Kapitola 8 – Testovanie softvéru

Testovanie programu

- Testovanie má ukázať, že program robí to, čo je zamýšľané, a odhaliť chyby programu pred jeho uvedením do používania.
- Keď testujete softvér, spúšťate program využívajúci umelé dáta.
- Vo výsledkoch testovacej prevádzky skontrolujete chyby, anomálie alebo informácie o nefunkčných atribútoch programu.
- Dokáže odhaliť prítomnosť chýb NIE ich absenciu.
- Testovanie je súčasťou všeobecnejšieho procesu verifikácie a validácie, ktorý zahŕňa aj techniky statickej validácie.

Ciele testovania programu

- Demonštrovať vývojárovi a zákazníkovi, že softvér spĺňa jeho požiadavky.
  - V prípade zákazkového softvéru to znamená, že pre každú požiadavku v dokumente požiadaviek by mal existovať aspoň jeden test. Pre generické softvérové produkty to znamená, že by mali existovať testy všetkých funkcií systému plus kombinácie týchto funkcií, ktoré budú zahrnuté vo vydaní produktu.
- Na zistenie situácií, v ktorých je správanie softvéru nesprávne, nežiaduce alebo nezodpovedá jeho špecifikácii.
  - Testovanie defektov sa týka odstránenia nežiaduceho správania systému, ako sú pády systému, nechcené interakcie s inými systémami, nesprávne výpočty a poškodenie údajov.

Validácia a testovanie defektov

- Prvý cieľ vedie k overovaciemu testovaniu
  - Očakávate, že systém bude fungovať správne pomocou danej sady testovacích prípadov, ktoré odrážajú očakávané použitie systému.
- Druhý cieľ vedie k testovaniu defektov
  - Testovacie prípady sú navrhnuté tak, aby odhalili chyby. Testovacie prípady pri testovaní defektov môžu byť zámerne nejasné a nemusia odzrkadlovať, ako sa systém bežne používa.

Ciele testovacieho procesu

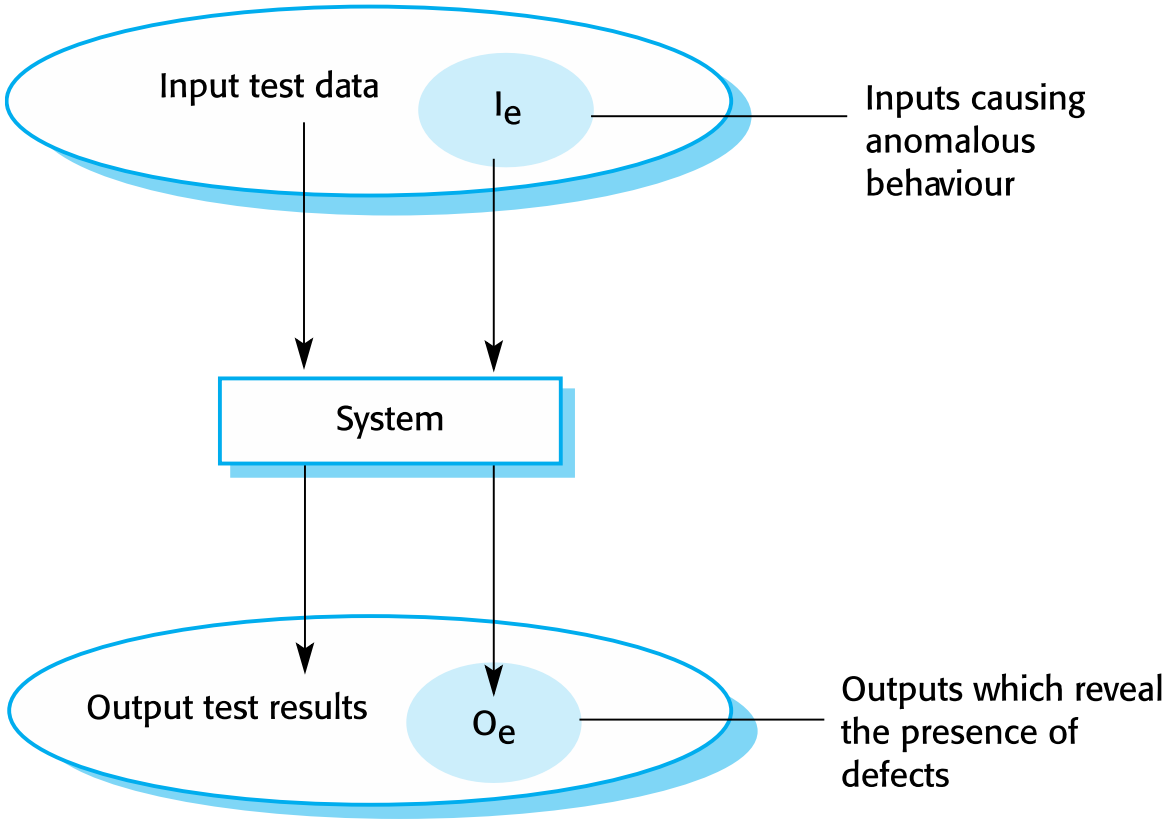
✧ Overovacie testovanie /validácia/

- Demonštrovať vývojárovi a zákazníkovi systému, že softvér spĺňa jeho požiadavky
- Úspešný test ukazuje, že systém funguje tak, ako má.

✧ Testovanie defektov /verifikácia/

- Na zistenie chýb alebo defektov v softvéri, ktorého správanie je nesprávne alebo nie je v súlade s jeho špecifikáciou
- Úspešný test je test, ktorý spôsobí, že systém funguje nesprávne, a tak odhalí chybu v systéme.

Vstupno-výstupný model testovania programu



Overenie a validácia (V&V)

- Verifikácia:
  „Vyrábame produkt správne?“
  - Softvér by mal zodpovedať jeho špecifikácii .

- Validácia:
  „Vyrábame správny produkt?“
  - Softvér by mal robiť to, čo používateľ skutočne vyžaduje.

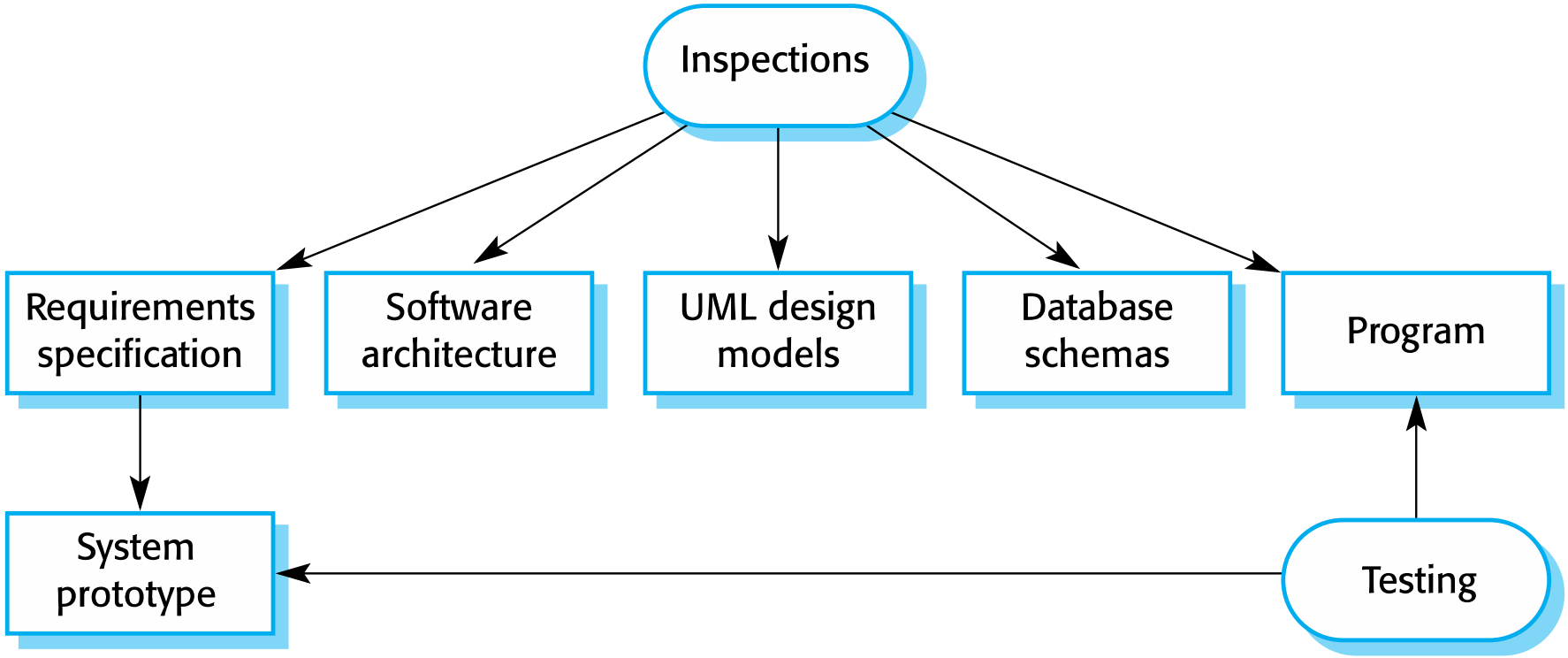
V & V ==> dôvera

- Cieľom V & V je vytvoriť dôveru, že systém je „vhodný na daný účel“.
- Závisí od účelu systému, očakávaní používateľov a marketingového prostredia
  - Účel softvéru
    - Úroveň dôvery závisí od toho, nakoľko kritický je softvér pre organizáciu.
  - Očakávania používateľov
    - Používatelia môžu mať od určitých druhov softvéru nízke očakávania.
  - Marketingové prostredie
    - Včasné uvedenie produktu na trh môže byť dôležitejšie ako hľadanie chýb v programe.

Kontroly a testovanie

- Softvérové kontroly, inšpekcie Zaoberajú sa analýzou reprezentácie statického systému s cieľom odhaliť problémy (statické overenie)
  - Môže byť doplnený nástrojovou analýzou dokumentov a kódu.
- Testovanie softvéru Zaoberá sa pozorovaním správania produktu (dynamické overovanie)
  - Systém sa vykonáva s testovacími údajmi a sleduje sa jeho prevádzkové správanie.

Kontroly a testovanie



Softvérové kontroly

- Ľudia skúmajú s cieľom objaviť anomálie a defekty.
- Inšpekcie nevyžadujú spustenie systému, takže sa môžu použiť pred implementáciou.
- Môžu byť aplikované na akúkoľvek reprezentáciu systému (požiadavky, návrh, konfiguračné údaje, skúšobné údaje, ...).
- Ukázalo sa, že sú účinnou technikou na odhaľovanie chýb programu.

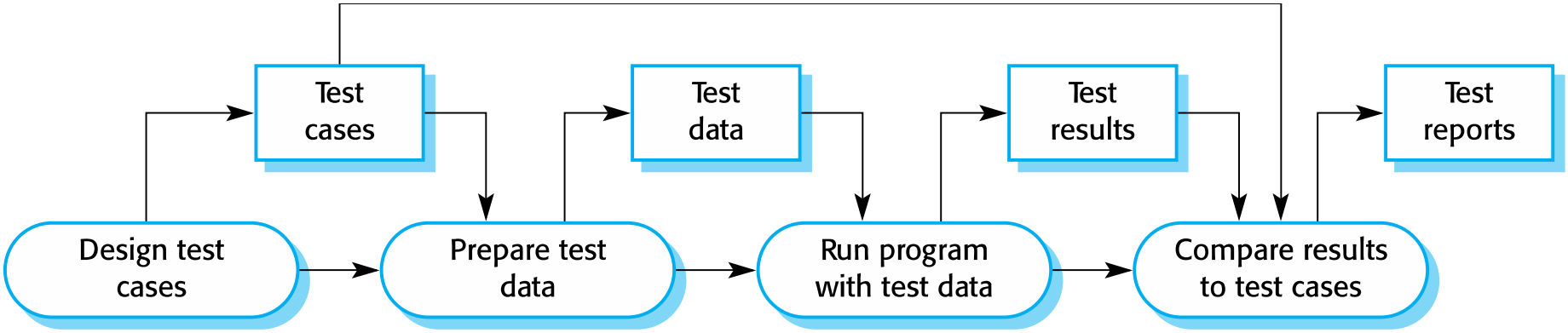
Výhody inšpekcií

- Počas testovania môžu chyby maskovať (skryť) iné chyby. Pretože kontrola je statický proces, nemusíte sa zaoberať interakciami medzi chybami.
- Neúplné verzie systému je možné kontrolovať bez dodatočných nákladov. Ak je program neúplný, musíte vyvinúť špecializované testovacie prípady na testovanie častí, ktoré sú k dispozícii.
- Okrem hľadania chýb programu môže inšpekcia zvážiť aj širšie atribúty kvality programu, ako je súlad so štandardmi, prenosnosť a udržiavateľnosť.

Kontroly a testovanie

- Inšpekcie a testovanie sa dopĺňajú a nie sú v rozpore s overovacími technikami.
- Obe by sa mali používať počas procesu V & V.
- Inšpekcie môžu kontrolovať zhodu so špecifikáciou, ale nie zhodu so skutočnými požiadavkami zákazníka.
- Inšpekcie *nemôžu* kontrolovať nefunkcionálne vlastnosti, ako je výkon, použiteľnosť atď.

Model procesu testovania softvéru



Etapy testovania

- Vývojové testovanie, kde sa systém testuje počas vývoja s cieľom odhaliť chyby a defekty.
- Release testing, kde samostatný testovací tím testuje kompletnú verziu systému pred jej vydaním používateľom.
- Používateľské testovanie, kde používatelia alebo potenciálni používatelia systému testujú systém vo svojom vlastnom prostredí.

Vývojové testovanie

- Vývojové testovanie zahŕňa všetky testovacie aktivity, ktoré vykonáva tím vyvíjajúci systém.
  - Unit testing, kde sa testujú jednotlivé programové jednotky alebo triedy objektov. Jednotkové testovanie by sa malo zamerať na testovanie funkčnosti objektov alebo metód.
  - Testovanie komponentov, kde je integrovaných niekoľko samostatných jednotiek na vytvorenie kompozitných komponentov. Testovanie komponentov by sa malo zamerať na testovanie rozhraní komponentov.
  - Systémové testovanie, kde sú niektoré alebo všetky komponenty v systéme integrované a systém je testovaný ako celok. Testovanie systému by sa malo zamerať na testovanie interakcií komponentov.

Testovanie jednotiek

- Unit testing je proces testovania jednotlivých komponentov v izolácii.
- Ide o proces testovania defektov.
- Jednotky môžu byť:
  - Jednotlivé funkcie alebo metódy v rámci objektu
  - Triedy objektov s niekoľkými atribútmi a metódami
  - Komponenty s definovanými rozhraniami používané na prístup k ich funkcionalite.

Testovanie tried objektov

- Kompletné testovacie pokrytie triedy zahŕňa
  - Testovanie všetkých operácií spojených s objektom
  - Nastavenie a skúmanie všetkých atribútov objektu
  - Cvičenie objektu vo všetkých možných stavoch.
- Dedičnosť sťažuje navrhovanie testov tried/objektov, pretože informácie, ktoré sa majú testovať, nie sú lokalizované.

Automatizované testovanie

- Vždy, keď je to možné, testovanie jednotiek by malo byť automatizované, aby sa testy vykonávali a kontrolovali bez manuálneho zásahu.
- Pri automatizovanom testovaní jednotiek sa používa na písanie a spúšťanie testov programu testovací automatizačný rámec (napríklad JUnit).
- Rámce testovania jednotiek poskytujú všeobecné testovacie triedy, ktoré rozšírite na vytváranie špecifických testovacích prípadov.

Skladba automatizovaného testovania

- Časť nastavenia, kde inicializujete systém pomocou testovacieho prípadu, konkrétne vstupov a očakávaných výstupov.
- Časť volania, kde voláte objekt alebo metódu, ktorá sa má testovať.
- Časť tvrdenia, kde porovnávate výsledok hovoru s očakávaným výsledkom. Ak sa tvrdenie vyhodnotí ako pravdivé, test bol úspešný, ak je nepravdivý, potom zlyhal.

Výber jednotkových testovacích prípadov

- Testovacie prípady by mali ukázať, že keď sa testovaný komponent používa podľa očakávania, tak robí to, čo má robiť.
- Ak sú v komponente chyby, mali by sa odhaliť pomocou testovacích prípadov.
- To vedie k 2 typom testovacieho prípadu jednotky :
  - Prvý z nich by mal odrážať normálnu činnosť programu a mal by ukazovať, že komponent funguje podľa očakávania.
  - Iný typ testovacieho prípadu by mal byť založený na skúsenostiach z testovania, kde sa vyskytujú bežné problémy. Mal by používať abnormálne vstupy na kontrolu, či sú správne spracované a nespôsobujú pád komponentu.

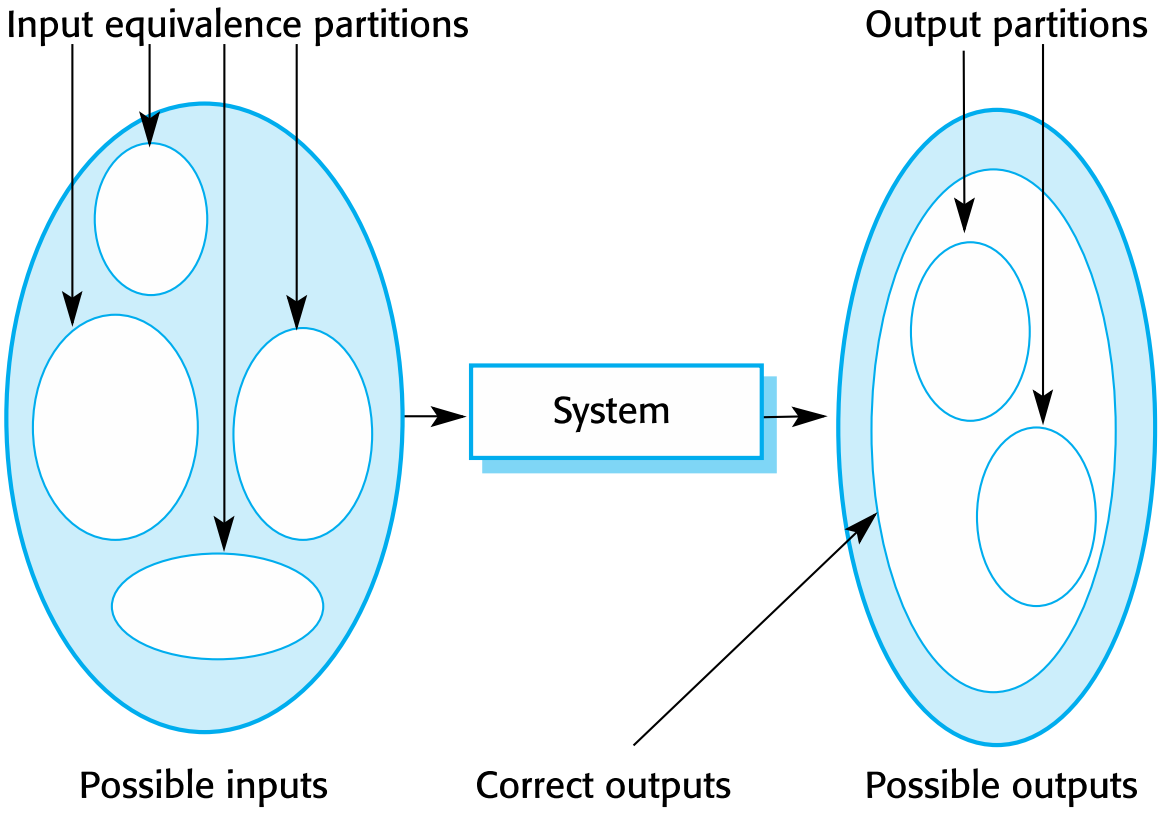
Stratégie testovania

- Testovanie partícií, kde identifikujete skupiny vstupov, ktoré majú spoločné charakteristiky a mali by byť spracované rovnakým spôsobom.
  - Testy by ste si mali vybrať z každej z týchto skupín.
- Testovanie založené na pokynoch, kde na výber testovacích prípadov používate pokyny na testovanie.
  - Tieto pokyny odrážajú predchádzajúce skúsenosti s typmi chýb, ktoré programátori často robia pri vývoji komponentov.

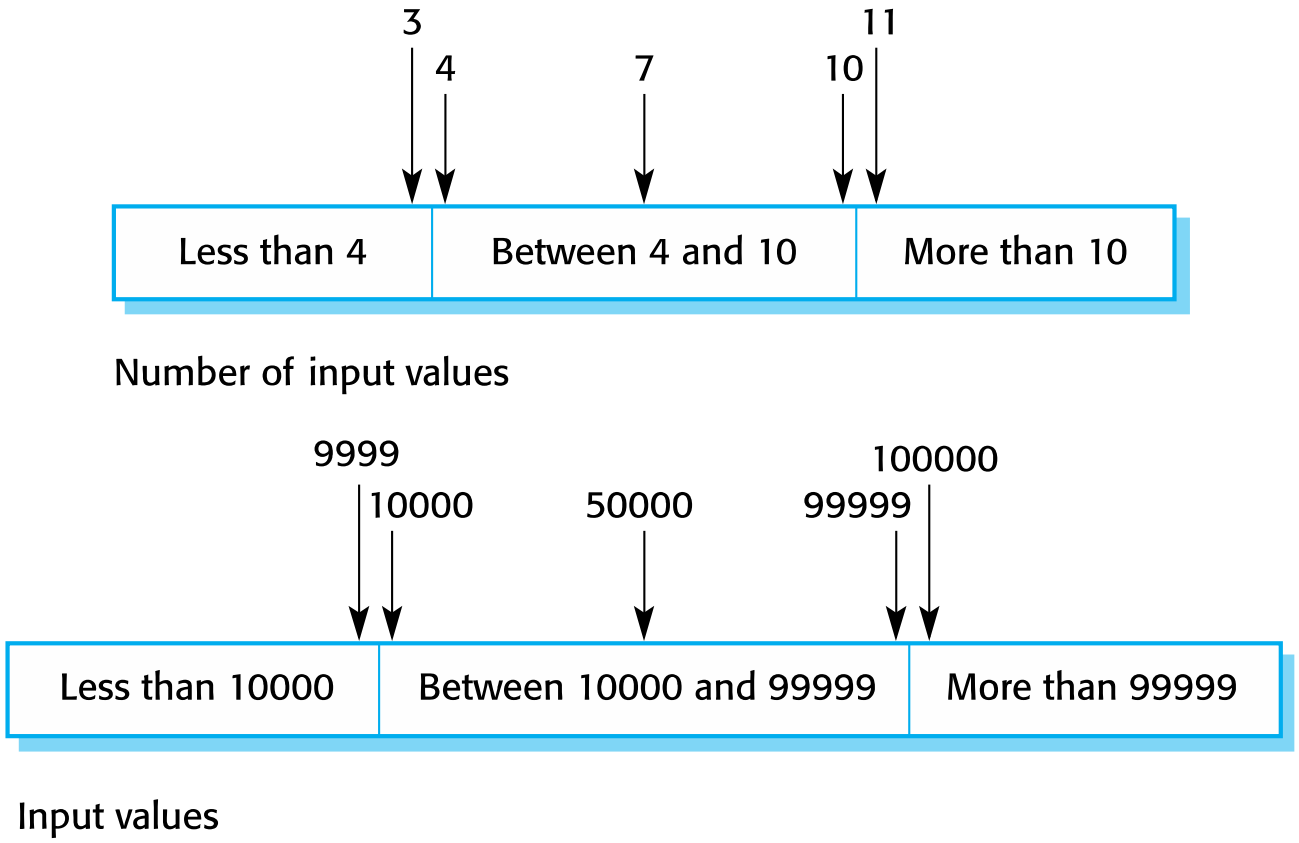
Testovanie partícií

- Vstupné údaje a výstupné výsledky často spadajú do rôznych tried, kde sú všetky prvky triedy príbuzné.
- Každá z týchto tried je ekvivalentnou partíciou alebo doménou, kde sa program správa rovnakým spôsobom pre každý prvok triedy.
- Testovacie prípady by sa mali vybrať z každej partície.

Rozdelenie na partície



Rozdelenie na partície



Pokyny na testovanie (sekvencie)

- ✧ Použite sekvencie rôznych veľkostí v rôznych testoch.
  - Testujte softvér pomocou sekvencií, ktoré majú iba jednu hodnotu.
  - Test so sekvenciami nulovej dĺžky.
- ✧ Odvodzujte testy tak, aby boli prístupné prvé, stredné a posledné prvky sekvencie.

Všeobecné pokyny na testovanie

- Zvoľte vstupy, ktoré prinútia systém generovať všetky chybové hlásenia
- Navrhnite vstupy, ktoré spôsobia pretečenie vstupných vyrovnávacích pamätí
- Opakujte rovnaký vstup alebo sériu vstupov niekoľkokrát
- Vynútiť generovanie neplatných výstupov
- Vynútiť, aby boli výsledky výpočtu príliš veľké alebo príliš malé.

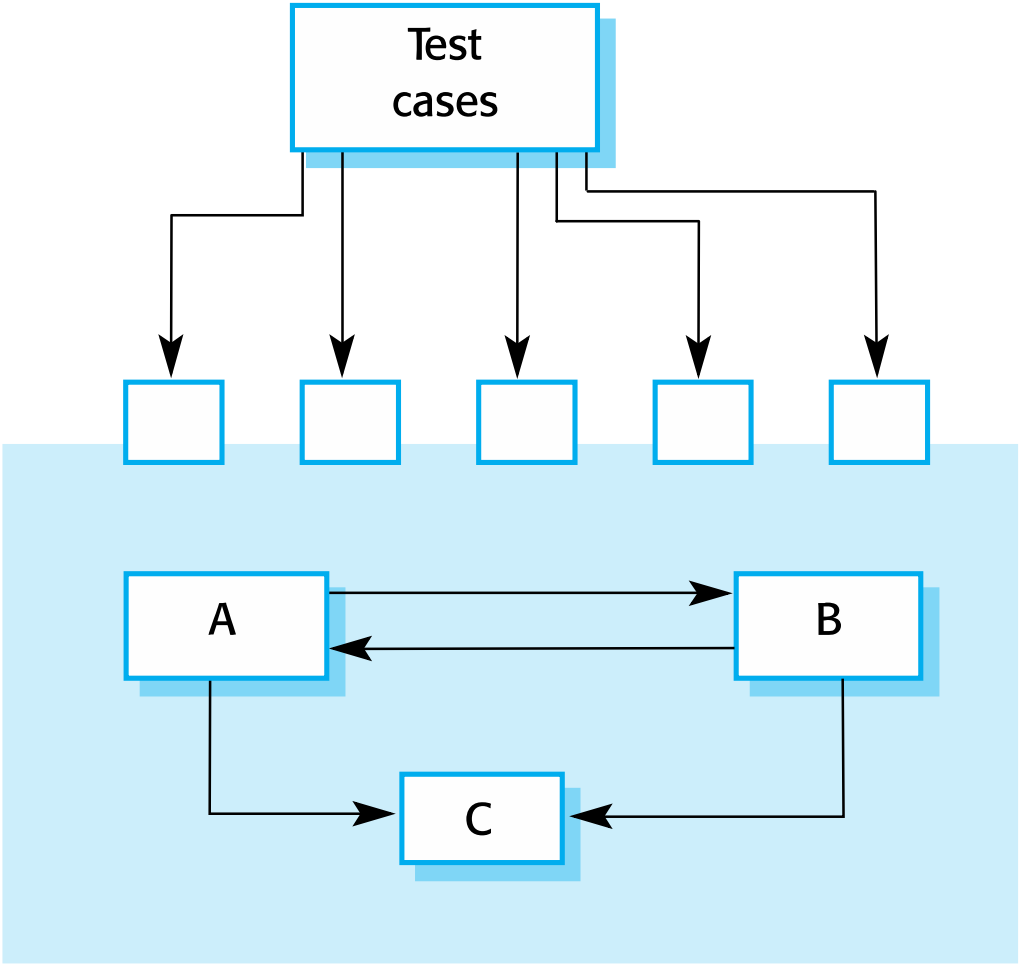
Testovanie komponentov

- Softvérové komponenty sú často zložené komponenty, ktoré sa skladajú z niekoľkých vzájomne pôsobiacich objektov.
- K funkcionalite týchto objektov pristupujete cez definované rozhranie komponentov.
- Testovanie komponentov by sa preto malo zamerať na preukázanie, že rozhranie komponentu sa správa podľa svojej špecifikácie.
  - Môžete predpokladať, že testy jednotiek na jednotlivých objektoch v rámci komponentu boli dokončené.

Testovanie rozhrania

- Cieľom je odhaliť chyby spôsobené chybami rozhrania alebo neplatnými predpokladmi o rozhraniach.
- Typy rozhraní
  - Rozhrania parametrov: Údaje prenášané z jednej metódy alebo procedúry do druhej.
  - Rozhrania zdieľanej pamäte: Blok pamäte je zdieľaný medzi procedúrami alebo funkciami.
  - Procedurálne rozhrania: Subsystém zapuzdruje súbor procedúr, ktoré môžu vyvolať iné podsystémy.
  - Rozhrania odovzdávania správ: Podsystémy požadujú služby od iných podsystémov

Testovanie rozhrania



Chyby rozhrania

✧ Zneužívanie rozhrania

- Volajúci komponent volá iný komponent a urobí chybu pri používaní svojho rozhrania, napr. parametre v nesprávnom poradí.

✧ Nepochopenie rozhrania

- Volajúci komponent vkladá predpoklady o správaní volaného komponentu, ktoré sú nesprávne.

✧ Chyby časovania

- Volaný a volajúci komponent pracujú rôznymi rýchlosťami a pristupuje sa k neaktuálnym informáciám.

Pokyny na testovanie rozhrania

- Navrhnite testy tak, aby parametre volanej procedúry boli na extrémnych hraniciach svojho rozsahu.
- Vždy testujte parametre ukazovateľa s nulovými ukazovateľmi.
- Navrhnite testy, ktoré spôsobia zlyhanie komponentu.
- Použite stresové testovanie v systémoch odosielania správ.
- V systémoch so zdieľanou pamäťou zmeňte poradie, v ktorom sú komponenty aktivované.

Testovanie systému

- Testovanie systému počas vývoja zahŕňa integráciu komponentov na vytvorenie verzie systému a následné testovanie integrovaného systému.
- Pri testovaní systému sa kladie dôraz na testovanie interakcií medzi komponentmi.
- Systémové testovanie kontroluje, či sú komponenty kompatibilné, správne spolupracujú a prenášajú správne údaje v správnom čase cez ich rozhrania.
- Testovanie systému testuje vznikajúce správanie systému.

Testovanie systému a komponentov

- Počas testovania systému môžu byť opakovane použiteľné komponenty, ktoré boli vyvinuté samostatne, a bežne dostupné systémy integrované s novo vyvinutými komponentmi. Potom sa otestuje celý systém.
- V tejto fáze môžu byť integrované komponenty vyvinuté rôznymi členmi tímu alebo podtímami. Testovanie systému je skôr kolektívny ako individuálny proces.
  - V niektorých spoločnostiach môže testovanie systému zahŕňať samostatný testovací tím bez účasti dizajnérov a programátorov.

Testovanie white-box, clear-box, glass-box

- Odvodenie testovacích prípadov podľa programovej štruktúry. Znalosť programu sa používa na identifikáciu ďalších testovacích prípadov.
- Štrukturálne testovanie – cieľom je vykonať všetky programové príkazy. Prípady použitia vyvinuté na identifikáciu systémových interakcií možno použiť ako základ pre testovanie systému.
- Testovanie cesty – cieľom je vybrať vstupy, ktoré prejdú všetkými cestami programu (cykly, príkazy if, ...) aspoň raz.

Testovanie prípadov použitia

- Prípady použitia vyvinuté na identifikáciu systémových interakcií možno použiť ako základ pre testovanie systému.
- Každý prípad použitia zvyčajne zahŕňa niekoľko systémových komponentov, takže testovanie prípadu použitia si vynúti tieto interakcie.
- Sekvenčné diagramy spojené s prípadom použitia dokumentujú komponenty a interakcie, ktoré sa testujú.

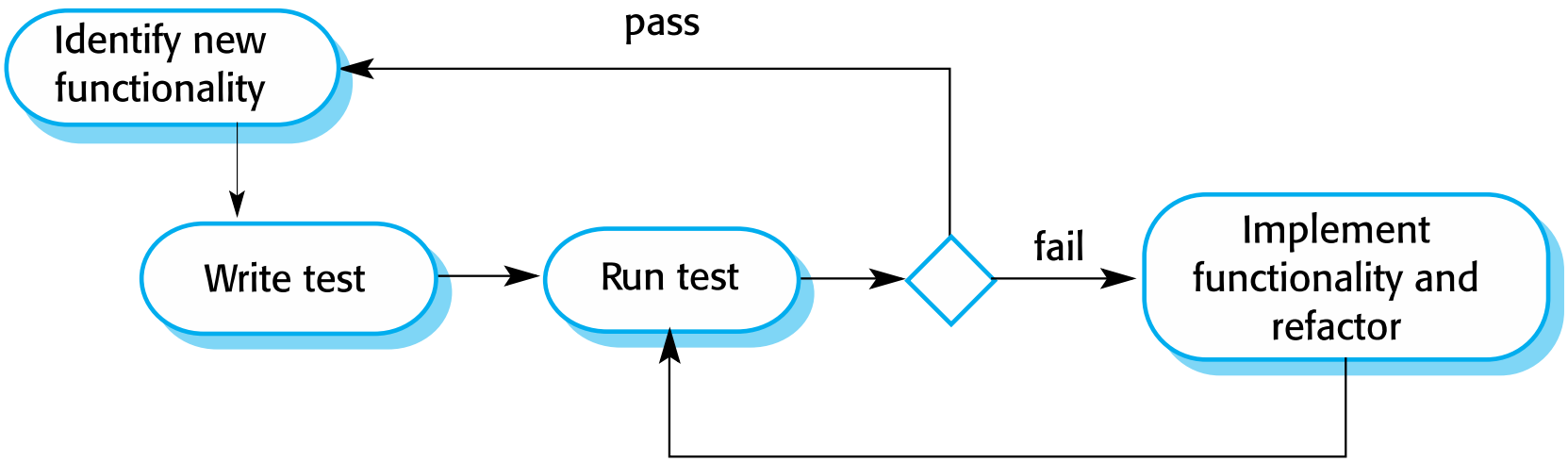
Zásady testovania

- Vyčerpávajúce testovanie systému nie je možné , takže je možné vyvinúť testovacie politiky, ktoré definujú požadované pokrytie systémových testov.
- Príklady testovacích pravidiel:
  - Všetky systémové funkcie, ku ktorým sa pristupuje cez menu, by sa mali otestovať.
  - Kombinácie funkcií (napr. formátovanie textu), ku ktorým sa pristupuje cez rovnaké menu, musia byť otestované.
  - Ak je poskytnutý vstup používateľa, všetky funkcie sa musia otestovať so správnym aj nesprávnym vstupom.

Testom riadený vývoj

- Testom riadený vývoj (TDD) je prístup k vývoju programu, v ktorom prelínate testovanie a vývoj kódu.
- Testy sa píšu pred kódom a „absolvovanie“ testov je kritickým hnacím motorom vývoja.
- Kód vyvíjate postupne spolu s testom tohto prírastku. Neprejdete na ďalší prírastok, kým kód, ktorý ste vytvorili, neprejde testom.
- TDD bol predstavený ako súčasť agilných metód, ako je extrémne programovanie. Dá sa však použiť aj v plánom riadených procesoch.

Testom riadený vývoj - najprv testovanie



Aktivity procesu TDD

- Začnite identifikáciou potrebného prírastku funkčnosti. To by malo byť zvyčajne malé a implementovateľné v niekoľkých riadkoch kódu.
- Napíšte test pre túto funkčnosť a implementujte ho ako automatický test.
- Spustite test spolu so všetkými ostatnými testami, ktoré boli implementované. Spočiatku ste neimplementovali funkciu, takže nový test zlyhá.
- Implementujte funkčnosť a znova spustite test.
- Keď všetky testy prebehnú úspešne, prejdete k implementácii ďalšej časti funkčnosti.

Výhody vývoja riadeného testami

- Pokrytie kódu
  - Každý segment kódu, ktorý napíšete, má aspoň jeden priradený test, takže každý napísaný kód má aspoň jeden test.
- Regresné testovanie
  - Sada regresných testov sa vyvíja postupne s vývojom programu.
- Zjednodušené ladenie
  - Keď test zlyhá, malo by byť zrejmé, kde je problém. Novo napísaný kód je potrebné skontrolovať a upraviť.
- Systémová dokumentácia
  - Samotné testy sú formou dokumentácie, ktorá popisuje, čo by mal kód robiť.

Regresné testovanie

- Regresné testovanie testuje systém, aby sa skontrolovalo, či zmeny „neporušili“ predtým fungujúci kód.
- V procese manuálneho testovania je regresné testovanie drahé, ale s automatickým testovaním je jednoduché a priamočiare. Všetky testy sa opakujú pri každej zmene v programe.
- Testy musia prebehnúť „úspešne“ pred vykonaním zmeny.

Testovanie vydania

- Testovanie vydania je proces testovania konkrétneho vydania systému, ktorý je určený na použitie mimo vývojového tímu.
- Primárnym cieľom procesu testovania vydania je presvedčiť dodávateľa systému, že je dostatočne dobrý na použitie .
  - Testovanie vydania preto musí preukázať, že systém poskytuje svoju špecifikovanú funkčnosť, výkon a spoľahlivosť a že pri bežnom používaní nezlyhá.
- Testovanie vydania je zvyčajne proces testovania čiernej skrinky, kde sú testy odvodené iba zo špecifikácie systému.

Testovanie vydania a testovanie systému

- Testovanie vydania je forma testovania systému.
- Dôležité rozdiely :
  - Za testovanie vydania by mal byť zodpovedný samostatný tím, ktorý sa nezúčastnil na vývoji systému.
  - Testovanie systému zo strany vývojového tímu by sa malo zamerať na odhaľovanie chýb v systéme (testovanie defektov). Cieľom testovania vydania je skontrolovať, či systém spĺňa jeho požiadavky a je dostatočne dobrý na externé použitie (testovanie platnosti).
- Testovanie založené na požiadavkách zahŕňa preskúmanie každej požiadavky a vypracovanie testu alebo testov pre ňu.

Testovanie výkonu

- Súčasťou testovania vydania môže byť testovanie vznikajúcich vlastností systému, ako je výkon a spoľahlivosť.
- Testy by mali odrážať profil používania systému.
- Testy výkonu zvyčajne zahŕňajú plánovanie série testov, pri ktorých sa záťaž neustále zvyšuje, až kým sa výkon systému nestane neprijateľným.
- Záťažové testovanie je forma testovania výkonu, pri ktorej je systém zámerne preťažený, aby sa otestovalo jeho chybové správanie.

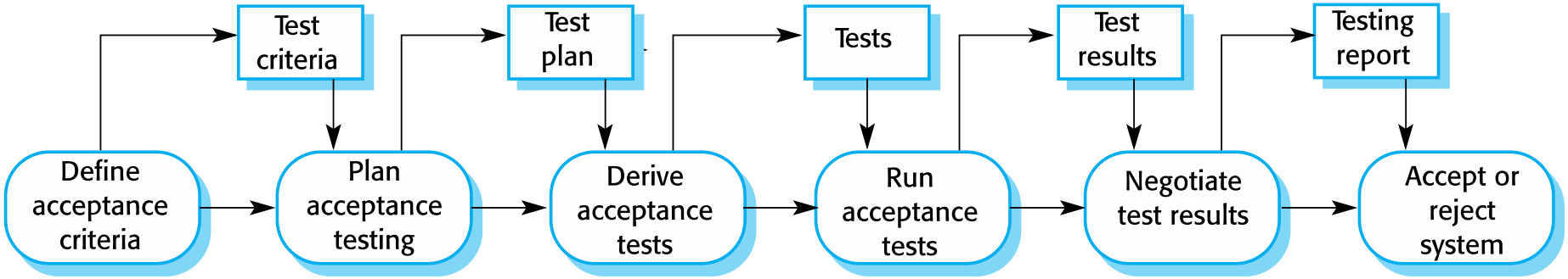
Používateľské testovanie

- Používateľské alebo zákaznícke testovanie je fázou testovacieho procesu, v ktorej používatelia alebo zákazníci poskytujú vstupy a rady týkajúce sa testovania systému.
- Používateľské testovanie je nevyhnutné, aj keď sa vykonalo komplexné testovanie systému a vydania.
  - Dôvodom je, že vplyvy z pracovného prostredia používateľa majú veľký vplyv na spoľahlivosť, výkon, použiteľnosť a robustnosť systému. Tieto nemožno replikovať v testovacom prostredí.

Typy používateľského testovania

- Alfa testovanie
  - Používatelia softvéru spolupracujú s vývojovým tímom na testovaní softvéru na stránke vývojára.
- Beta testovanie
  - Používateľom je k dispozícii vydanie softvéru, ktoré im umožní experimentovať a upozorniť na problémy, ktoré zistia u vývojárov systému.
- Akceptačné testovanie
  - Zákazníci testujú systém, aby sa rozhodli, či je pripravený na prijatie od vývojárov systému a nasadenie v prostredí zákazníka. Primárne pre zákazkové systémy.

Proces akceptačného testovania



Etapy v procese akceptačného testovania

- ✧ Definujte kritériá prijatia
- ✧ Naplánujte akceptačné testovanie
- ✧ Odvodiť akceptačné testy
- ✧ Spustite akceptačné testy
- ✧ Vyjednať výsledky testov
- ✧ Systém odmietnuť/prijať

Agilné metódy a akceptačné testovanie

- Pri agilných metódach je používateľ/zákazník súčasťou vývojového tímu a je zodpovedný za rozhodovanie o prijateľnosti systému.
- Testy sú definované používateľom/zákazníkom a sú integrované s ostatnými testami tak, že sa spúšťajú automaticky pri vykonaní zmien.
- Neexistuje žiadny samostatný proces akceptačného testovania.
- Hlavným problémom je, či používateľ v tíme je alebo nie je „typický“ a môže zastupovať záujmy všetkých zainteresovaných strán systému.

Treba vedieť… (1)

- Terminológia a metódy:
  - Verification & Validation (V&V)
  - Defect testing, Validation testing
  - Inspection (static), Testing (dynamic)
  - Debugging
  - White-box (=glass-box = clear-box) metódy - kód je známy: structural testing (all statements), path testing (all paths)
  - Black-box metódy: partitioning, guideline-based testing
- Development testing
  - Unit testing (UIT) – functions, objects
  - Interface testing
  - Component testing

Treba vedieť… (2)

- System testing
  - Integration testing = System integration testing (SIT)
    - metódy: Incremental integration testing (=Regression)
  - Release testing
    - metódy: Requirements-based testing, Performance testing, Stress testing
- User testing
    - metódy: Alpha testing & Beta testing, Use-case testing
  - User acceptance testing (UAT)
- Viď tiež prednáška 2: V-model, vstupy pre testovanie + kto vykonáva jednotlivé testy (developer, tester, user)

Fázy testovania v SW procese (V-model)