Hurkok többszörös belépési vagy kilépési pontokkal, nem elérhető kód, stb.
- Adathasználat analízis. Nem inicializált változók, többször írt változók közbülső értékadás nélkül, deklarált, de nem használt változók, stb.
- Interfész analízis. Konzisztens eljárás-deklaráció és használat.

A statikus analízis lépései

- Információ-folyam analízis. A kimenő változók függőségeinek feltárása. Önmagában nem tud anomáliákat detektálni, de kijelöl kódrészleteket a vizsgálat céljára.
- Útvonal analízis. Útvonalakat keres a program végrehajtása során és felsorolja a végrehajtott utasításokat. Hasznos lehet a vizsgálat során.
- Ez a két lépés rengeteg információt generál, óvatosan használandók.

LINT statikus analízis

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
 int Anarray;
{   printf("%d",Anarray);  }

main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(11)
printf returns value which is always ignored
```

A statikus analízis felhasználása

- Nagyon hasznos olyan nyelveknél, ahol a típusellenőrzés gyenge és így a fordító sok hibát nem tud észlelni (pl. C).
- Kevésbé hasznos erős típusellenőrzéssel ellátott nyelvek esetén, ahol sok hiba fordítás közben kiderül (pl. Java).

A verifikáció és formális módszerek

- Formális módszerek alkalmazhatók, ha a rendszer matematikai modellje adott.
- Ez az alapvető statikus analízis technika.
- A specifikáció matematikai analízise, formális indoklás: a program megfelel a matematikai specifikációnak.

Formális módszerek előnyei

- A matematikai specifikáció elkészítéséhez a követelmények részletes elemzése szükséges, ami valószínűleg felfedi a hibákat.
- Implementációs hibákat még a tesztelés előtt fel tud fedni a program és a specifikáció együttes vizsgálatával.

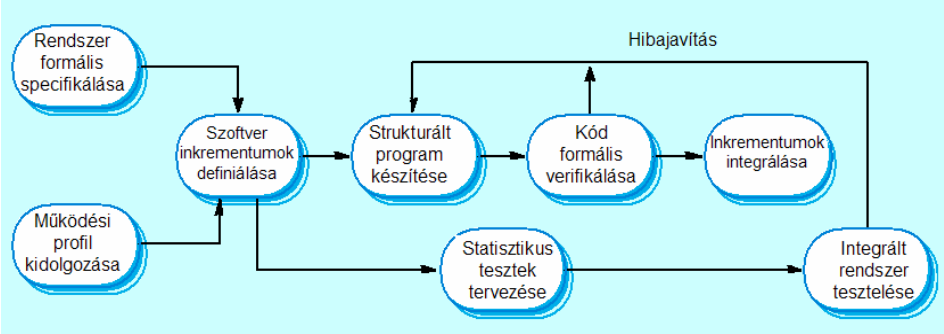
A formális módszerek hátrányai

- Speciális jelölésrendszer használata szükséges, amelyet az alkalmazási környezet szakértői nem értenek.
- A specifikáció kidolgozása nagyon drága. Még drágább bizonyítani, hogy a program megfelel a specifikációnak.
- Más V & V módszerek alkalmazásával is el lehet jutni ugyanolyan bizalmi szintre.

A Cleanroom szoftverfejlesztés

- A név a félvezető-gyártásban használt 'Cleanroom‘ eljárásból ered. Filozófia: a hibák elkerülése, nem a hibák eltávolítása.
- A következő elveken alapul:
  - Inkrementális fejlesztés;
  - Formális specifikáció;
  - Statikus verifikáció helyességbizonyítással;
  - Statisztikus tesztelés a program megbízhatóságának meghatározásához.

A Cleanroom eljárás



A Cleanroom eljárás jellemzői

- Formális specifikáció állapot-átmeneti modellekkel.
- Inkrementális programfejlesztés a megrendelő prioritásai szerint.
- Strukturált programozás – korlátozott vezérlési és absztrakciós eszközök alkalmazása.
- Statikus verifikáció szigorú vizsgálatokkal.
- Statisztikus tesztelés.

Formális specifikáció és vizsgálatok

- Az állapotgép egy rendszerspecifikáció. Az ellenőrző eljárás ezzel a modellel veti össze a programot.
- A programozási modell olyan, hogy a modell és a rendszer közötti kapcsolat világos.
- Matematikai állítások növelik a vizsgálati eljárás bizalmi szintjét.

A Cleanroom eljárásban részt vevő csoportok

- Specifikációs csoport. A rendszer-specifikáció kidolgozásáért és karbantartásáért felel.
- Fejlesztési csoport. A szoftver fejlesztéséért és verifikálásáért felelős. A szoftvert ebben a fázisban nem futtatják, még le sem fordítják.
- Tanúsító csoport. A fejlesztés után a szoftveren végrehajtandó statisztikus tesztek kidolgozásáért felelős.

A Cleanroom eljárás értékelése

- A Cleanroom eljárással készül rendszerekben nagyon kevés hiba van.
- Nem drágább, mint más módszerek.
- Kevesebb hiba, mint a „hagyományos” fejlesztési eljárásoknál.
- Ennek ellenére nem használt széles körben. Kérdés: hogyan lehet olyan környezetbe átültetni, hol kevésbé képzett és motivált szoftvermérnökök dolgoznak?

Összefoglalás

- A verifikáció és validáció nem ugyanazt jelenti. A verifikáció a specifikáció teljesítését mutatja, a validáció pedig azt, hogy a program kielégíti a felhasználó igényeit.
- A tesztelési eljárást tesztelési tervek segítik, ezek elkészítése szükséges.
- A statikus verifikációs technikák hibadetektálás céljából vizsgálják és analizálják a programot.
- A hibák felderítésének nagyon hasznos módja a programok vizsgálta.
- A vizsgálat során hibakeresés céljából a programkódot egy kis létszámú csoport szisztematikusan átvizsgálja.
- A statikus analízis eszközök esetleges programhibákat jelző anomáliákat keresnek a kódban.
- A Cleanroom fejlesztési eljárás elemei a inkrementális fejlesztés, statikus verifikáció és a statisztikus tesztelés.