Strategie testování, validace a verifikace. Testování v průběhu životního cyklu SW díla. Testování jednotek, integrační testování, validační testování, systémové testování, ladění. Principy testování, testovatelnost, návrh testů, návrh testovacích dat. Testování podle struktury dat (black-box) versus testování podle struktury programu (white-box). Akceptační testy, testy použitelnosti (usability tests)

Obsah

Strategie testování, validace a verifikace

Strategie testování integruje metody návrhu testů do celkového plánu 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 celého 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

Verifikace
Verifikace je ověření, že software správně implementoval specifické funkce. Dává odpověď na otázku, jestli jsme vytvořili software správně (funguje správně to, co jsme vytvořili?).

Validace
Validace je ověření, že software odpovídá požadavkům zákazníka. Dává odpověď na otázku, jestli jsme vytvořili správný produkt (je to co jsme vytvořili, to co zákazník chtěl?).

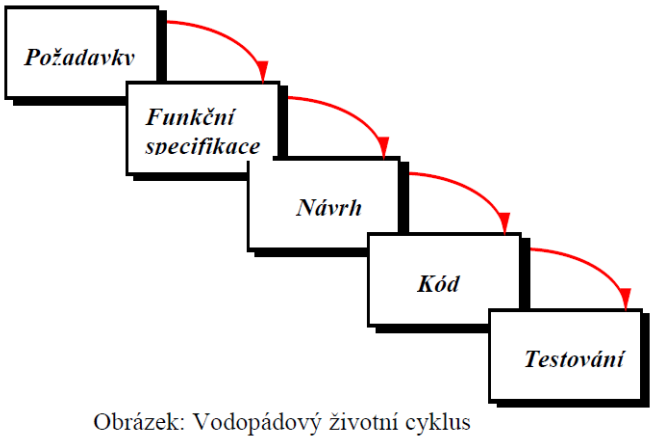
Verifikace a validace jsou součásti SQA (Software Qaulity Assurance – zajištění kvality softwaru) a testování hraje při hledání odpovědi na výše uvedené otázky důležitou roli.

Nesprávné názoru na strategii testování
- 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

Testování v průběhu životního cyklu SW díla

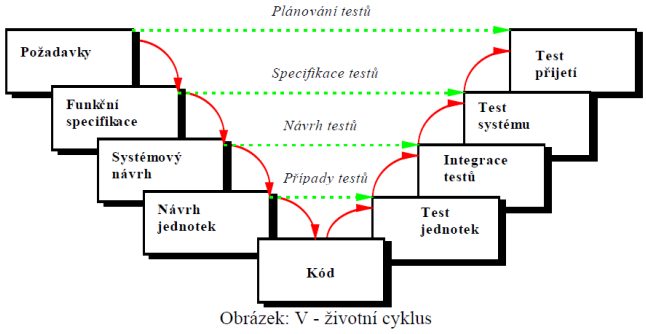
Životní cyklus SW díla je závislý na volbě metodiky zvolené pro vývoj SW. Metodik pro vývoj SW je celá řada a do značné míry ovlivní způsob testování a tvorbu strategie testování SW díla.

Vodopád (Waterfall)



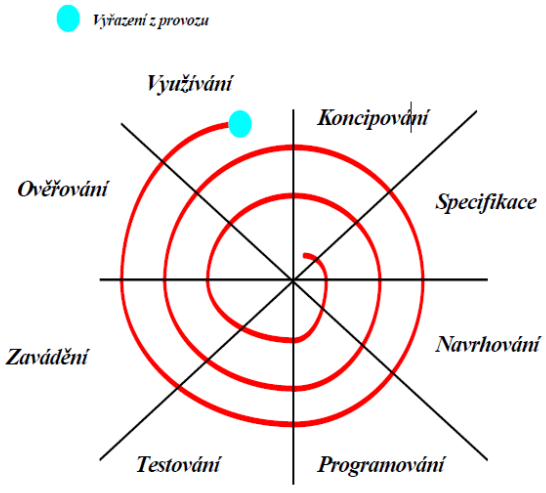
Obrázek: Vodopádový životní cyklus

V-životní cyklus (The V life cycle)



Obrázek: V - životní cyklus

Iterační životní cyklus



Obrázek: Iterační životní cyklus

Atd….

Testování jednotek, integrační testování, validační testování, systémové testování, ladění

Testování jednotek (unit testing)
Je zaměřené na testování malých jednotek – modulů. Podle popisu návrhu procedur jsou testovány důležité cesty uvnitř modulu. Testování jednotek je prováděno metodou white box testování.
V rámci testování jednotky se ověřuje:

- rozhraní
- lokální datové struktury (integrita dat)
- okrajové podmínky
- nezávislé cesty (zaručující, že každý příkaz bude proveden minimálně jednou)
- cesty pro zpracování chyb
- a další

Integrační testování
Dva možné přístupy:

- Přístup velkého třesku
  Spojí se všechny moduly a testuje se vše najednou. Není vhodný pro velké systémy – obvykle generuje mnoho chyb (jedna chyba může vyvolat mnoho dalších a díky tomu se často jako chybové označují moduly, které chybu neobsahují)
- Inkrementální integrace
  Dva přístupy (resp. tři – třetí kombinuje oba níže uvedené přístupy):
  - Integrace shora-dolů
    Při této strategii se začíná hlavním modulem a postupuje se směrem dolů (do hloubky nebo do šířky)
    Postup:
    1. Testujeme hlavní modul (driver), podřízené moduly jsou nahrazeny maketami
    2. Postupně jsou makety nahrazovány skutečnými moduly
    3. Při každém přidání modulu je systém otestován
    4. Regresní testování má zajistit, že nebyly zavlečeny nové chyby
  - Integrace zdola-nahoru
    Eliminuje potřebu maket
    1. Nejnižší moduly jsou spojeny do skupin (cluster), které provádí specifické funkce
    2. Je napsán řídící driver, které koordinuje clustery
    3. Cluster je otestován
    4. Driver se nahradí a skupiny se spojí směrem výše v programové struktuře

Validační testování
Provádí se po integraci a slouží k ověření, že produkt splňuje „rozumná“ očekávání zákazníka, která jsou definována ve specifikaci softwarových požadavků (validační kritéria). Provádí se metodou black-box.

Systémové testování
Slouží k ověření celého systému (všech aspektů)

- Testování obnovy
  Umí se systém zotavit z chyb? (Výpadky proudu, nesmyslná data atd.)
- Bezpečnostní testování
  Je sytém ošetřen proti útokům? (pokus o neautorizovaný přístup, SQL injection atd.)
- Zátěžové testování
  Jakou zátěž systém unese? (počet uživatelů, zpracování extrémního množství dat, složitá data atd.)

Ladění (Debugging)
Nalezla se chyba (popsaná formálně) a nyní je potřeba najít její příčinu. Nalezení pravé příčiny chyby je mnohdy obtížné, proto se často stává, že programátor přidává do kódů pomocné funkce, které mu usnadní odhalení příčiny (pak se otestuje – vyvolá chyba), problém se odstraní, stejně jako podpůrné funkce přidané během ladění a provede se znovu retest opravy.

Principy testování, testovatelnost, návrh testů, návrh testovacích dat

Principy testování

- Základní principy testování můžeme shrnout do následujících bodů:
- Testy by se měly vztahovat k požadavkům zákazníka
- Testy by měly být plánovány v předstihu
- Princip Pareto (80:20): většina všech nalezených má původ v několika málo modulech (je však problém tyto moduly odhalit)
- Testování by mělo začít v malém (jednotkový test) a pokračovat ve velkém (Systémový test)
- Úplné otestování není možné!
- Testování by mělo být provedenou nezávislou třetí stranou (Přináší jiný úhel pohledu)

Testovatelnost

Následující vlastnosti by měl mít dobře testovatelný produkt:

- **Spustitelnost(Operability)**
  Chyby nebrání běhu programu, je tedy možné testovat a vyvíjet současně
- **Přehlednost (Observability)**
  Různé výstupy pro různé vstupy, stavy systému a proměnné jsou viditelné během chodu programu, nesprávné výstupy jsou lehce identifikovatelné, vnitřní chyby jsou automaticky detekovány a zaznamenávány, je dostupný zdrojový kód
- **Kontrolovatelnost(Controlability)**
  všechny možné výstupy jsou generovány nějakou kombinací vstupů, veškerý kód je spustitelný nějakou kombinací vstupů, vstupní a výstupní formát je konzistentní
- **Dekomponovatelnost**
  Kontrolování rozsahu testování můžeme rychleji oddělit problémy a provést následné testy
- **Jednoduchost**
  Čím jednodušší tím lepší
- **Stabilita**
  Zněny nejsou časté, jsou řízené, neovlivní provedené testy
- **Srozumitelnost**
  Srozumitelný návrh, srozumitelné závislosti mezi vnitřními, vnějšími a sdílenými komponentami, dostupnost a srozumitelnost technické dokumentace

Návrh testů

Ačkoliv existují různé příručky a popisy jak dobře testovat SW navrhnout test není tak jednoduché jak by se mohlo na první pohled zdát. Při návrhu testů je potřeba brát v úvahu mnoho aspektů počínaje povahou SW díla a cílovou skupinou uživatelů konče. Bez znalostí kontextu, pochopení funkce systému a cílové skupiny uživatelů není možné navrhnout kvalitní testy. Návrh testů vychází i z použité metodologie tvorby SW.

Vlastnosti dobrého testu:

- S velkou pravděpodobností objeví chybu.
- Měl by hledat chybu tam, kde může být, není redundantní.
- Na testování je málo času, proto by měl mít každý test jiný účel.
- Měl by objevit celou třídu chyb.
- Neměl by být příliš jednoduchý ale ani příliš složitý.

Testy by se měly spouštět nezávisle, aby jejich případné vedlejší efekty nepřekryly chyby.

Návrh testovacích dat

Testovací data je nutné připravovat s ohledem na použitou metodu a účel testu, Jiná tetovací data budeme připravovat pro White box, jiná pro black box testování. Jinou množinu dat budeme používat pro performance testy a jinou pro funkční testování. V praxi se často využívají reálná data dodaná zadavatelem jako základ pro vytváření testovacích dat. Pokud taková data nemáme k dispozici, musíme je vytvořit „z ničeho“ resp. ze specifikace, u white box testu ze znalosti kódu a black box testu vytváříme testovací data do jisté míry intuitivně na základě znalosti zamýšlených funkcí systému. Dobrá testovací data by měla ověřit jak okrajové podmínky, tak množinu očekávaných běžných vstupů. Dobře navržená testovací data umožnují ověření všech komponent systému (těch co mají co dočinění s daty) a jejich funkčnosti. Navrhnout dobrá testovací není jednoduchý úkol.

Testování podle struktury dat (black-box) versus testování podle struktury programu (white-box)

Black-box testování
Testování bez znalosti detailů implementace. Testuje správnost provádění všech funkcí programu. Testy jsou navrhovány a prováděny pouze na základě znalosti koncového využití produktu. Často se definují alespoň základní obrysy testů již ve fázi analýzy a přípravy dokumentace. Tím, že není nutná znalost implementace, je možné připravit a projektovému teamu poskytnout testovací scénáře ještě předtím, než se s implementací vůbec začne.

White-box testování
Testování s použitím znalosti implementace. Testuje správnou funkci všech vnitřních komponent. Na základě znalosti implementace, můžeme definovat testy, které prověří všechny kritické cesty programu. Můžeme lépe testovat a odhalit okrajové podmínky.

Srovnání
Hlavní nevýhodou Black-Box testování je to, že při testování nemusíme postihnout všechna temná zákoutí implementace. Do značné míry je tento typ testování, resp. jeho výsledek závislý na kvalitě vstupních informací, které máme v průběhu testu k dispozici (např. co je a co není chybovým výstupem). Není možné spolehlivě prohlásit, že jsme modul plně otestovali (nevíme, jestli skutečně vyčerpali všechny možnosti).
Pokud potřebujeme ověřit, že moduly programu řádně fungují, použijeme white-box testování. Při použití white-boxu však existuje riziko, že naše testy neověří všechny alternativy (může jich být opravdu mnoho).
Není možné prohlásit, která z uvedených technik testování je lepší či horší. Obě metody jsou hojně používány a obě jsou stejně dobré. Každá se hodí pro ověření funkčnosti produktu avšak v různých fázích jeho vývoje. White-box je logicky dominantní ve fázi jednotkového testování, zatímco black-box vládne při validačním testování (Systém test atd.).

Akceptační testy, testy použitelnosti (usability tests)

Akceptační test

Akceptační test popisuje sadu testovacích případů, které prokáží, že software splňuje zadání zadavatele. Testovací scénáře jsou připravovány ve spolupráci dodavatele a zadavatele. Správně by mely být akceptační testy definovány již při definici zadání a měly by určovat, co zákazník akceptuje jako funkční produkt. V praxi jsem se s tím ještě nesetkal. Akceptační testy se buď neprovádějí vůbec (výjimečný případ) nebo je (častý případ) akceptační test prováděn na základě scénářů, které si připraví zadavatel sám a dodavateli pouze oznámí, že součástí akceptačního testu bude ověření požadovaných funkcí systému. Pokud je obsah akceptačního testu znám dodavateli, je možné začít budovat testovací strategii směřující k úspěšnému zvládnutí akceptačního testu. Pokud není obsah akceptačního testu znám, je potřeba jej u dodavatele simulovat. Na základě znalosti dané oblasti, pro kterou je SW vyvíjen musí být testeři a analytici schopni odhadnout jak bude systém používán a na základě takového odhadu vytvoří obrysy „pseudo“ akceptačního testu. Cílem tohoto snažení je dosažení co nejnižší chybovosti dodaného SW v průběhu akceptačního testu na straně zákazníka.

Testy použitelnosti

Testy použitelnosti dávají odpověď na následující otázky:

- Jak je naše dílo vnímáno uživateli?
- Je použití SW snadné a jednoduché?
- Splňuje SW očekávání uživatelů?
- Apod.

Testy použitelnosti se zaměřují na ověření ovládání, srozumitelnosti a uživatelského komfortu. Často se test použitelnosti provádí ve spolupráci se skutečnými uživateli (beta testování). Výsledkem testu je jejich feedback a chyby nalezené v průběhu jejich používání produktu. Testy použitelnosti má smysl provádět zejména pro produkty, které jsou určeny široké skupině uživatelů. Jenom díky reakcím skutečných uživatelů je možné dodávat produkt, který bude dobře přijímán uživateli. U testování použitelnosti je největším problémem, určit rozsah a složení vzorku uživatelů.