# 12 Verifikácia a validácia

– procesy na zistenie, že softvér zodpovedá špecifikácii a spĺňa požiadavky používateľa

☞ *verifikácia: "Vytvárame výrobok správne?"*

☞ *validácia: "Vytvárame správny výrobok?"*

## Sledované vlastnosti

– SPRÁVNOSŤ: miera splnenia špecifikácie výrobku

– POUŽITEĽNOSŤ: miera splnenia potrieb používateľa

– ROBUSTNOSŤ: miera odolnosti voči nepriaznivým vplyvom okolitého prostredia (zadanie nepovolených vstupov, zmena operačného prostredia, zlyhanie časti systému, …)

– SPOĽAHLIVOSŤ: miera frekvencie a kritickosti zlyhania prevádzky výrobku (sleduje sa stredná doba poruchy a doba opravy následkom poruchy)

– VÝKONNOSŤ: definuje sa pomocou viacerých ukazovateľov: doba odozvy, pamäťové nároky, …

– PRENOSNOSŤ: miera prispôsobivosti systému rôznym operačným prostrediam

– BEZPEČNOSŤ: miera odolnosti voči neoprávneným zásahom do systému

…. (pozri vlastnosti softvérových výrobkov)

☞ *Správnosť výrobku nepostačuje!*

☞ *Dokonca správnosť niekedy nie je nevyhnutná!*

**Príklad:**

*Špecifikácia* procedúry SORT:

Vstupná podmienka A: array(1..N) of INTEGER

Výstupná podmienka B: array(1..N) of INTEGER, pričom $B(1) \le B(2) \le \ldots \le B(N)$

*Implementácia*

```
procedure SORT
   begin
        B := 0
   end   {SORT}
```

**!** V špecifikácii chýba:

*...a prvky poľa B sú permutáciou prvkov poľa A*

**Testovanie:** proces odvodenia určitých vlastností výrobku na základe výsledkov použitia, prevádzky výrobku (vykonania softvérového systému) v známom prostredí s vybranými vstupmi.

**Cieľ verifikácie a validácie (V & V)**

– odhaliť chyby počas vývoja: *test, ktorý neodhalí nesprávne správanie sa systému je neúspešný*

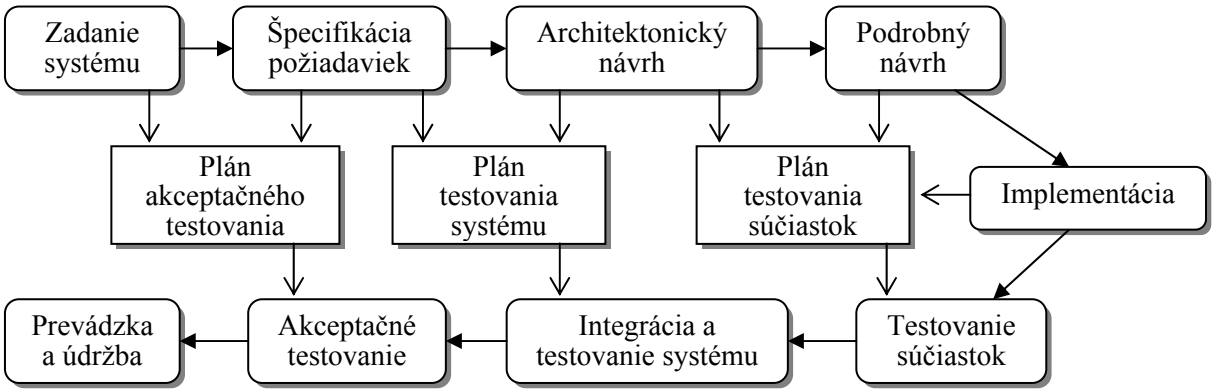
– preukázať požadované vlastnosti

Dijkstra: "Testovanie nemôže preukázať, že v programe nie sú chyby. Môže iba ukázať, že tam chyby sú!"

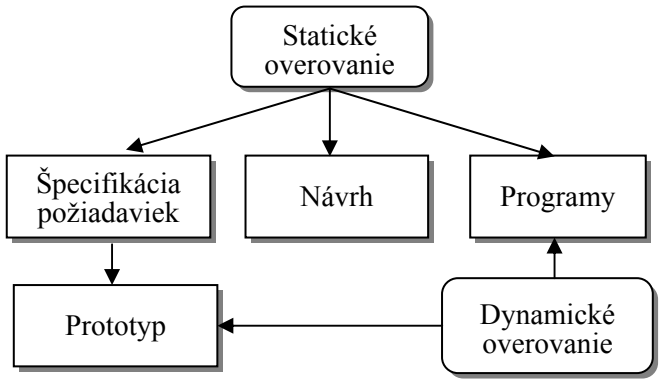
Murphy: "Ak systém môže 'spadnúť", spadne – a to v najhoršom (najnevhodnejšom) možnom okamihu."

**V & V**
- etapa životného cyklu
  - úroveň
    - modul
    - subsystém
    - systém
- techniky
  - statické
  - dynamické
- stratégie
  - zhora-nadol
  - zdola-nahor
  - jednofázové
  - sendvičové
  - porovnávaním
  - rizikové

**Plán testovania**

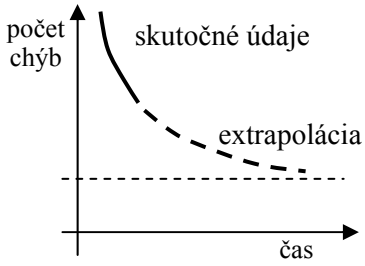


**Statické a dynamické overovanie**




**Predpovedanie počtu chýb**

Pred testovaním sa vložia „umelé chyby“. Potom sa porovnáva pomer nájdených skutočných a umelých (vložených) chýb.




Problém nových chýb pri oprave existujúcich.

# Techniky verifikácie a validácie

## Dynamické overovanie (testovanie)

- veľkosť testovacej množiny musí byť “prijateľná”
- testovací vstup sa vyberá na základe *testovacieho kritéria* (určuje podmienky, ktoré musia spĺňať množiny testovacích vstupov, napr. pokrytie všetkých príkazov v programe)
- rôzne testovacie vstupy by mali odhaliť rôzne chyby
- mali by sa odhaliť “všetky” chyby (nedostatky)


**Vlastnosti testovacieho kritéria:**

- *spoľahlivosť:* kritérium *K* je spoľahlivé, ak všetky množiny testovacích vstupov, ktoré spĺňajú kritérium *K* odhalia tie isté chyby, t.j. nezáleží na tom, ktorá množina testovacích vstupov sa vyberie, vždy odhalíme tie isté chyby
- *platnosť:* kritérium *K* je platné, ak pre každú chybu v programe existuje množina testovacích vstupov, ktorá spĺňa kritérium *K* a ktorá odhalí chybu

Ak je testovacie kritérium spoľahlivé a platné a množina testovacích vstupov, ktorá spĺňa kritérium uspeje (neodhalí žiadne chyby), tak program *neobsahuje chyby*.

ALE

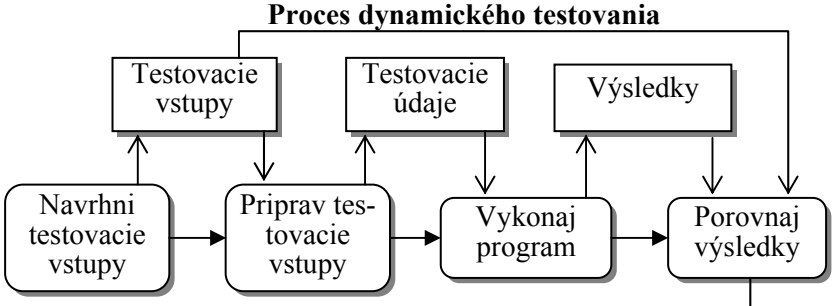
Bolo ukázané, že neexistuje algoritmus, ktorý určí platné kritérium pre ľubovoľný program.

**Prístupy dynamického testovania:**

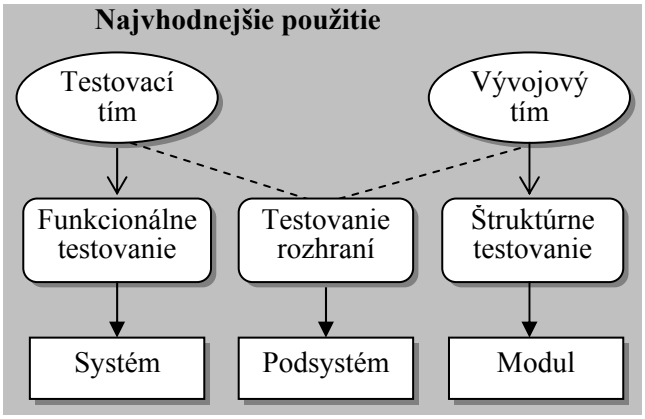
1. **náhodné testovanie**
2. **funkcionálne testovanie** – na základe špecifikácie programu (vstupy, výstupy)
   → *metóda čiernej skrinky* (angl. black box, data driven, functional, input/ output driven, closed box)
3. **štrukturálne testovanie** – na základe vnútornej štruktúry
   → metóda bielej skrinky (angl. white box, glass box, logic driven, path oriented, open box)
4. **testovanie rozhraní** – na základe znalostí rozhraní medzi modulmi a špecifikácie programu

Použitie jednotlivých prístupov:
na ľubovoľnej úrovni

**Proces dynamického testovania**




**Najvhodnejšie použitie**



## Statické overovanie

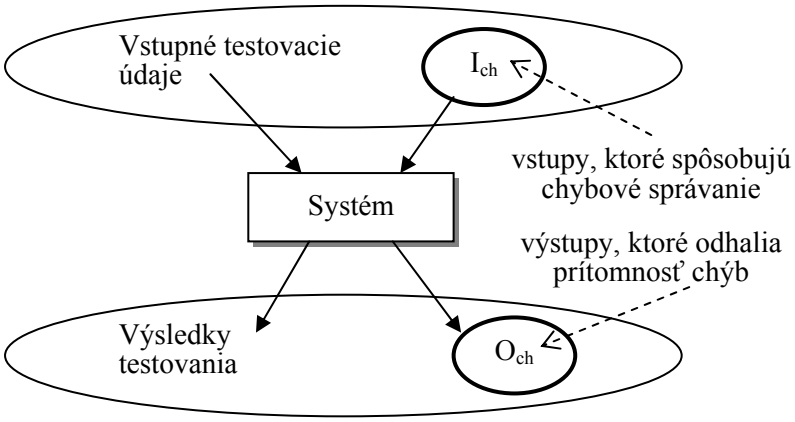
- nevyžaduje vykonanie programu
- najčastejšie sa sleduje zhoda so špecifikáciou a požiadavkami používateľa

**Prístupy statického overovania:**

1. *formálna a neformálna prehliadka dokumentu* (angl. walkthrough, inspection)
   - zakladá sa na statickej prehliadke vytvorených dokumentov (aj textov programov)
2. *matematická verifikácia*
   - formálny matematický dôkaz
   - sledovaný dokument musí byť formálne reprezentovaný (definovanie sémantiky)

# Funkcionálne testovanie

- zistenie, či vstupno-výstupné správanie vyhovuje špecifikácii (matematická funkcia sa špecifikuje vstupmi a výstupmi)
- neuvažuje sa vnútorná štruktúra, logika modulu (systému)
- testovacie vstupy sa odvádzajú priamo zo špecifikácie
- úplné funkcionálne testovanie v praxi nemožné
- neuvažovaním logiky systému môže narásť množina testovacích vstupov, najmä pri testovaní celého systému alebo podsystému (nezávislé časti systému)




**Cieľ funkcionálneho testovania:** vybrať také testovacie vstupy, pre ktoré pravdepodobnosť príslušnosti do množiny $I_{ch}$ je vysoká

**Príklad:**

```
//program vynásobí dve čísla
        ...
prod = x * y;
if (prod == 42)
      prod = 0;
else printf("%d", prod)
```

chyba pri výsledku 42 sa funkcionálnym testovaním pravdepodobne neodhalí

## Rozdelenie vstupov/výstupov do tried ekvivalencie

- triedy ekvivalencie, pre ktoré je správanie systému identické (vstup-výstup)
- vytvorenie tried ekvivalencie na základe *vstupnej podmienky* a *výstupnej podmienky*
- kombinácia pri viacerých vstupoch/ výstupoch

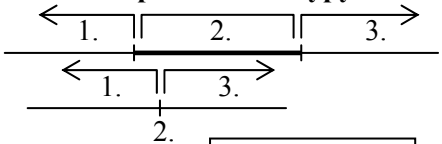
**Vlastnosti tried ekvivalencie**

- každý možný vstup/výstup patrí do jednej z tried ekvivalencie
- žiadny vstup/výstup nepatrí do viacerých tried ekvivalencie
- ak sa pri danom vstupe/výstupe zistí chyba, tak rovnakú chybu možno odhaliť použitím iného vstupu/výstupu z danej triedy ekvivalencie

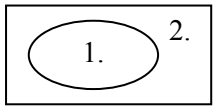
☞ *Nezabudnúť na chybové prípady!*

**Určenie tried ekvivalencie pre niektoré typy V/V**

1. ROZSAH




2. HODNOTA




3. PRVOK MNOŽINY

4. PRAVDIVOSTNÁ HODNOTA — TRUE 1. FALSE 2.

**Výber testovacích údajov z tried ekvivalencie**

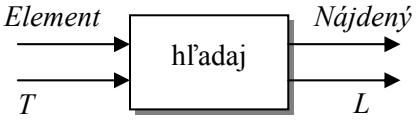
- „stred“ triedy ekvivalencie
- „hranice“ triedy ekvivalencie (prípadne spolu so susediacimi hodnotami)
- náhodne (používa sa iba na doplnenie množiny testovacích vstupov)

**Prístupy k testovaniu údajovej štruktúry pole:**

- pole s jednou hodnotou
- polia s rôznymi veľkosťami v rôznych testoch
- testy s prístupom k prvému, „strednému“ a poslednému prvku

**Príklad:**

*Element*: ELEM; *T*: ELEM_ARRAY;
*Nájdený*: BOOLEAN; *L*: ELEM_INDEX




Vstupná podmienka:

- pole *T* má aspoň jeden prvok

Výstupná podmienka:

- nájdený prvok má index *L* (*Nájdený* a *T*(*L*) = *Element*) alebo
- hľadaný prvok nie je v poli (not *Nájdený* a not ∃ *i*: *T*(*i*) = *Element*)

Triedy ekvivalencie:

1. vstupy, kde sa prvok nachádza v poli (*Nájdený* = true)
2. vstupy, kde sa prvok nenachádza v poli (*Nájdený* = false)

Triedy ekvivalencie:

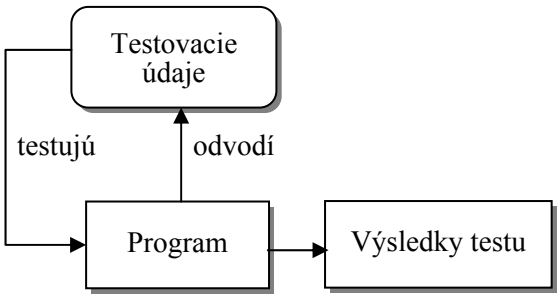
| POLE | PRVOK |
|---|---|
| jedna hodnota | je v poli |
| jedna hodnota | nie je v poli |
| viac ako jedna hodnota | prvý prvok poľa |
| viac ako jedna hodnota | v strede poľa |
| viac ako jedna hodnota | posledný prvok poľa |
| viac ako jedna hodnota | nie je v poli |

Testovacie vstupy a očakávané výstupy:

| POLE (*T*) | PRVOK (*Element*) | VÝSTUP (*Nájdený*, *L*) |
|---|---|---|
| 17 | 17 | true, 1 |
| 17 | 0 | false, ?? |
| 17, 29, 21 | 17 | true, 1 |
| 41, 18, 9, 31, 30,16 | 9 | true, 3 |
| 17, 18, 21, 23, 5, 1, 19 | 17 | true, 6 |
| 21, 23, 29, 33 | 25 | false, ?? |

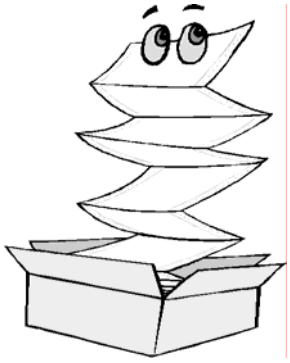
## Štrukturálne testovanie

- vychádza sa z vnútornej štruktúry programu; testuje sa implementácia programu
- snaha o pokrytie rôznych štruktúr programu
  - riadenie
  - údaje
- kritériá:
  - založené na tokoch riadenia (pokrytie ciest, pokrytie rozhodovacích blokov alebo podmienok a pokrytie príkazov)
  - založené na tokoch údajov
  - mutačné testovanie




### Kritériá založené na riadení

- vychádza sa z grafu tokov riadenia
  - uzly → príkazy
  - hrany → možný tok riadenia (pri vykonávaní programu)
- nemožnosť úplného testovania tokov riadenia
- testovanie založené na riadení je *nespoľahlivé*



*Problémy:*

☞
```
if (x + y + z) / 3 = x
      then  put("x, y, z majú rovnaké
                  hodnoty");
      else  put("x, y, z sú rôzne");
```
Test 1: x = 1, y = 2, z = 3

Test 2: x = y = z = 2

Cesty sú pokryté a chyba ostala neodhalená.

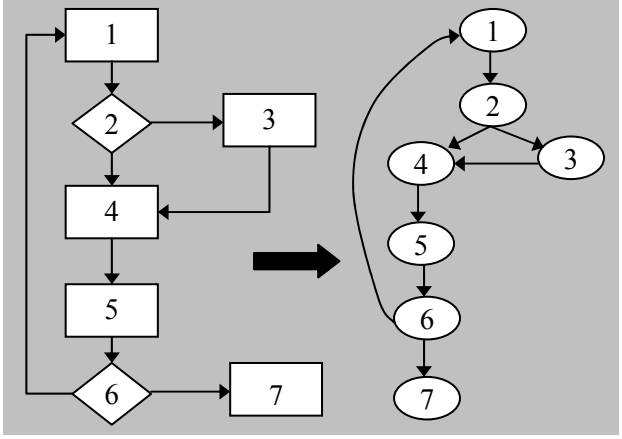
☞
(a)
```
if d = 0
   then delenie nulou;
   else x := n/d;
```
(b)
```
x := n/d;
```
iba jedna cesta, ale treba testovať d = 0 aj d ≠ 0

**Pokrytie ciest**

– každá cesta v programe sa vykoná aspoň raz

1. Vytvorenie grafu tokov riadenia
2. Vytvorenie množiny nezávislých ciest (každá cesta definuje aspoň jeden nový príkaz alebo podmienku)
   Cesta 1: 1-2-3-4-5-6-7
   Cesta 2: 1-2-3-4-5-6-1-…
   Cesta 3: 1-2-4-5-6-7
   Cesta 4: 1-2-4-5-6-1-…
3. Príprava testovacích údajov, ktoré zabezpečia vykonanie každej cesty
4. Vykonanie testov, porovnanie dosiahnutých výsledkov s očakávanými

**Príklad grafu tokov riadenia**




**Pokrytie rozhodovacích blokov**

– všetky možné výsledky rozhodnutí v programe sa vykonajú aspoň raz

1. Vytvorenie grafu tokov riadenia a nájdenie všetkých bodov rozhodovania
2. Vytvorenie množiny ciest tak, aby sa zahrnuli všetky možnosti rozhodovania
   Cesta 1: 1-**2-3**-4-5-**6-7**
   Cesta 4: 1-**2-4**-5-**6-1**-…
3. Príprava testovacích údajov, ktoré zabezpečia vykonanie každej cesty
4. Vykonanie testov, porovnanie dosiahnutých výsledkov s očakávanými

**Pokrytie príkazov**

– každý príkaz v programe sa vykoná aspoň raz

1. Vytvorenie grafu tokov riadenia a nájdenie všetkých bodov rozhodovania
2. Vytvorenie množiny ciest tak, aby spolu zahŕňali každý príkaz (uzol grafu) aspoň raz
   Cesta 1: 1-2-3-4-5-6-7
3. Príprava testovacích údajov, ktoré zabezpečia vykonanie každej cesty
4. Vykonanie testov, porovnanie dosiahnutých výsledkov s očakávanými

☞ *Vo všeobecnosti pokrytie príkazov vyžaduje menej ciest ako pokrytie rozhodovacích blokov a pokrytie rozhodovacích blokov menej ako pokrytie ciest.*

## Mutačné testovanie

– cieľom je vytvorenie dôveryhodných testovacích vstupov
– testovacie kritérium: testovacie vstupy musia byť také, aby rozlíšili medzi originálnym programom a mutovaným (zmeneným) programom
  - chyby určeného typu sa zavedú do programu
  - testuje sa s cieľom identifikácie zavedených chýb
  - ak sa identifikujú všetky zavedené chyby, pôvodný program by nemal mať tieto chyby; inak by sa v programe identifikovali použitím zistených testovacích vstupov
– testuje sa dovtedy, kým *mutačné skóre* nemá hodnotu 1 (alebo blízku 1)

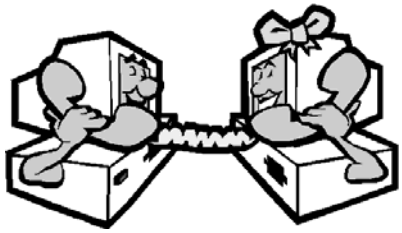
$$D / (N - E)$$

D: počet rozlíšiteľných mutantov (testovacími vstupmi)
N: počet mutantov celkovo
E: počet ekvivalentných mutantov

## Kritériá založené na tokoch údajov

– využíva sa informácia o definovaní a používaní údajov
– pokrytie definícií a použití premenných
– vychádza sa z grafu definície a použitia údajov
  uzly → DEF: reprezentuje definíciu premennej
  C-USE: reprezentuje použitie premennej
  P-USE: reprezentuje použitie premennej v podmienke

## Testovanie rozhraní



- najmä pri integrácii (pod)systémov
- snaha o odhalenie chýb, ktoré vznikajú pri komunikácii súčiastok (modulov)
  - nesprávne použitie rozhrania (napr. zlý typ parametrov)
  - chyby v časovaní (pri systémoch reálneho času)
  - nesprávne pochopenie rozhrania (volaný modul sa nespráva tak, ako sa očakáva a to spôsobí neočakávané správanie sa volajúceho modulu)

## Statické overovanie

- nevyžaduje vykonanie programu
- sleduje sa zhoda so špecifikáciou, resp. požiadavkami používateľa

### Prehliadka dokumentu

- ⇨ neformálna: *walkthrough*
- ⇨ formálna: *inspection* (Michael Fagan, IBM, 1976)

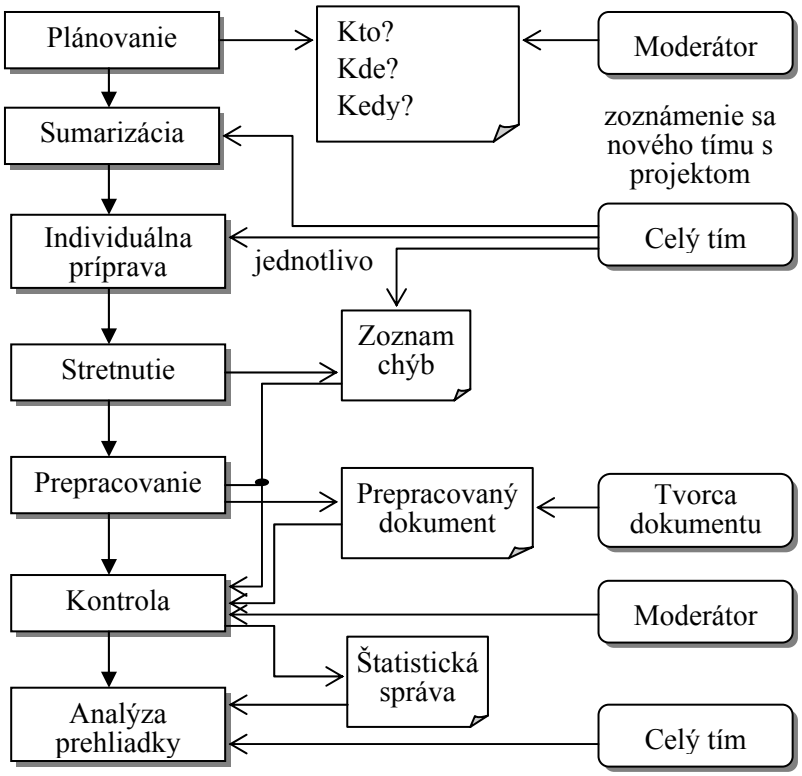
- používa sa v každej etape softvérového projektu
- sledujú sa všetky typy chýb; nie iba chyby v logike a funkcionalite
- formálna prehliadka sa vykonáva v predpísanej postupnosti krokov
- stretnutie sa ohraničuje max. 2 hodinami
- formálnu prehliadku vedie vyškolený moderátor
- efektívnosť prehliadky sa zlepšuje použitím zoznamov otázok (angl. checklist)
- udržiavajú sa štatistiky o type chýb a ich počte; tieto sa analyzujú s cieľom zvýšenia efektívnosti prehliadky

- zúčastňujú sa odborníci *všetkých* úrovní v organizácii okrem najvyššieho vedúceho
  - široké spektrum znalostí (revidujúca etapa, nasledujúca etapa, zákazník, používateľ, manažér, zástupca skupiny zabezpečenia akosti)
  - jednotliví zúčastnení môžu mať pridelené špeciálne úlohy

*Zoznam otázok pri prehliadke textu programu*

- Sú použité všetky premenné?
- Nie je konflikt medzi skutočnými parametrami a rozhraním?
- Je správna podmienka ukončenia cyklu?
- Má každá funkcia zrozumiteľný formulár?
- Vyhovujú mená premenných norme …

**Kroky formálnej prehliadky**



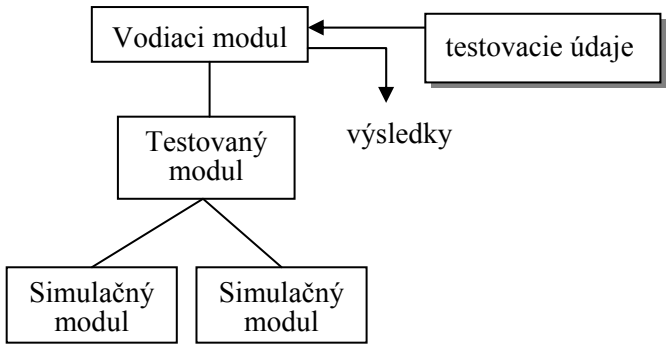

**Prehliadka textu programu**

Prehliadka textu programu predstavuje množinu procedúr, ktorých úlohou je identifikácia chýb počas spoločného prechádzania zdrojovým textom programu. Pri každej súčiastke treba vykonať dve činnosti:

- programátor najskôr podrobne vysvetlí logiku modulu a
- potom sa modul analyzuje vzhľadom na zoznam najčastejšie sa vyskytujúcich programátorských chýb (napr. chyby pri výpočte a porovnávaní, neriešené vetvenia a pod.).

Pri prehliadke sa často vychádza aj zo špecifikácie: na spoločnom stretnutí inšpektorov sa analyzuje logika modulu vzhľadom k jeho špecifikácii. Analýza sa vo väčšine prípadov vykonáva s vybranými testovacími údajmi.

## Stratégie testovania

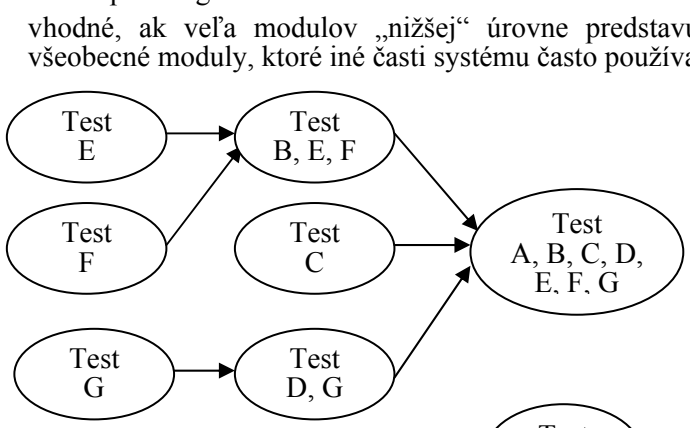


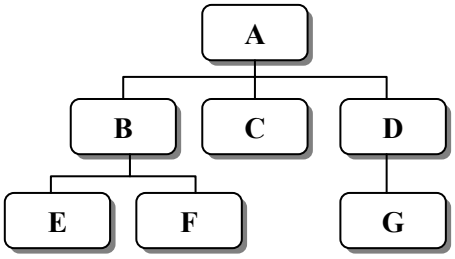

- prístup k testovaniu celého systému
  - testovanie zdola-nahor
  - testovanie zhora-nadol
  - jednofázové testovanie
  - sendvičové testovanie
  - testovanie porovnávaním

### Testovanie zdola-nahor

angl. bottom-up testing

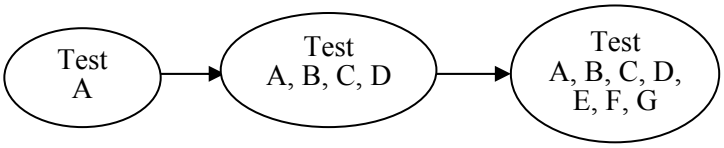
- vhodné, ak veľa modulov „nižšej“ úrovne predstavuje všeobecné moduly, ktoré iné časti systému často používajú




### Testovanie zhora-nadol

angl. top-down testing

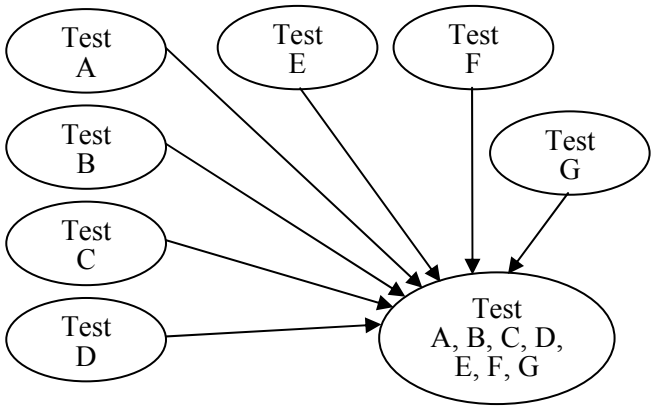



- inkrementálne zaraďovanie modulov
  - do hĺbky $A \rightarrow B \rightarrow E \rightarrow F \rightarrow C \rightarrow D \rightarrow G$ (možnosť paralelného testovania)
  - do šírky $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$
- problémy so simuláciou modulov na nižších úrovniach
- obmedzená znovupoužiteľnosť modulov

### Jednofázové testovanie

angl. big-bang testing

- moduly sa otestujú samostatne a potom sa naraz integrujú
- hodí sa iba pre malé systémy
- náročná identifikácia miesta chyby pri integrácii
- náročné odlíšenie chýb v rozhraniach modulov od ostatných chýb




☞ *Jednofázové testovanie sa neodporúča.*

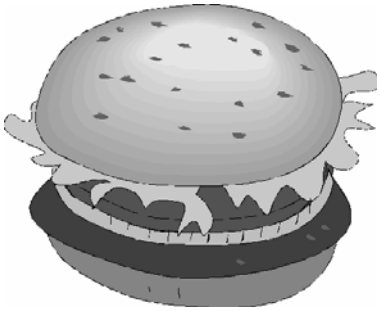
### Sendvičové testovanie

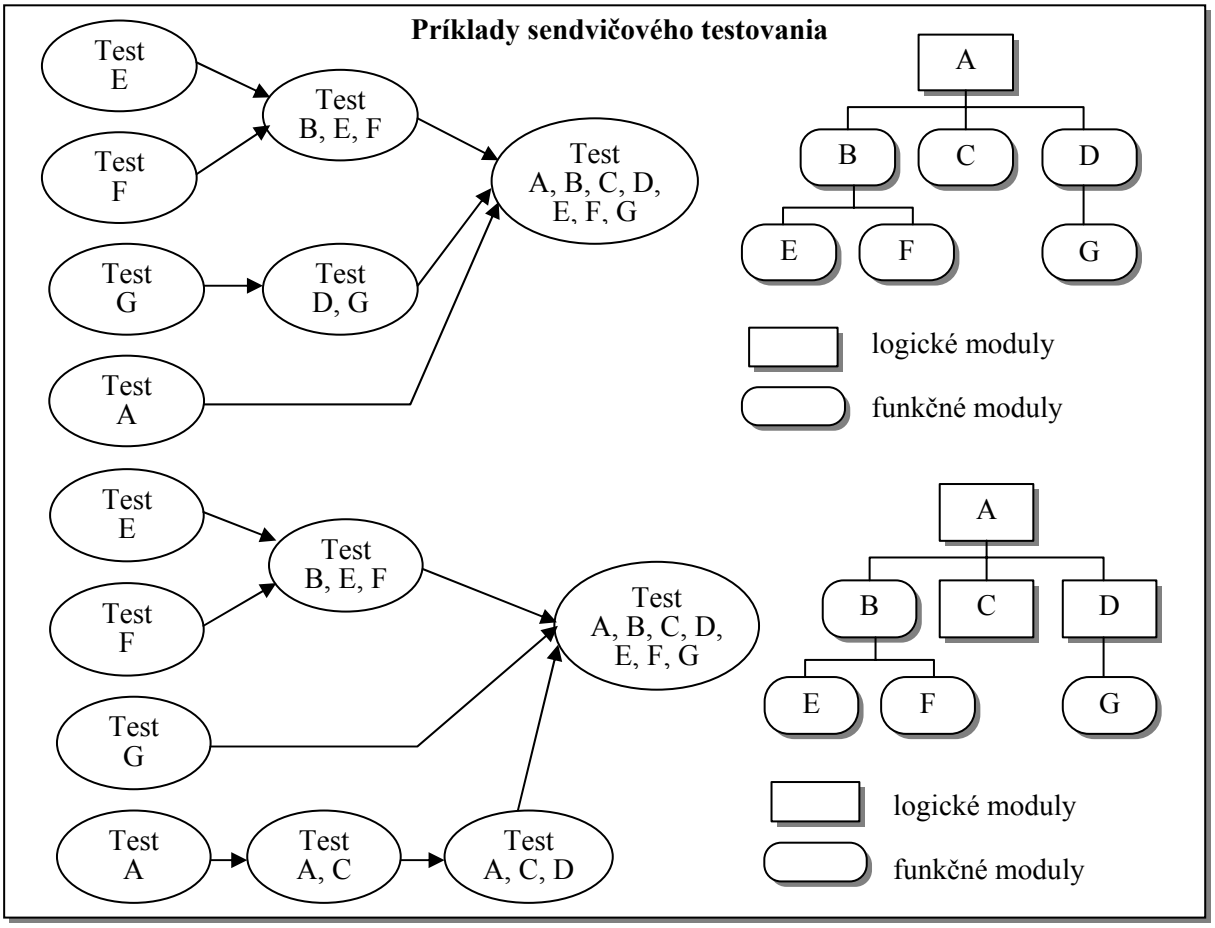
angl. sandwich testing

- kombinácia stratégie zhora-nadol a zdola-nahor
- moduly sa rozdelia do dvoch skupín:
  - *logické:* zabezpečujú najmä riadenie a rozhodovanie
  - *funkčné:* zabezpečujú najmä vykonávanie požadovaných funkcií (operácií)

LOGICKÉ MODULY → testujú sa zhora-nadol

FUNKČNÉ MODULY → testujú sa zdola-nahor

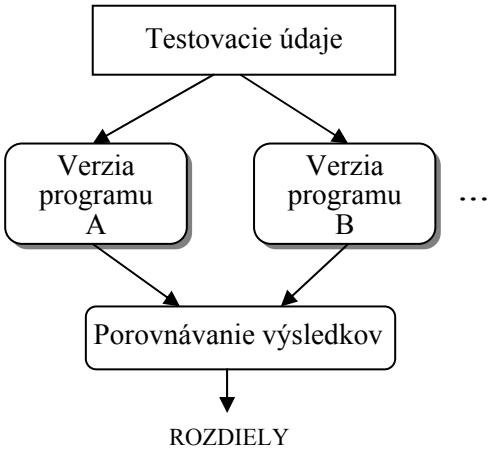





## Testovanie porovnávaním

angl. comparison testing, back-to-back testing

- existuje viac verzií systému na testovanie
  - prototyp
  - vývoj vysoko spoľahlivých systémov použitím techniky programovania N-verzií
  - vývoj viacerých verzií produktu pre rôzne platformy
- rovnaké výsledky značia, že verzie *pravdepodobne* pracujú správne
- problémy:
  - rovnaké chyby vo verziách
  - nevyhovujúca špecifikácia




# Testovanie produktu

CIEĽ: zaručiť úspešný priebeh akceptačného testovania

- testovanie funkčnosti celého systému
- testovanie robustnosti celého systému
- kritické testovanie (angl. stress testing): testujú sa hraničné podmienky
- testovanie objemu údajov (anlg. volume testing)
- regresné testovanie (v prípade, ak už existoval nejaký systém)
- testovanie použiteľnosti
- testovanie splnenia ohraničení
  - doba odozvy
  - pamäťové nároky
  - bezpečnosť (preniknutie do systému by malo byť drahšie ako informácie v ňom)
- testovanie zotavenia (chyby, nesprávne vstupy, zariadenia, odpojenie napájania,…)
- testovanie dokumentácie: dodržanie štandardov, aktuálnosť, použiteľnosť

## Akceptačné testovanie

- používateľ určí, či produkt spĺňa zadanie
  - správnosť
  - robustnosť
  - výkonnosť
  - dokumentácia
- testuje sa na reálnych údajoch

☞ *Ďalšie zmeny po akceptácii predstavujú údržbu.*

## Alfa a beta testovanie

- produkt, u ktorého sa predpokladá veľké rozšírenie a nemožno vykonať akceptačné testy u každého zákazníka (kompilátory, operačné systémy, rôzne podporné prostriedky)
- *alfa test*
  - na pracovisku, kde sa softvér vyvíja
  - testuje používateľ, vývojoví pracovníci ho sledujú a zaznamenávajú chyby
  - známe prostredie
- *beta test*
  - testovanie u používateľov používateľmi
  - prostredie pre vývojový tím neznáme
  - výsledkom je správa používateľa ⇒ modifikácie softvéru ⇒ softvér sa odovzdáva (predáva) na používanie

## Prostriedky na podporu testovania

**Statické analytické prostriedky**

Statická analýza predstavuje analýzu zdrojového textu programu bez jeho skutočného vykonávania. Vstupmi statických analytických prostriedkov sú požiadavky, návrhové dokumenty a text programu. Tieto prostriedky kontrolujú tok údajov a zaznamenávajú chyby ako napr. neinicializované premenné, nekonzistentné rozhrania, nikdy nevykonávané príkazy. Na základe týchto chýb možno odvodiť ďalšie chybové charakteristiky programu.

Informácie poskytované statickými analyzátormi zahŕňajú:

- chybové správy o syntaktických chybách,
- počet výskytov príkazov v programe podľa ich typu,
- mapy vzájomných odkazov použitia identifikátorov,
- analýza spôsobu použitia identifikátorov v každom príkaze (dátový zdroj a cieľ, volajúci parameter, nepoužívaný parameter),
- procedúry a funkcie volané každým podprogramom,
- neinicializované premenné,
- premenná s priradenou hodnotou, ktorá sa nikde nevyužíva,
- izolované časti programu, ktoré nemožno vykonať pri žiadnej kombinácii vstupných údajov,
- odchýlky od implementačných štandardov (jazykových a interných pre danú organizáciu),
- nesprávne používanie globálnych premenných, zoznamov parametrov (napr. nesprávny počet parametrov, typová nezhoda, neinicializované vstupné parametre, výstupné parametre s nepriradenou hodnotou, výstupné parametre bez ďalšieho použitia vo volajúcom programe, nepoužité parametre).

Experimentálne výsledky s použitím metód statickej analýzy ukazujú, že kombináciou s metódami prehliadky textu programu možno identifikovať 50-80% chýb v programe.

**Dynamické analytické prostriedky**

Dynamické analytické prostriedky sú prostriedky, ktoré zhromažďujú a analyzujú informácie na základe vykonávania testovaného programu. V rámci dynamickej analýzy sa vykonávajú tieto operácie:

- analýza pokrytia (stanovenie, určenie miery aktivácie štrukturálnych elementov programu na určenie adekvátnosti vykonaného testu);
- trasovanie (vytvorenie histórie vykonávania programu; môže ísť o trasovanie výpočtovej cesty, trasovanie tokov logiky spracovania, trasovanie dátových tokov a trasovanie nastavených bodov prerušenia);
- analýza zaťaženia (určenie, ktoré časti programu sa vykonávali najčastejšie);
- simulácia (reprezentujú sa určité vlastnosti správania fyzického alebo abstraktného systému pomocou počítača);
- analýza časovania (určenie skutočného spotrebovaného CPU času na vykonanie programu alebo jeho časti);
- analýza využívania zdrojov (analýza využívania hardvérových a softvérových systémových prostriedkov);
- symbolické vykonávanie (rekonštrukcia logiky a výpočtu pre danú výpočtovú cestu prostredníctvom vykonania cesty so symbolickými údajmi namiesto skutočných);
- overovanie platnosti tvrdení definovaných používateľom o vzťahu medzi jednotlivými elementmi programu (tvrdenie možno reprezentovať logickým výrazom, ktorý špecifikuje podmienku alebo vzťah medzi programovými premennými; overovanie možno vykonať na symbolickej úrovni alebo pri vykonávaní programu);
- vyhodnocovanie ohraničení (generovanie a/alebo riešenie vstupných alebo výstupných ohraničení na dokázanie správnosti programu; táto funkcia býva časťou symbolických vyhodnocovačov alebo automatických generátorov testovacích údajov).