Testování Softwaru

Strategie testování integruje metody návrhu testů do celkového plánu procesu tvorby softwarového produktu.

Strategie testování zahrnuje plánování testů, návrh testů, provedení testů a vyhodnocení jejich výsledků.

- **Testování začíná na úrovni modulu a pokračuje “směrem ven” k integraci kompletního počítačového systému.**
- **Pro různé situace se používají různé techniky testování.**
- **Pro malé projekty je testování řízeno realizátorem, pro větší nezávislou skupinou.**
- **Testování a odstraňování chyb jsou rozdílné aktivity, avšak odstraňování chyb musí být zahrnuto do strategie testování.**

V&V (verification and validation):

Verifikace: **Ověření, že software správně implementoval specifické funkce.**

“Vytvořili jsme produkt správně?”

Validace: **Ověření, že software odpovídá požadavkům zákazníka.**

“Vytvořili jsme správný produkt?”

V&V jsou součástí SQA (SW Quality Acquisition).

Testování hraje důležitou roli ve V&V.

Nesprávné názory na strategii testování

Kdo tu práci požaduje, kdo ji bude užívat, jaký bude ekonomický užitek při úspěšném ukončení, je ještě jiná možnost, jak to vyřešit?

- **Tvůrce by neměl provádět žádné testování!**
- **Software má testovat někdo nezávislý, který to provede bez jakýchkoli ohledů**
- **Tester má vstoupit do projektu až v okamžiku testování.**

Tvůrce by měl testovat jednotky (moduly),

v mnoha případech řídí také integrační testy.

Až v okamžiku, kdy je software kompletní

nastupuje nezávislá skupina testérů (ITG - independent test group)

systémové požadavky

analýza požadavků

návrh

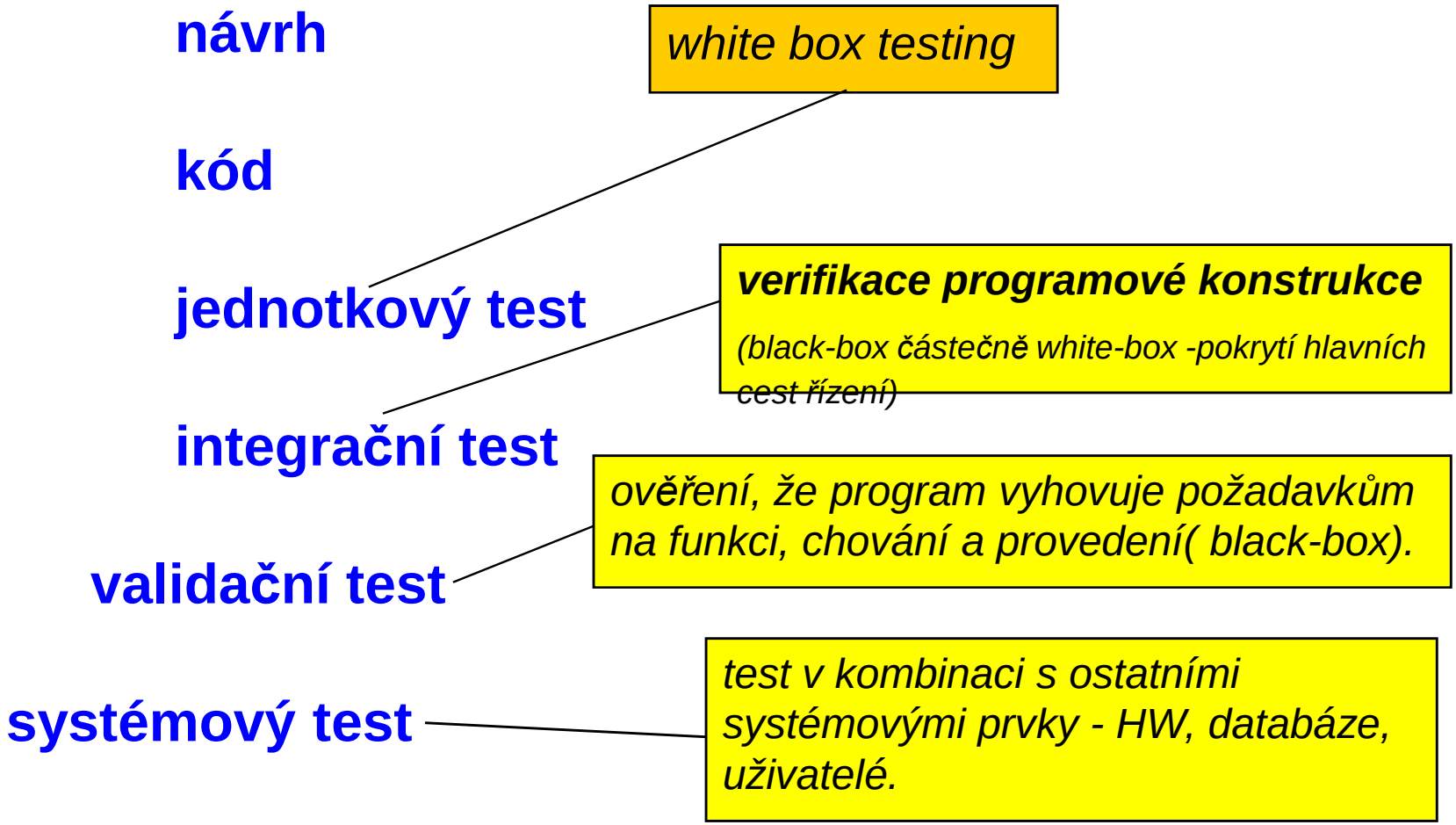
kód

jednotkový test

integrační test

validační test

systémový test



Kdy skončit testování,

jak poznat, že jsme testovali dostatečně?

Musa, J.D., Ackerman, A.F.: Qualifying Software Validation: When to Stop Testing? IEEE Software, May 1989, pp. 19-27

Logarithmic Poisson execution-time model:

$$f(t) = (1/p) \ln (l_0 \, p \, t + 1)$$

*kde **f(t)** je kumulativní počet poruch, které se očekávají, že nastanou po testování softwaru po určitou dobu t,*

***l_0** je počáteční softwarová **intenzita poruch** (poruchy za časovou jednotku) na začátku testování*

***p** exponenciální snížení intenzity poruch, po odhalení chyb a jejich odstranění*

Okamžitá intenzita poruch l(t) :

$$l(t) = l_0 / (l_0 \, p \, t + 1)$$

Testování jednotek

Testování zaměřené na verifikaci malých jednotek softwarového návrhu - modulů. Podle popisu návrhu procedur jsou testovány důležité cesty uvnitř modulu. Testování jednotek je prováděno metodami white-box testing, paralelně pro více modulů.

V rámci testování jednotky se prověřuje

- rozhraní
- lokální datové struktury (zda dočasně uložená data zachovávají svou integritu)
- okrajové podmínky (zda modul pracuje správně na hranicích, omezujících výpočet)
- nezávislé cesty (zaručující, že každý příkaz bude proveden alespoň jednou)
- cesty pro zpracování chyb

Myers G. The Art of Software Testing, Wiley, 1979

Kontrolní seznam (checklist) pro testování rozhranní

1 **Je počet vstupních parametrů roven počtu argumentů?**

2 **Odpovídají atributy parametrů a argumentů?**

3 **Je počet argumentů přenášených do volaného modulu roven počtu parametrů?**

4**Jsou definice globálních proměnných konsistentní v modulu?**

6 **.....**

Pokud modul ovládá externí I/O, musí být doplněny další testy:

- Správné atributy souborů?
- Správné příkazy OPEN/CLOSE?
- Odpovídá příkaz I/O specifikacím?
- Odpovídá velikost bufferu velikosti záznamu?
- Jsou soubory před použitím otevírány?
- Správné podmínky EOF?
- Ošetření I/O chyb?
- Kontrola textových výstupních informací?

Pro lokální datové struktury

- Nevhodné a nekonsistentní typy?
- Chybná inicializace nebo default hodnota?
- Nesprávné (překlepy) jména proměnných?
- Nekonsistentní datové typy?
- Podtečení, přetečení a adresní výjimky?

Kromě toho je potřeba kontrolovat zpracování globálních dat.

Mezi časté chyby patří

- nepochopené nebo nesprávné priority aritmetických operací
- smíšené operace (mixed mode)
- nesprávná inicializace
- nesprávná symbolická reprezentace výrazu

Antibugging.

Předvídání chybného chování při návrhu cesty pro zpracování chybného stavu, které ukončují výpočet s chybovým hlášením. Je třeba testovat, zda nenastane některý z následujících případů:

- Popis chyb není příliš inteligentní.
- Oznámené chyby nekorespondují se skutečnými.
- Chyba způsobí zásah do systému dříve, než je ošetřena.
- Nesprávné ošetření výjimečných podmínek.
- Popis chyby neposkytuje dostatečné informace k lokalizaci chyby.

Pomocné programy pro testování modulů

řídicí programy* - *drivers (nahrazují nadřazené programy)

testovací kódy (makety)* - *stubs (nahrazují volané podprogramy).

Někdy nelze tyto programy napsat pro každý modul, pak se musí testování jednotky odložit na úroveň kroku integračního testování.

Integrační testování

- Přístup “velkého třesku”
- Inkrementální integrace

Integrace shora-dolů a zdola-nahoru

Integrace shora-dolů

*začíná se hlavním řídicím modulem a postupuje se směrem dolů: buď strategií **do hloubky** nebo **do šířky**.*

Hlavní modul je "driver", "stubs" nahrazují všechny podřízené moduly

Postupně (buď podle přístupu do hloubky nebo do šířky) jsou nahrazovány "stubs" skutečnými moduly.

Po každém přidání modulu je systém otestován.

Regresní testování má zajistit, že nebyly zavlečeny nové chyby.

Problém může nastat, když očekáváme významné výstupy z podřízených modulů, které nám "stubs" neumí poskytnout.

Úrovně složitosti "stubs":

- zobrazí zprávu
- zobrazí aktuální parametry
- navrací hodnoty z tabulky nebo externího souboru
- provádí vyhledání z tabulky podle vstupních parametrů a vrací příslušné výstupní parametry

Integrace zdola nahoru

Eliminuje se potřeba "stubs".

- Nejnižší moduly jsou spojeny do skupin (clusterů), které provádí specifické funkce.
- Je napsán "driver", který je koordinuje.
- Skupina (cluster) je otestována.
- Drivery se odstraní a skupiny se pospojují směrem výše v programové struktuře.

Regresní testování

Testuje se, zda nenastal vedlejší efekt přidáním nového modulu (zavlečené chyby) - opakováním předchozích testů.

Existují nástroje na toto testování (Capture-playback tools) - opakují tři rozdílné třídy testů:

- representativní vzorek testů, zkoušejících všechny softwarové funkce
- testy zaměřené na funkce, které by mohly být pravděpodobně zasaženy změnou
- testy softwarových komponent, které byly změněny

Regresní testování

Testuje se, zda nenastal vedlejší efekt přidáním nového modulu (zavlečené chyby) - opakováním předchozích testů.

Existují nástroje na toto testování (Capture-playback tools) - opakují tři rozdílné třídy testů:

- representativní vzorek testů, zkoušejících všechny softwarové funkce
- testy zaměřené na funkce, které by mohly být pravděpodobně zasaženy změnou
- testy softwarových komponent, které byly změněny

Integrační testy musí být dokumentovány (plán testování, popis pomocného sw - drivery a stubs, pořadí integrace, popis testových případů, očekávaných a skutečných výsledků).

Validační testování

Provádí se po integraci a slouží k ověření, že software splňuje “rozumná očekávání” zákazníka, která jsou definovaná ve specifikacích softwarových požadavků (“validační kritéria”).

Provádí se metodami black-box testing.

Pokud je sw určen pro jednoho uživatele, předá se mu do užívání k ***akceptačním testům***.

Validační testování (pokr.)

Pokud je SW určen pro více různých uživatelů, provádí se alfa a beta testování.

Alfa testování provádí zákazník v řízeném prostředí dodavatele (“programátor se mu dívá přes rameno”).

Beta testování se provádí u jednoho nebo více zákazníků. Vývojář není přítomen. SW se zkouší v “živých” podmínkách. Zákazník zapisuje všechny problémy (skutečné i imaginární) a určitých intervalech je posílá dodavateli.

Systémové testování

Je série různých testů, která prověřuje celý počítačový systém (HW, SW prostředí, databáze, lidi…).

Systémové testování (pokr.)

Recovery testing (testování obnovy)

Systémové testování ověřující, že poruchy byly řádně ošetřeny v předepsaném čase. Pokud je ošetření automatické, vyhodnocuje se re-inicalizace, mechanismus checkpointů, obnova dat, restart. Je-li prováděno manuálně, ověřuje se, zda "mean-time to repair" byl v limitu.

Systémové testování (pokr.)

Security testing

(bezpečnostní testování)

Testování odolnosti proti útokům hackerů. Tester supluje roli hackera a snaží se proniknout do systému. Má-li dost času a zdrojů, tak se mu to podaří. Úlohou systémového návrháře je, aby cena proniknutí do systému byla větší než cena informací, které lze získat.

Systémové testování (pokr.)

Stress testing

Cílem je prověřit program v abnormální situací (kvantita, frekvence nebo obsah).

Jak dlouho mohu systém namáhat, než padne?

Systémové testování (pokr.)

Např.

- *generovat 10 přerušení za sekundu*
- *vstupní data zvětšit o řád než je dovoleno a vyhodnotit chování vstupní funkce*
- *požadovat maximální paměť či jiné zdroje*
- *pokusit se způsobit problém v operačním systému*
- *velké nároky na disk*

apod.

Variantou je ***sensitivity testing***, které se snaží objevit kombinace dat (v rámci platného omezení), které mohou způsobit nestabilitu či nesprávnou funkci.

Performance testing (pro zapouzdřené s. nebo s. reálného času).

Ladění (The art of debugging)

Zatímco testování lze plánovat a systematicky provádět podle nějaké strategie, ladění, které je důsledkem úspěšného testování (našly se chyby) je více méně uměním.

Oprava nesouhlasu mezi očekávaným a skutečným výstupem spočívá v nalezení příčiny daného symptomu.

Příčina se buď najde a chyba se opraví, nebo se musí navrhnout jiný test, který ji pomůže lokalizovat

Ladění (pokr.)

Zatímco testování lze plánovat a systematicky provádět podle nějaké strategie, ladění, které je důsledkem úspěšného testování (našly se chyby) je více méně uměním.

Oprava nesouhlasu mezi očekávaným a skutečným výstupem spočívá v nalezení příčiny daného symptomu.

Příčina se buď najde a chyba se opraví, nebo se musí navrhnout jiný test, který ji pomůže lokalizovat

Ladění (pokr.)

Všechny přístupy mohou být podpořeny ladicími nástroji (debugging tools) - traces, automatické generátory testů, výpisy paměti, křížové odkazy (cross-reference), ladicí nástroje překladačů.

Často stačí vysvětlit problém jinému kolegovi (jiný pohled) a chyba se objeví.

Náměty k zamyšlení

- Neopakuje se náhodou chybný kód ještě v jiných částech programu?
- Nemůžeme zavléci do programu další chyby (a jaké), opravíme-li chybu tímto způsobem?
- Co bychom mohli udělat, abychom zabránili vzniku takové chyby v budoucnu? (Snaha odstranit chybu nejenom z produktu, ale i z procesu.)

Tohle je pro Tebe