TVS – zkouška 2010/2011

Vypracoval: Radek Nguyen

1. Kvalita – koncept, filosofie a systémy, UML

(a) Statistika softwarových projektů.

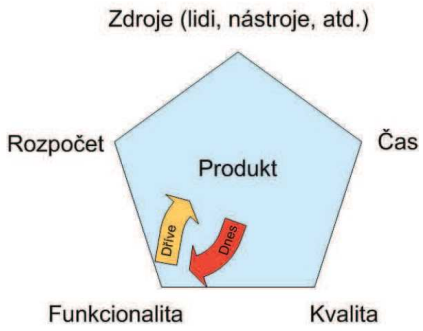
- jenom 1 ze 6ti úspěšných SW projektů (r.1994),
- projekty předávány za dvojnásobnou cenu než dohodnuto,
- projekty se předávají za dvojnásobně dlouhou dobu, než se plánuje.
- dodání projektu za delší dobu nebo za vyšší ceny,
- 1 ze 3 přerušena.
- Zavedení metodologie řízení se daří zvýšit procenta úspěšných SW projektů na poměr 1 ze 3
- komplexita SW stoupá
- poměr testerů oproti vývojářům se mění z 1:9 až na 2:3 (někdy až do 10:1)
- až polovina chyb je v požadavcích, čtvrtina v návrhu, jen 7% v implementaci

(b) Požadované vlastnosti softwarových procesů.

jasné definované cíle, požadavky, milníky, plánování prostředků (čas, rozsah, cena, kvalita, lid. zdroje), zkušené řízení podporované top managementem, přiměřené odhady, zahrnutí koncových uživatelů, standardní procesní infrastruktura, metodika, zainteresované a odpovědné strany, komunikace

obecné prvky modelu – koncept, požadavky, návrh, implementace, testování, nasazení, údržba

- přes stěnu, vodopádový, spirálový – risková analýza, prototyp, simulace, návrh, plánování, revize, rozhodnutí, U-model, tunelový, iterativní
- risk u iterativní se zmenšuje časem, oproti vodopádové, kdy až do závěru je stálé veliké riziko



!!!proces testování: žádá se, stěžuje se na, diskutuje se (co, kdy, kdo)

vlastnosti:

Opakované použití: testovací metodika by neměla být vyvíjena pouze pro jediný projekt.

Flexibilita: vyjádření nových konceptů, návrhových šablon (framework).

Adaptabilita: (odolnosti vůči změnám v SW) malé modifikace v implementaci softwaru by měly být pokryty automatizovaně.

Komplexita: pokrytí dostatečné části testovacích případů je často za možnostmi manuální přípravy.

Údržba: potřebné úsilí je nepřímo úměrné flexibilitě a adaptabilitě, přímo úměrné komplexitě testovaného produktu. **Prezentace stavu:** dokumentace pravidelně obnovovaná, např. WWW stránky.

Nástroje: integrovaná řešení adresující výše uvedené položky.

Cena/čas: efektivita vyjádřená pomocí výše uvedených položek.

Proveditelnost: trh/cena/čas/zdroje/kvalita efektivnost.(komplexní odpověď vyhodnocené podle základních atributů projekt management)

Nepřetržitý běh: rychlá odezva, několik fází (zahořovací, regresní, dokumentace pravidelně obnovovaná) **Zásady**: kvalita určuje úspěch testovací úsilí, zabraň migraci defektů již v počátku vývoje, použití testovací nástroje, určení odpovědnosti, profesní přístup, positivní postoj týmu

(c) Známé bugy historie, jejich shrnutí s hlediska testování. Typické chyby softwarových projektů.

selhání rakety Arian, radiace Therac, satelity, F16 simulace, jaderné elektrárny

velikost vektoru počítána jako součet složek, modul byl napsán studentem na praxi

- znovu-testování zamítnuto kvůli nedostatku času

- chybně rozložené náklady, chybné požadavky, chyby v kódu (nechráněný převod), změna specifikací vs opakované využití
- chyba v striktním oddělení vývoje a jeho testování
- uživatelské rozhraní nedbá na bezpečnost, složitý návrh, nedostatečné testování, problémy paralelních a souběžných systémů

(d) Koncept kvality, proaktivita a reaktivita.

Kvalita je:

- charakteristika, která definuje základní podstatu věcí či jejích význačných vlastností
- mírou souhlasu s požadavky
- souhrn vlastností a charakteristik produktu či služby, týkající se schopnosti uspokojit určené nebo vyplývající potřeby
- ztráta, kterou produkt způsobí po jeho dodání, způsobená funkční změnami a škodlivými účinky mimo těch (které vyplývají z vlastní funkce)
- ověření plnění požadavků (FURPS), IKIWISI, pohled zákazníka, krátkodobá (výkonnost), dlouhodobá (spolehlivost)
- ideální, pokud produkt poskytuje cílenou výkonnost kdykoliv, za všech podmínek, po celou dobu životnosti a nemá žádné škodlivé postranní efekty
- externí = funkčnost (správnost, spolehlivost, použitelnost, integrita)
- vnitřní = efektivita, testovatelnost, dokumentace, struktura
- budoucí = adaptabilita (flexibilita, znovupoužitelnost, údržba)

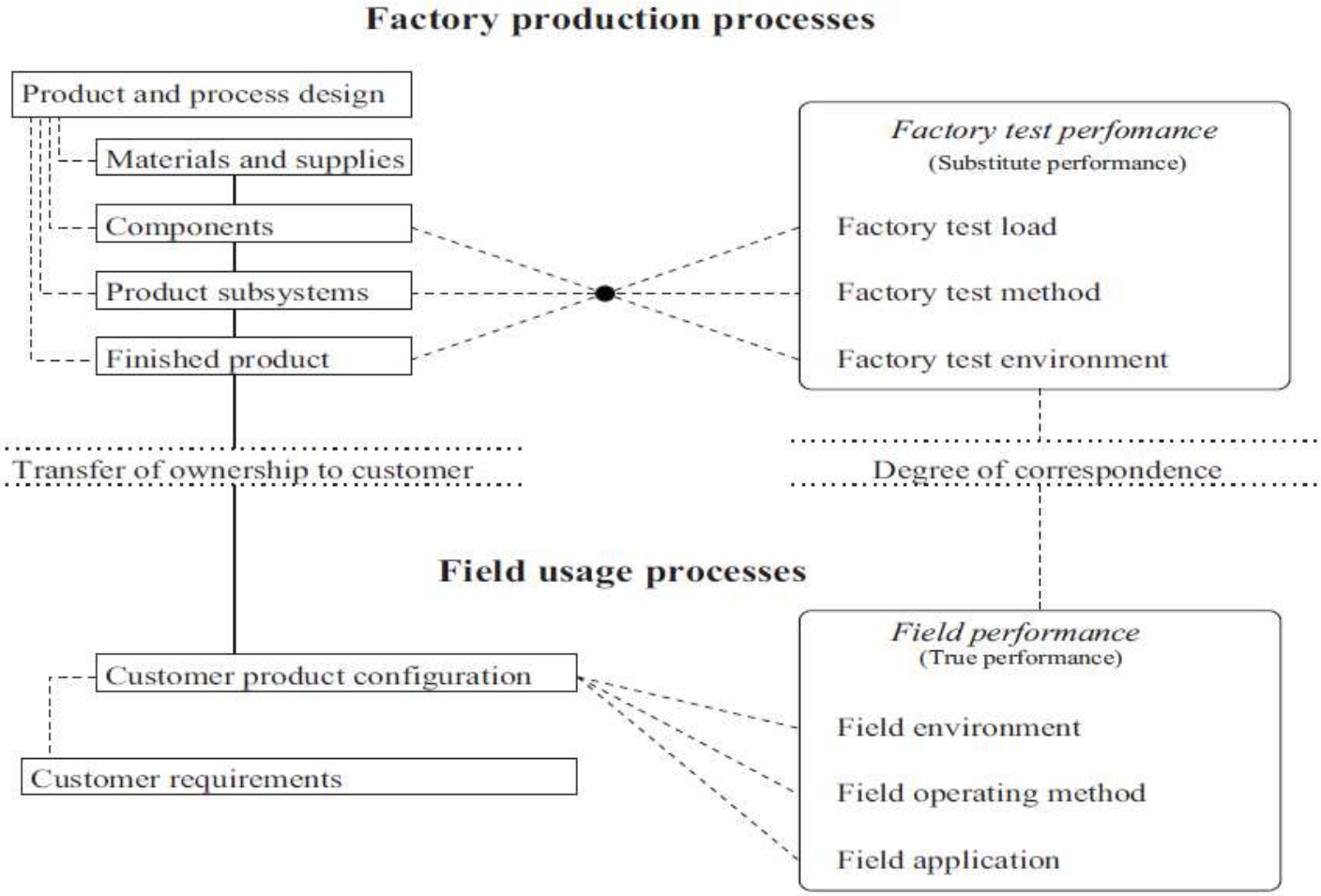
reaktivita = detekce a řešení problémů, které již nastaly. Vyhodnocení tradičních ztrát pomocí statické analýzy známých nashromážděných informací, vede k omezení ztrát
proaktivita = prevence problémů, hledání příčiny a následků, riskové analýzy, zkušenosti, spekulace, zdůvodnění prevence. Vede k rychlejší vývoji, vyhnutí se ztrátám

(e) Definice testování softwaru.

Testování je **proces:**

- MĚŘENÍ KVALITY SW
- určení věrohodnosti, že systém dělá to, co se očekává.
- s úmyslem nalézt chyby.
- s cílem vyhodnocení schopnosti plnění požadavků systému.
- kontroly oproti specifikacím
- určení míry akceptace uživatelem
- ověření připravenosti systému k použití
- získání důvěry, že systém je správný
- demonstrace bezchybnosti (správnosti) systému
- hodnocení schopnosti = porozumění omezení výkonnosti systému (určení toho, co systém není schopen dělat)
- verifikace dokumentace systému
- vzorkování, určení pravděpodobnosti selhání (míry nespokojenosti zákazníka)

(f) Vztah mezi kvalitou pociťovanou zákazníkem a kvalitou měřenou producentem softwaru



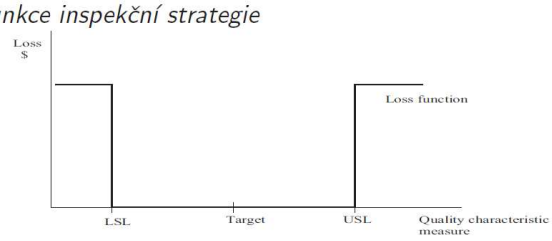
- vnější kvalita softwarového produktu se hodnotí až po jeho vytvoření a hlavně zákazníkem (software je spuštěn a hodnocen jako součást systému v produkčním prostředí a podroben *skutečnými* požadavky zákazníka)
- kvalita meziproduktů a produktu, která předpovídá vnější kvalita (navíc zohledňují, kromě funkčním požadavkům a jiné, i požadavky na vnitřní kvalitu)

(g) Taguchiho přístup, ztrátová funkce.

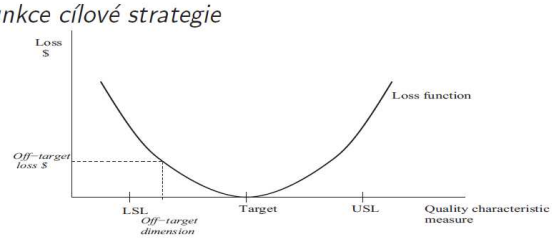
Kvalita je ztráta, kterou produkt způsobí po jeho dodání, způsobená funkční změnami a škodlivými účinky mimo těch, které vyplývají z vlastní funkce.

cílem QA je produkt s malými odchylkami (poloha a rozptyl). Produkce **řízené inspekcí**(odpad nesplňující požadavky) **vs cílem** (six-sigma, 2 defekty na miliard)

Ztrátová funkce inspekční strategie



Ztrátová funkce cílové strategie



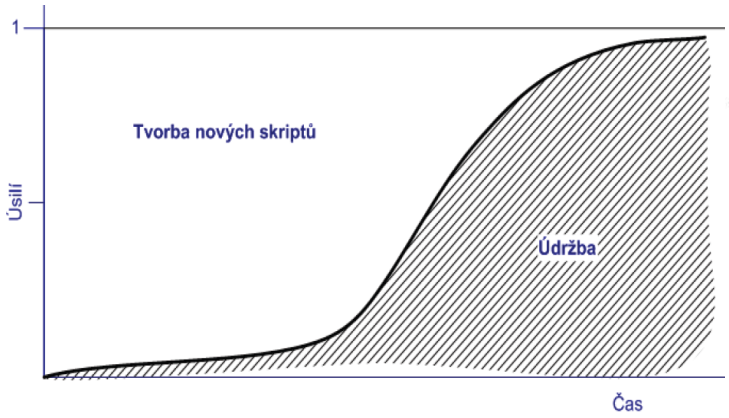
ztrátová fce: L(produkt) = ***k(produkt – index_výkonnosti)^2***

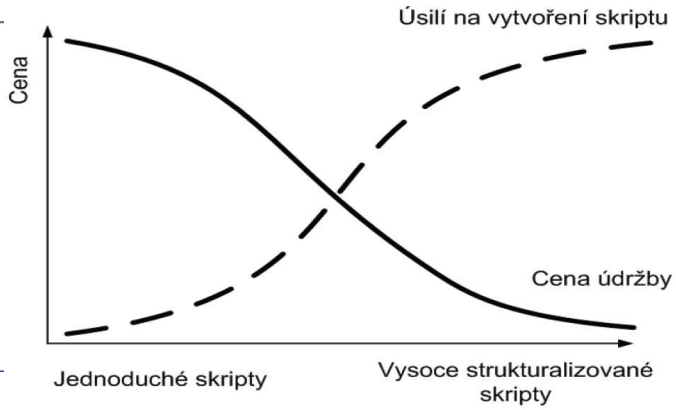
k je spočítaná jako poměr ztrátu způsobenou danou odchylkou – kvadratická aproximace Taylorovy řady

staré nástroje – diagram příčin a následků (rybí kost), kontrolní seznam, paretova analýza, histogram, diagram rozptylu

nové nástroje – diagram příbuznosti, dům kvality (maticový diagram), relační diagram

(h) Co (ne)lze očekávat při automatizaci testovacího procesu.





lze očekávat (výhody) – regrese testování, častější testování, provedení obtížné testy pro manuální činnost, lepší využití prostředků (night-buildy), konzistence (opakovatelnost), znovupoužití, kratší timeToMarket, kombinatorické pokrytí, uplatnění při vyšší komplexnosti a integrace systému. Automatizace vykonání, analýzy, návrhu, řízení, vyhodnocení, úsilí při návrhu automatizace a údržba jsou nepřímo úměrné.
nelze očekávat – vše je automatizované (příprava, údržba), úplné nahrazení manuálních testů, flexibilitu a adaptabilitu, vysoké detekční schopnosti (prvotní detekce vs regresní ověření při změně)

2. Optimalizace testování

(a) Princip párového testování.
bezestavové chování fce, inicializace objektu, parametry
ideální všechny kombinace je časově nemožné (kombinatorická exploze)
Praktický testovací plán – selhání jsou způsobena interakcí pouze několika parametrů, testování kombinacemi pokrývající všechny možné k-tice,
kombinace pokrytí k-tice(1,2,3), cílem je redukce (optimalizace) počtu testovacích případů a přitom otestovat to, co je potřeba
testovací plán pomocí ortogonálního pole nebo projektivní rovina(latinský čtverec)
Příklad

- 5 parametrů (faktor),
- 17 hodnot pro každý parametr (úroveň)

Ideální testovací plán

- $1.419.857 = 17^5$ kombinací
- **230 dnů testování**

Praktický testovací plán:

- Nezávislé parametry: 17 kombinací
- Párová závislost parametrů: 10 = 4 + 3 + 2 + 1 parametrových párů,
- $289 = 17^2$ kombinací hodnot pro každý pár,
- 2890 = 289*10 párů parametrových hodnot,
- 289 kombinací obsahující všechny páry,
- **30 minut**

(b) Postup optimalizace metodou ortogonálních polí.
určení faktoru (entita, parametr, na kterém závisí testovací případ) a úrovně (konečná množina hodnot faktoru)
OATS, použití standardních polí (nejmenší pole, který řeší problém),
Ortogonální pole je matice velikosti m x n, ve které při testování parametrických párů jednotlivé sloupce odpovídají faktorům (parametrům funkce, případně jiným hodnotám, na kterých závisí

testovací případ) a Řádek v ortogonálním poli se mapuje podle kódování, a odpovídá jednotlivým testovacím případům.
Ortogonální pole jsou označována podle toho, kolik úrovní (významných hodnot) umožňují jednotlivým faktorům.
Pro použití této techniky je zapotřebí příklad nejprve zakódovat. Nalezneme všechny faktory (parametry příkladu) a zjistíme, kolik mají úrovní. Zakódování úrovně(domény) faktoru jako čísla.
Poté nalezneme vhodné ortogonální pole. Vybrané pole musí umožňovat alespoň tolik faktorů, kolik má náš případ. Dále je nutné, aby jednotlivé faktory měly dostatečný počet úrovní.
Nyní můžeme přejít k samotnému kódování. Každému parametru naší úlohy přiřadíme jeden sloupec. Každé významné hodnotě přiřadíme jednu úroveň – pokud má sloupec více úrovní než potřebujeme, tak můžeme některé významné hodnotě přiřadit více úrovní, tím tuto hodnotu otestujeme více než ostatní (ale stále platí, že otestujeme každou parametrickou dvojici alespoň jednou).

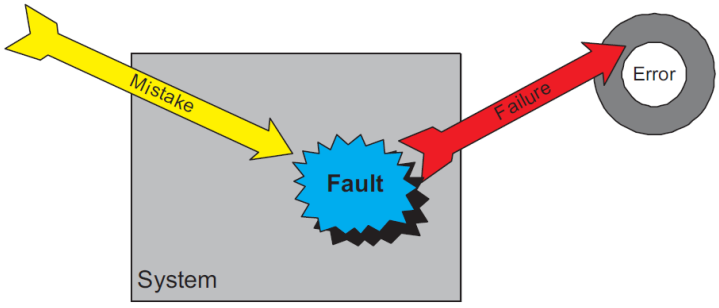
(c) Postup optimalizace metodou latinských čtverců.
určení faktoru a úrovně – z toho se určí N (jak?), N je max(faktor-2, úroveň)
vzorec pro výpočet l. čt. Ak(i,j) =[k(i-1)+(j-1)] mod N
použití souřadnice (řádek, sloupec) pro zakódování faktoru

(d) Vlastnosti latinských čtverců a jejich ortogonálních párů.
je matice n řádků a n sloupců, prvek má hodnotu 1 až n tak, ze se v žádném sloupci nebo řádku nevyskytuje více než jednou.

- mezi dvěmi párově ortogonálními latinskými čtvercem prvek (element) s danou souřadnici vystupuje v relaci s prvkem druhého čtverce právě jednou
- pro jakoukoliv mocninu prvočísla N existuje N - 1 párově ortogonálních latinských čtverců velikosti N x N (2,4,8,9..., ale ne pro 6=2*3)

3. Klasická metodologie testování

(a) Softwarová chyba, jejich distribuce.



- je to prezentace toho, že program nedělá něco nepředpokládaného
- míra toho, kdy program přestává být užitečný (lidem)
- je to nesouhlas mezi programem se specifikací, ale jen když jsou správné
- distribuce od Pochybení (Mistake), které produkuje nesprávný krok, tedy Vada (fault), potenciálně vede k Selhání (failure, nemusí, např. mrtvý kód), tedy nesprávný výsledek, Chyba (error) je jeho kvantitativní vyjádření, na kolik je výsledek nesprávný.
- cena nalezení a opravy chyb je čím dříve, tím levnější
- polovina chyb vzniká požadavků, čtvrtina chyb vzniká návrhu

(b) Úrovně testování. Typologie testování.
Úrovně testování

- jednotek – funkční a strukturní požadavky na úrovni jednotky,
- komponent – požadavky na úrovni komponenty (několik jednotek),
- integrační – za předpokladu funkčních komponent testování kombinace komponent,
- systému – zabývá se problematikou chování, ke kterému dochází v plně integrovaném systému.

Typy testování

- **Formální t.** je proces provádění testovacích aktivit a hlášení výsledků testů podle odsouhlaseného testovacího plánu.

- **Akceptační t.** je formální testování prováděného za účelem stanovit, zda systém splňuje akceptační kritéria a umožňuje zákazníkovi určit zda přijme systém či nikoliv.
- **Systémové t.** je proces testování integrovaného systému za účelem ověření, zda vyhovuje specifikovaným požadavkům.
- **Regresní t.** je částečné testování s cílem ověřit, že provedené modifikace nezpůsobují nechtěné vedlejší efekty nebo že modifikovaný systém stále splňuje požadavky.
- **Hodnocení výkonnosti (zátěžové t.)** – určení dosažení efektivnosti operativní charakteristiky.

(c) Terminologie testování. Testovací plán, specifikace procedury, záznam testu.
Procedura, specifikace:

- **Požadavek** – podmínka nebo schopnost, kterou uživatel potřebuje k řešení problému nebo vyřešení úlohy.
- **Specifikace** – vyjádření množiny požadavků, kterým by měl produkt vyhovět.
- **Testovací plán** – dokument popisující zvolený přístup k zamýšleným testovacím aktivitám.
- **Testovací případ** – specifická množina testovacích dat společně s očekávanými výsledky vztažené k vybranému cílu testu.
- **Návrh testu** – výběr a specifikace množiny testovacích případů, které splnují úlohu testu nebo kritéria pokrytí.
- **Dobrý test** – nezanedbatelná pravděpodobnost detekce dosud neobjevené chyby.
- **Úspěšný test** – detekuje dosud neobjevenou chybu.

Terminologie:

- **Testovací data** – vstupní data a podmínky pro soubory asociované s daným testovacím případem.
- **Očekávané výsledky** – predikované výstupní data a podmínky souborů asociované s daným testovacím případem.
- **Orákulus** je jakýkoliv program, proces nebo objem dat, které specifikují očekávaný výsledek množiny testů, pokud jsou aplikovány na testovaný objekt.
- **Testovací procedura** – dokument definující kroky směřující k pokrytí alespon části testovacího plánu nebo běhu množiny testovacích případů.
- **Záznam testu** – chronologický záznam všech význačných podrobností testovací aktivity.
- **Platnost testu** – stupen, jak dalece test dosahuje specifického cíl
- hraniční podmínky, průzkumné testy, plán pomocí seznamy, tabulky

(d) UI chyby, chyby omezení, procesní chyby.
UI chyby

- problém s **funkčností**, jestliže:
  - nedělá něco, co by měl dělat nebo
  - to dělá nevhodně, zmatečným způsobem či neúplně,
  - lze některé operace provést obtížně,
- Konečná definice, co se „předpokládá“ od programu, žije pouze v mysli uživatele.
- Všechny programy mají problémy s funkčností vzhledem k různým uživatelům.
- **Funkční problém** je **chybou**, pokud očekávání uživatele jsou rozumná.
- **Vstupy** – Komunikace, Struktura příkazů, Chybějící příkazy
- **Výstupy**
  - Rychlost (interakce), pocit, že program pracuje pomalu
  - potřeba uživatele, smysluplné reporty, přizpůsobení podle uživatele (přesměrování – monitor, tiskárna, soubor…)

Chyby omezení

- **hraniční** podmínky (numerické, meze paměti)

- **výpočetní** chyby (aritmetika: zaokrouhlování, ořezání)
- **zpracování** výjimek (zabránění, podmínky, detekce a zpracování)

Procesní chyby

- **sekvenční**
- počáteční a jiné speciální stavy (po resetu, první použití, inicializační informace)
- chybný krok nebo zastavení programu, neovladatelný program
- **paralelní**
- chyby souběhu (závodění multiprocesorové systémy, multiprocesové programy, obtížně se opakují)
- zátěžové podmínky (velký objem dat, velký stres, limity programů – co nastane?)

(e) Chyby řízení, strukturální chyby, chyby požadavků.

řízení (vedení)

- hardware (neexistující, vytížená zařízení)
- řízení zdrojů a verzí (staré verze komponenty)
- dokumentace (důvěryhodnost pro uživatele)
- chyby testovaní (chyby testerů jsou nejčastějšími chybami objevenými při testování)

strukturální

- řízení a sekvenční: příkazy GOTO
- zpracování (vyhodnocení a výběr algoritmu)
- logiky (neporozumění sémantice logických výrazů, operátory, datové toky, řídicí toky)
- inicializační
- anomálie v toku dat (neočekávaná práce s daty)

požadavků

- neúplné, nejednoznačné požadavky a specifikace
- chybné, chybějící, nevyžádané vlastnosti
- interakce vlastností (nepredikovatelné, např. přesměrování v smyčce)
- problém s komunikací člověk-člověk (nepřesné vyjádření)

(f) Datové chyby, chyby kódu, chyby v manipulaci s pamětí.

datové

- ve specifikacích datových objektů, jejich formátu, počtu objektu a jejich počátečních hodnotách
- dynamické, statické (efekt poškození při dynamické chyby může být vzdálené od příčin), u statické známe třídy, ale ne konkrétní instance, složitá syntaxe jazyka může způsobit obtíže při detekce chyb
- údaj plní role: parametr, řízení, zdroj informace
- nedostatečná validace dat
- obsah (bitový vzor objektu), struktura (metadata datové položky), atributy (sémantika)

v kódu

- syntaktické chyby chytá překladače nebo nástroje SA (neukončený komentář, nedodržení kódovací standard)
- chybějící disciplína programátorů, špatná interpretace návrhu, složité nepřehledné řešení
- vedou k větší úsilí při údržbě

manipulace paměti

- těžko se lokalizují, ale jsou důležité
- jsou nepredikovatelné, projev vzdálené od příčiny, nahodilé projevy
- chyby hranic poli, nedefinovaný ukazatel, čtení z neinicializované paměti, únik paměti, alokace paměti

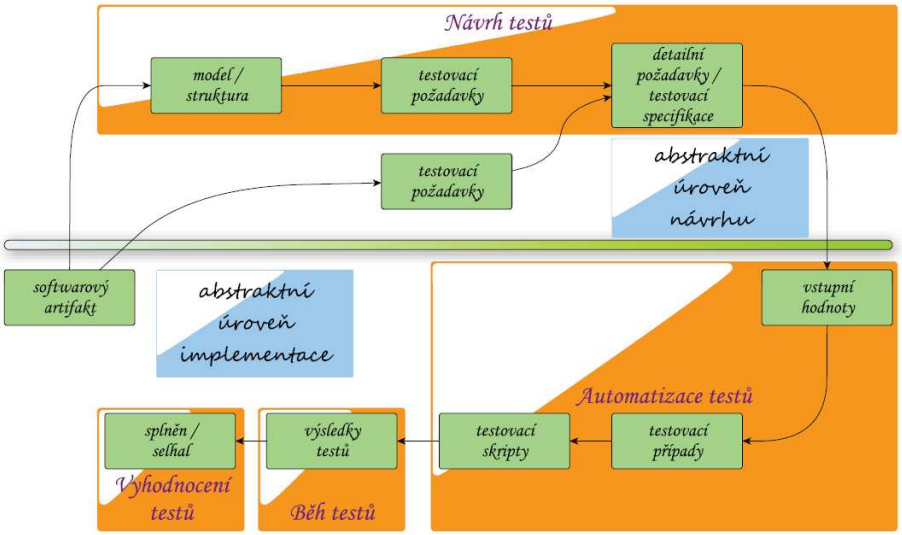
4. Strukturované testování. Testování toku řízení

(a) Obecný postup vytvoření testů pomocí grafů.

Výstavba grafu

- objekty, o které se zajímáme (uzly)
- relace mezi uzly – vztah s jinými (hrany)
- vlastnosti asociované s hranami (atributy hran)

Návrh testovaní podle modelu (MDTD – model driven test design)
artefakt, návrh (model, požadavky, detailní požadavky, test specifikace), automatizace (vstupy, test case, test script), běh (výsledky), analýza
dosažitelnost (test dostane k místu vady), infekce(program dostane do chybného stavu), propagace(chybný výstup – projev)
pozorovatelnost a řiditelnost



postup:

- definuj graf, definuj relace
- testy pro pokrytí uzlu
- navrhni testy pro pokrytí hran
- otestuj všechny atributy
- navrhni testy smyček

(b) Kritéria pokrytí.

cesta – sekvence operací, které se provedou od začátku běhu testu do jeho ukončení
Pro řídicí toky kritéria pokrytí specifikují třídu cest, které by se měly provést v rámci testování, redukce testů na proveditelnou úroveň

- Pokrytí **řádek** – provedení každé řádky kódu alespoň jednou
- Pokrytí **větví** – podmínka každého větvení musí být alespoň jednou pravdivá a jednou nepravdivá
- Pokrytí **podmínek** – zkontroluje všechny možné způsoby (kombinace), za kterých daná podmínka je pravdivá či nepravdivá
- Úplné pokrytí **cest** – vyžaduje provedení všech možných různých úplných cest (v praxi neproveditelné)
  DU-cesta je jednoduchá cesta, která je def čistá vzhledem k proměnné ***v*** a definována v ***n***
  Pro Def-Use testy (datové toky) se vyžadují vedlejší výlety def-čisté vzhledem k dané proměnné
- Pokrytí **všech definic** – nejméně 1 cestu pro každou def-cestu množiny **S=du(n,v)**
- Pokrytí **všech užití** – nejméně 1 cestu pro každý def-pár množiny **S=du(ni,nj,v)**
- Pokrytí **všech du-cest** – všechny cesty pro každý def-pár množiny **S=du(ni,nj,v)**

(c) Základní typy testovacích modelů.

TODO – graf, automat?

(d) Testovaní toku řízení – metoda hlavních cest.

Jednoduchá cesta – cesta z n_i do n_j, na které se žádný uzel neobjevuje více jak jedenkrát s výjimkou, že počáteční a koncové uzly mohou být identické.
Hlavní cesta – cesta z n_i do n_j je hlavní, jestliže je to jednoduchá cesta a není žádnou vlastní podcestou jakékoliv jiné jednoduché cesty (tj. je maximální)
Procházka: s vedlejšími výlety (návrat do stejného uzlu**), s objížďkami** (nevrátí do výchozího uzlu)

Princip algoritmu nalezení hlavních cest

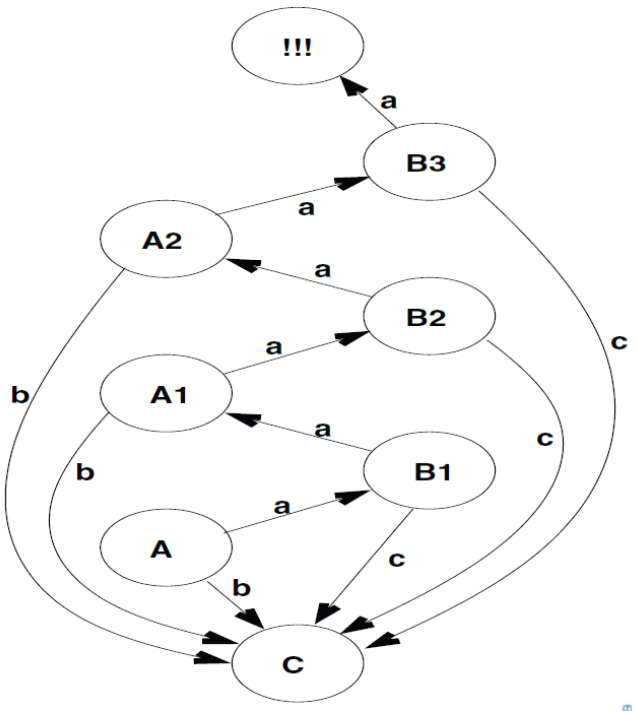
1. Nalezni cesty délky 0 (uzly).
2. Kombinuj cesty délky 0 do cest délky 1 (hrany).
3. Kombinuj cesty délky 1 do cest délky 2.
4. až zbydou jen cesty, které mají následující označení

Označení
! . . . cesta nemůže být prodloužena
* . . . cesta tvoří smyčku
konstrukce testovacích cest – musí obsahovat všechny hlavní cesty, eliminace podcest z výstupu algoritmu, kombinace pomocí nejdelších hlavních cest.

(e) Testovaní datového toku – metoda du-cest.
testy zajišťující, že hodnoty vzniklé na jednom místě jsou použity správně na jiných místech.
***Def*:** místo, kde je hodnota proměnné uložena do paměti.
***Use*:** místo, kde se přistupuje k hodnotě proměnné.
DU páry *def – use*: asociace určující přenosy hodnot.
V… množina proměnných asociovaná se softwarovým artefaktem.
def (n), def (e): podmnožina množiny proměnných V, které jsou definovány uzlem n (hranou e).
use (n), use (e): podmnožina množiny proměnných V, které jsou použity v uzlu n (na hraně e).
DU-cesta je jednoduchá cesta, která je def čistá vzhledem k proměnné ***v*** a definována v ***n***
du(ni, v) – množina všech DU-cest vzhledem k proměnné ***v*** a **začíná** v ***ni***
du(ni,nj,v) - množina všech DU-cest vzhledem k proměnné ***v*** a **začíná** v ***ni* a končí** v ***nj***
Postup:
z modelu se vytvoří množiny def(n), use(n), def(e), use(e)
po tom du(nj,v) a du(ni,nj,v)
podle kriterií pokrytí se určí potřebné testovací případy

5. Testování stavových automatů

(a) Charakteristiky stavů z pohledu testování. Skryté stavy.
Konečná množina ***Q***, obsahující podmnožiny **počátečních stavů** a **koncových stavů**
Stavy (uzly v diagramu stavového automatu) jsou **hodnoty** důležitých proměnných, **mód** chování **systému**
kód stavu (přiřazení symbolů ke stavům), **okamžitý** stav, **čítač** stavu(místo v paměti, kde je uloženo kód stavu), konečný **počet** stavů, **úplně specifikovaný systém**
pracovní stavy – systém po opuštění počátečních stavů (boot sekvence) se pohybuje v silně souvislé množině stavů



skryté stavy – předpoklad věci, které obecně neplatí (prostor stavů) – např. Inkrementace integeru
přechod je odezva systému na Input, mění stav, vydá výstupní událost

(b) Postup návrhu testů.

1. identifikuj vstupy (Input), definuj kódy vstupů
2. identifikuj stavy (Q), definuj kódy stavů
3. identifikuj výstupní akce (Output), definuj kódy výstupních akcí
4. specifikuj tabulku přechodů a tabulku výstupů

- **Přechodová fce**: F: $Q \times$ Input → Potenční podmnožiny Q
- **Výstupní fce**: G: $Q \times$ Input → Output

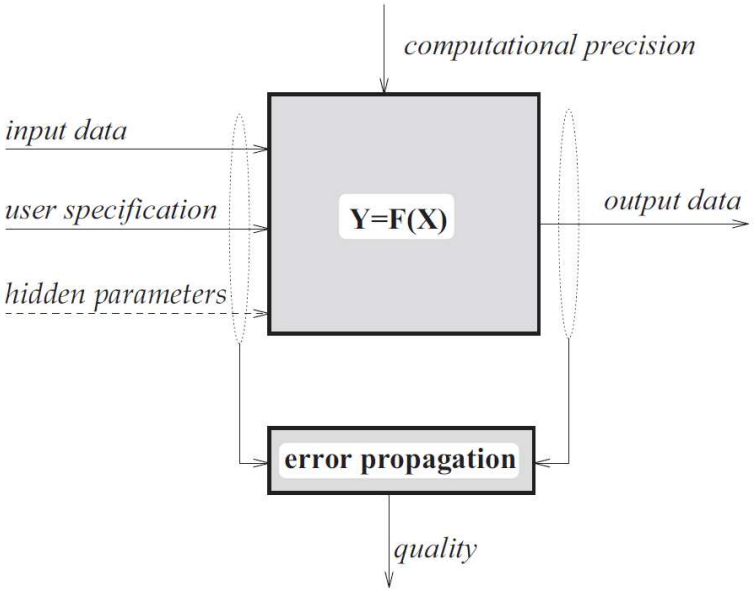
5. navrhni testy
   test **začíná** v počátečním stavu, **nejkratší cestou** se přivede k vybranému stavu, provede zadaný přechod a systém se nejkratší cestou přivede opět do počátečního stavu(tj. **okružní cesta** – cesta z stavu A do stavu A přes nějaké stavy), je potřeba určit vstupní kódy pro každý přechod i výstupní kódy asociované s přechody okružní cesty.
6. proveď testy
7. Ověříme kódování vstupů, výstupů, stavů i každý přechod

- **dosažitelnost –** existuje sekvence Input tak, že systém převede z stavu A do B
- **silně souvislý graf –** dosažitelnost z počátku pro všechny stavy
- **isolované stavy –** skupina stavů nedosažitelných z počátku (pravděpodobně chyba)
- **reset –** přechod z jakéhokoliv stavu do počátečního stavu
- **testovatelnost** – resetování, počítadlo stavů, krokování, explicitní tabulky vstupů, výstupů, přechodů

(c) Formální konstrukce testů.

- **Množina vstupní sekvencí (L) – rozlišení *2 stavů*** od sebe aplikací sekvence vstupu ***podle výstupu***
- **minimální automat** bez redundantních stavů (minimalizace automatů)
- **charakterizační množina** je množina vstupních sekvencí, pokud dokáže rozlišit jakékoliv dva stavy automatu
- **pokrytí stavu** – je ***L*** taková, že pomocí prvků v ***L*** lze dostat do jakéhokoliv žádaného stavu z počátečního stavu
- **pokrytí přechodů** – je ***T,*** která je pokrytím stavů(L) a uzavřená z hlediska pravé kompozice s množinou vstupů ***Input***
- **množina *Z* –** implementace je ve stejném stavu, který určuje specifikace
- **parametr *k* –** do dané úrovně jsou testovány skryté stavy implementace
  $Z = Input^{k} \times W \cup Input^{(k-1)} \cup \ldots \cup Input \times W \cup W$
- **množina testů: *T*×*Z*** *(kartézský součin)*

6. Statistické testování softwaru



(a) Propagace kovarianční matice - explicitní vztah. TVS 5

Y=F(X)

X – vstup, Y – výstup

Taylorův rozvoj, Jakobián

(b) Propagace kovarianční matice - implicitní vztah. TVS 5
Skalární fce F(X,Y)

(c) Přímé propagování variance.
Není potřeba odvození chybového modelu.
Substituce hodnot proměnnými strukturami (**model chyby**)

- „Symbolická derivace“ následovaná výpočtem
- bodová analýza
- výpočet okamžité chyby
- hodnocení přesnosti výpočtu
- předpoklad dostupnosti zdrojového kódu
- chybový model vstupních operandů operace, jeho parametry jsou dáno nebo určeno předchozími výpočty
- v OOP pomocí přetížení operátorů, přepínání mezi normálním kódem a kódem propagující chyby (s chybovými modely)
- pro fce $\boldsymbol{z = g(x,y)}$ – $\boldsymbol{g}$ je základní (standardní) aritmetické fce jsou určené kovariace pomocí variace vstupních parametrů. Propagace variace (kovariace) lze najít v tabulce(pro součet a rozdíl se variace sčítají)

(d) Přímé propagování min/max chyby. TVS 5,6

7. Ověřování modelu

(a) Princip a základní charakteristiky metod ověřování modelů.

Metoda QA – formální verifikace, matematicky založené jazyky umožňující specifikace (zápis požadavků) i verifikaci (formální důkaz splnění specifikaci) systémů
manipulace s obrovskými prohledávacími prostory, jasná odpověď „ano“ či „ne“, poskytnutí protikladu
odstraňování nekonečnosti – spojité proměnné, spojitý čas, pravděpodobnosti, zásobníkový automat
Vstup je matematický model (specifikace požadavků jsou formule φ určité temporální logiky)
Verifikace je rozhodnutí, zda model M je modelem formule φ: M = φ

Garance správného chování systémů

- Zvyšování složitosti softwarových systémů
- chyby vedou k velkým ztrátám (finance, lidské životy) – kritická bezpečnost (safety), letadla, vlaky, satelity, lékařská zařízení
- Demonstrace, že požadavky jsou správné, úplné, přesné, konzistentní, testovatelné

Temporální verifikace modelů

- použití temporální logiky (vyjádření času),
- systémy modelovány jako přechodové systémy s konečným počtem stavů.
- reaktivní, distribuované a paralelní systémy
- ověřuje se dosažitelnost, bezpečnost, živost, férovost

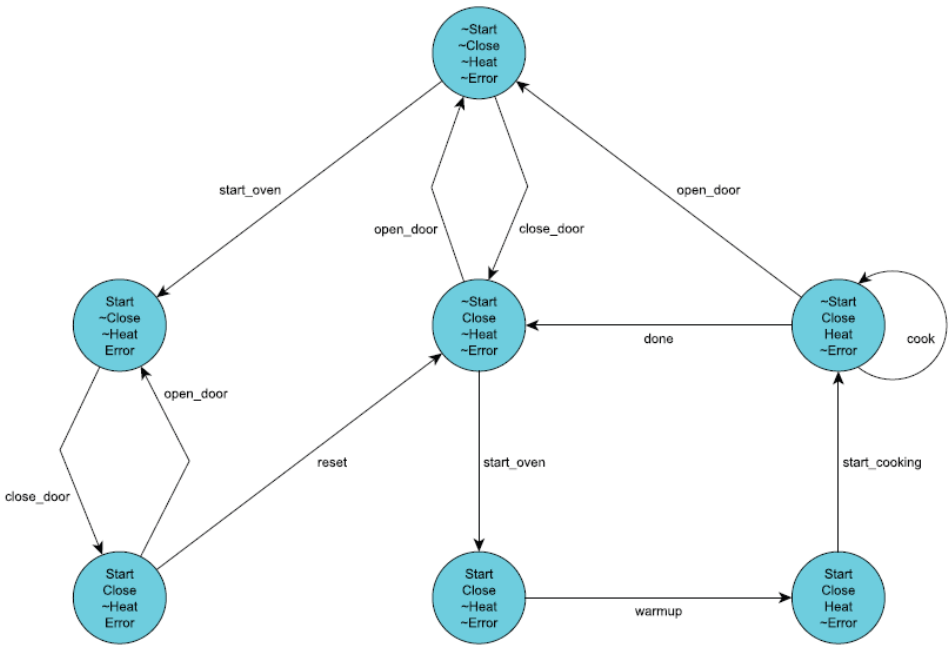
techniky – statická analýza: abstrakce, ověřování modelů (dosažitelnost), dokazování vět: důkaz vlastnosti

Výhody – úplná automatizace, vysoká rychlost, možnost verifikace i částečných specifikací, produkuje protipříklady.

Nevýhody – problém exploze stavů, binární rozhodovací diagramy (BDD), nástroje jsou schopny zvládnout systémy s 100 - 200 stavovými proměnnými, je možné zvládnout systémy s 10-120 stavy.

(b) Kripkeho struktura a její rozšíření.

- je **typ nedeterministického konečného automatu**.
- je to graf, který má dosažitelné stavy systému (vrcholy) a přechody mezi nimi (hrany)
- popisuje **chování** modelovaného **systému**, který je nezávislý na modelovacím jazyku.
- je dána **množinou atomických propozic** AP (platné pro daný stav)
- má **kompletní přechodovou fci**(každý stav má přechod z něho)
- je trojice **M = (S, T, I)**
  - **S** je ***konečná množina stavů***
  - **T** je ***přechodová relace*** (T je podmn. S × S)
  - **I** je ***interpretace AP*** ($S \rightarrow 2^{AP}$)
- **Rozšířená** Kripkeho struktura
  je čtveřice **M = (S, s_0, T, I),** **s_0** je ***počáteční stav***
- nebo pětice **M = (S, s_0, T, I, L), L** je ***značkovací funkce*** (mapuje přechody T do množiny akcí proveditelných programem)



(c) Architektura systému UPPAAL a jeho základní vlastnosti.

- modelování, simulace, verifikace reálných systémů
- **Systém** – síť ***paralelních*** časových automatů (***procesů***)
- **Proces** – ***instance*** parametrizovaného ***vzoru***
- požadavky na systém: výkonnost, použitelnost, diagnostický záznam
- jazyk nedeterministických podmíněných příkazů
- jednoduché datové typy (konstanty, ohraničená celá čísla, pole, iniciátory)
- sítě automatů s hodinami a datovými proměnnými: proces jako automat s množinou pozic, s přechody (orientované hrany)
- stav systému je charakterizován pomocí pozice automatu, hodnot proměnných a stavu hodin (řízení pomocí stráží (podmínka nad proměnnými a hodinami) a synchronizací
- Vlastnosti modelů
    - sada nedeterministických procesů s konečnou řídicí strukturou a reálnými hodinami,
    - komunikují pomocí kanálů nebo sdílených proměnných
- **Komponenty systému**

Systémový editor – Modelování

- tvorba grafického (vzory automatů) i textového (definice, deklarace dat) popisu systému

Verifikátor

- prověření všech možností dynamického chování modelu
- kontrola invariantů a živosti prohledáváním stavového prostoru
- dosažitelnost symbolických stavů reprezentovaných omezeními
- Editor specifikace požadavků

Simulátor

- grafická vizualizace a záznam možného chování systému (vyšetřování možných dynamických běhů systému)
- sekvence symbolických stavů
- detekce vad modelu před jeho verifikací,
- možnost vizualizace trasy generované verifikátorem (umožňuje analýzu záznamu běhů vedoucích k nežádaným stavům), vyšetřování běhu systému

(d) Časový automat a jeho sémantika

- je **konečný** stavový automat **s hodinami** (spojitý čas)
- je šestice ***(L,l_0, C,A,E,I)***
    - ***L*** je množina pozic
    - ***l_0*** je počáteční pozice
    - ***C*** je množina hodin
    - ***A*** je množina akcí, ko-akcí a interní τ-akce
    - ***E*** je množina hran mezi pozicemi s akcí, stráží a množinou hodin, které se resetují
        - E je podmnožinou $L \times A \times B(C) \times 2^C \times L$
    - ***I*** : $L \rightarrow B(C)$ přiřazuje invarianty k pozicím
- Sémantika:
    - Ohodnocení hodin je fce přiřazení prvkům C nezáporným hodnotám reálných čísel.
    - Přechod ($\rightarrow$ je podmnožinou $S \times (R_{\geq 0} \cup A) \times S$) je možné provést pomocí ***akce*** nebo ***zpoždění,*** při akce se některé hodiny nulují
    - množina stavů S je podmnožinou $L \times R^C$
    - $s_0 = (\ell_0, u_0)$ je počáteční stav (počáteční pozice a hodiny jsou nulové)
- síť automatů mají společnou množinu hodin a akcí, množina stavů je kartézským součinem množiny pozic ***L*** jednotlivých automatů

(e) Vysvětlete základní entity modelování v UPPAAL: synchronizace a její typy, pozice a jejich speciální vlastnosti, stráž, invariant.

- **Synchronizace** - kanál, synchronizuje se přes návěstí Expression**!** a Expression**?**
- **Binární** synchronizace - ***chan*** c, synchronizace hrany c! a c? Nedeterministicky
- **Broadcast** synchronizace - ***broadcast*** chan c, jedna hrana c! se všemi možnými c?, neblokuje.
- **Urgentní** synchronizace – urgent ***chan*** c, zpoždění není dovoleno, pokud je možný přechod na urgentním kanálu
- **Pozice** obsahuje jméno a invarianty
- **Počáteční** pozice **(*initial*)** – Nová instance modelu se nachází v tomto stavu.
- **Urgentní** pozice **(*urgent*)** – Čas systému nemůže plynout, pokud se systém nachází v urgentní pozici.
- **Prováděcí** pozice **(*commit*)** – nemůže se zpožďovat – následující přechod musí zahrnovat jednu výstupní hranu vedoucí z prováděcí pozice. (Musí se pokračovat ven z commit pozice a nelze dělat nic jiného)
- **Stráž** je logický výraz (vyhodnocení ***true/false***), je přiřazena k přechodům (hranám).
- **Invariant** je podmínka (konjunkce podmínek), která musí být splněna kdykoli se automat nachází v daném stavu (při vstupu, při setrvání a při výstupu)

(f) Vysvětlete pojem dosažitelnosti, bezpečnosti a živosti a jak se tyto vlastnosti ověřují v systému UPPAAL.

Stavové formule

- lze vyhodnotit pro daný stav, aniž by bylo nutné analyzovat chování modelu
- nadmnožinou stráží, tj. Nemá žádný postranní efekt
- použití disjunkcí není omezeno na rozdíl od stráží

Dosažitelnost

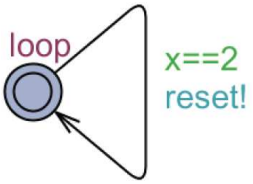
- existuje možnost, že daná stavová formule φ je splněna v každém dosažitelném stavu.
- Příklad – Je vůbec možné odeslat zprávu? Zpráva má nadějí být přijata?
- **E[] φ**

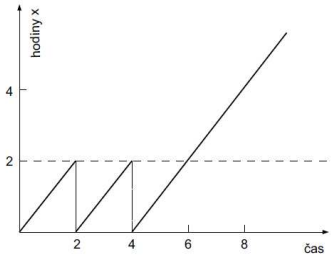
Bezpečnost

- Vlastnost, která nesmí nikdy nastat, např. nelze překročit maximální teplotu nebo konec souboru a podobně (často kontrola mezí).
- Něco není možné, aby vůbec nastalo
- Je formulována pozitivně – **A[] φ= ¬E<>¬φ ,** formule je pravdivá ve všech dosažitelných stavech

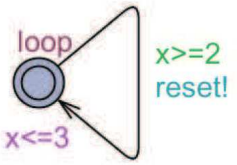
Živost

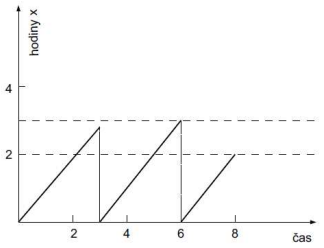
- něco jednoho dne určitě nastane
- Automat se jednou dostane do určitého stavu.
- př. stisknutí tlačítka ***on*** způsobí, že televize jednou zapne
- **A<> φ = ¬E[]¬φ**
- **φ → ψ ≡ A□(φ =>A◊ψ)**

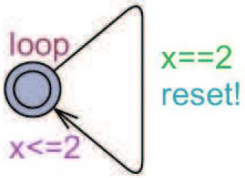


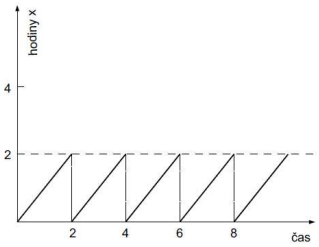


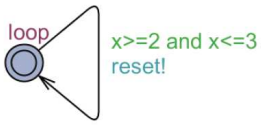
(g) Používání invariantů a stráží nad hodinami v systému UPPAAL.

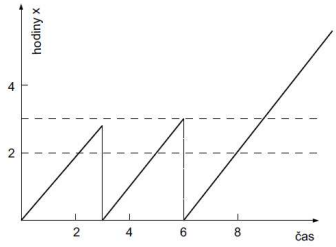












Invariant

- je podmínka, která musí být splněna kdykoli se automat nachází v daném stavu
- narozdíl od stráže neslouží jako kontrola podmínky, ale jde o součást stavu (automat se do daného stavu vůbec nemůže dostat, pokud není splněn invariant).

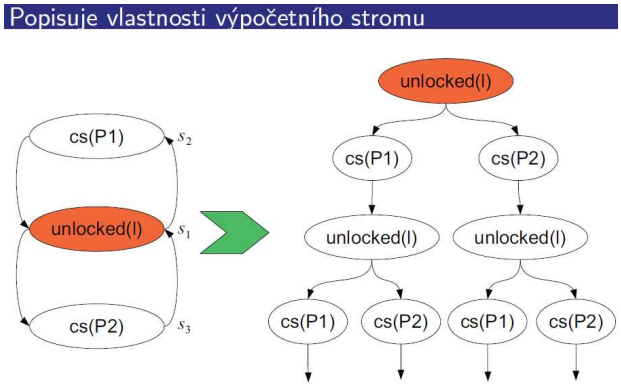
Stráž je výraz, kterou lze vyhodnotit true/false. Stráže jsou přiřazeny k přechodům (hranám).

- ***side-effect free*** (nelze přepisovat proměnné apod.)
- hodiny, rozdíly hodin, celočíselné proměnné a konstanty

8. Temporální logiky

(a) Cesta výpočtu a pojem času.

- Cesta π – v **Kripkeho struktuře M** je **nekonečná sekvence** stavů
- $\boldsymbol{\pi = s_i s_{i+1} s_{i+2} \ldots}$ taková, že pro všechny ***i*** patřící do ***N*** a $\boldsymbol{R(s_i , s_{i+1})}$
- **Π(M,s) –** množina všech cest v **M,** které začínají v ***s*** patřící do ***S***
- abstrakce
  - ***logický*** čas – pracuje s (částečným) uspořádáním stavů v chování systému
  - ***fyzický*** čas – měření doby uběhlou mezi dvěma stavy
- čas ve verifikaci modelů
  - ***lineární*** čas – dovoluje se vyjadřovat pouze o dané ***lineární trase*** ve stavovém prostoru (na všech trasách, následování)
  - ***větvící se*** čas – dovoluje kvantifikovat (existenčně i univerzálně) možné budoucnosti počínaje daným stavem (stavový prostor je rozvinutý nekonečný strom)



(b) CTL* logika a její temporální operátory.

- Skládá se z **atomické výroky**, **logické spojky**, kvantifikátory cest a temporální operátory
- **kvantifikátory** popisují strukturu větvení výpočetního stromu
  - E – existuje cesta výpočtu z daného stavu
  - A – pro všechny cesty výpočtu z daného stavu
- **temporální** operátory
  - X **φ** (ne**X**t time, ○) – platí v následujícím stavu (po počátečním) cesty
  - F **φ** (in **F**uture, ◊) – platí v nějakém stavu cesty
  - G **φ** (**G**lobally, □) – platí ve všech stavech cesty
  - **φ** U **ψ** (**U**ntil) – **ψ** platí v ***nějakém*** stavu cesty a **φ** platí ve všech ***předcházejících*** stavech cesty
  - **φ** R **ψ** (**R**elease) – **ψ** platí do ***nějakého*** stavu (včetně) cesty, začne platit **φ**
- **stavové** formule (popisují individuální stavy) jsou **výroky** (atomické výroky, ¬ φ, φ ^ ψ , φ v ψ**)** a **Eφ , A φ**

- **M,s |= φ** znamená, že platí v stavu **s**

- **běhové** formule (vyhodnocují se podél cest a stop modelu) jsou stavové formule, ¬ φ, φ ^ ψ , φ v ψ, X**φ** , F **φ**, G **φ** , **φ** U **ψ**, **φ** R **ψ**
  - **M, π |= φ** znamená, že platí podél cesty **π** v M

(c) CTL logika a její temporální operátory.

Sublogikou CTL*

- viz. (b)
- **běhové** formule jsou **OMEZENY** na X**φ** , F **φ**, G **φ** , **φ** U **ψ**, **φ** R **ψ**
- **modální** operátory AX, EX, AF, EF, AG, EG, AU, EU, AR, ER (základní EX, EG, EU)

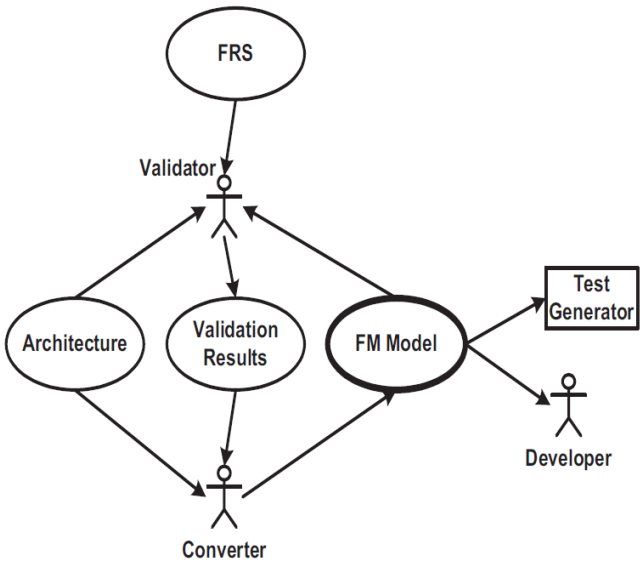
(d) LTL logika a její temporální operátory.

Sublogikou CTL*

- viz. (b)
- povoluje pouze formule tvaru **Aφ**
- φ = Aψ
  ψ ::= *p* | ¬*ψ* | *ψ* v *ψ* | *ψ* ^ *ψ* | **X***ψ* | **F***ψ* | **G***ψ* | *ψ***U***ψ* | *ψ***R***ψ*, kde *p* patří do AP
- ignoruje větvení (tím se myslí E?)

(e) Temporální logika systému UPPAAL.

- BNF pro vyjádření logiky
- Žádný výraz nesmí mít postranní efekty
- **Stavové formule φ**
  - lze vyhodnotit pro daný stav, aniž by bylo nutné analyzovat chování modelu
  - nadmnožinou stráží, tj. Nemá žádný postranní efekt
  - použití disjunkcí není omezeno na rozdíl od stráží
  - výraz ***process.location*** (instance automatu, specifický stav)testuje, zda určitý proces je v dané pozici
  - deadlock, splněná pro všechny zablokované stavy (neexistuje žádný akční přechod z daného stavu), příklad: *A[] not deadlock*.
- **E<> φ**: ***existuje*** cesta, kde φ ***bude*** alespoň ***jednou*** splněna
- **A[] φ**: ***pro všechny*** cesty podmínka φ je ***vždy*** splněna.
- **E[] φ**: ***existuje*** cesta, kde podmínka φ je ***vždy*** splněna.
- **A<> φ**: ***pro všechny*** cesty φ ***bude*** alespoň ***jednou*** splněna
- **φ → ψ**: ***pokud*** podmínka **φ** je splněna, ***pak bude*** podmínka **ψ** ***jednou*** splněna.



9. Formální metody

- specifikace SW, snaha o přesnější, stručné a strojově interpretované zápisy (oproti volným větám)
- využití v nástrojích pro testování (viz obr.)
- aplikace logiky a jednoduchá matematika, formální zápis SW
- modelování (jako orákulus), návrh, verifikace

(a) Z notace – principy a základní vlastnosti.

- je standardizovaná notace (ANSI) – právě jenom notace: není exekuční ani programovací jazyk
- je založena na teorii množin a matematické logice
- **predikátový kalkulus prvého řádu**

- Systém je modelován pomocí:
  - **stavu**, stavové proměnné a jejich hodnot
  - **operaci**, které mohou měnit stav
- **Schéma** – vzory deklarací a omezení – pojmenovaný důležitý koncept
- **Typy** – každý objekt má jednoznačný typ (maximální množina v rámci dané specifikace)
- deklarace typu „**[]**”, objektu „**:**“, **zkratek** (množiny) „==“
- axiomatické popisy (**deklarace** + omezující **predikát**) = schéma
- vstup „**?**“, výstup „**!**“, dekorace (stav před a po operacích „**’**“, **ΔState**, **ΞState**),
- binární relace v deklaracích (např. parciální fce⇸)
- celá operace je spojení schémat T_CheckOut = CheckOut ∨ CheckedOut ∨ Unathorized
- projekční operátory (first, second…)

(b) PVS systém – principy a základní vlastnosti.

- specifikační jazyk
- nástroj podporuje dokazování vět
- predikátová logika vyššího řádu
- funkcionální specifikační styl – stav systému je předáván jako argument funkcí

kroky dokazování:

- vytvoření specifikace
- syntaktická analýza
- typová kontrola
- dokazovaní – automatické a poloautomatické dokazování

syntaxe**:**

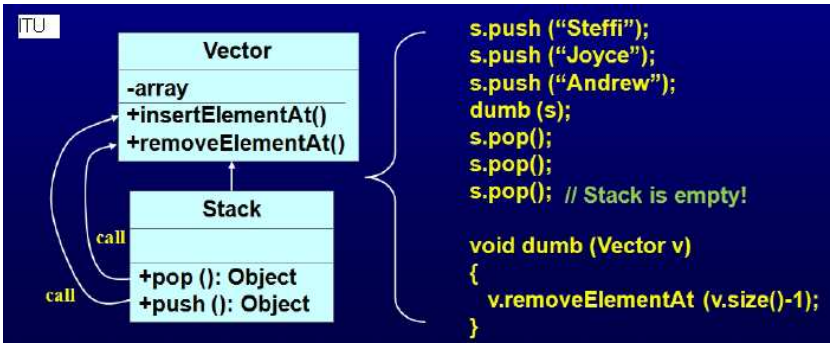
- **entita, funkce, typy entit** (N: TYPE)
- funkce B: TYPE = [N → P]
- páry, n-tice (…, …)
- dokazování pomocí inferenčních pravidel (nahoře předpoklady…, dole úsudek – [jméno pravidla])

```
cvs: THEORY
BEGIN
    Person: TYPE+
    Document: TYPE
    CheckedOut: TYPE = [Document -> Person]
    Permissions: TYPE = [Document -> setof[Person]]
    nobody: Person

    RealPerson: TYPE = {p: Person | p /= nobody}

    co: VAR CheckedOut
    p: VAR RealPerson
    d: VAR Document
    db: VAR Permissions

    Locked?(co, d): bool = co(d) /= nobody
    Permitted?(db,d,p): bool = member(p,db(d))

    CheckOut(db, co, p, d): CheckedOut =
          IF Locked?(co,d) OR NOT Permitted?(db, d, p) THEN co ELSE co WITH [(d) := p] ENDIF
    FindPerson(co, d): Person = co(d)

    FindAddPT: CONJECTURE
        NOT Locked?(co, d) AND Permitted?(db, d, p) => FindPerson(CheckOut(db,co,p,d),d)=p
END cvs
```

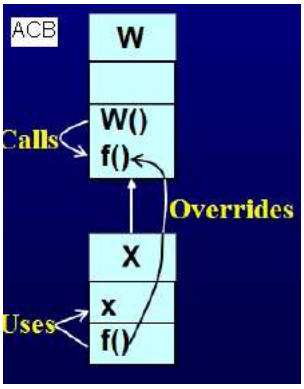
10. testování OO softwaru

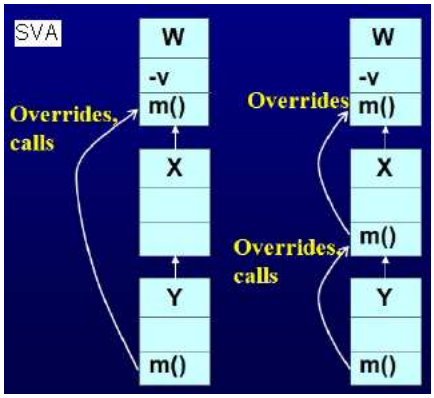
(a) Anomálie DU párů

- Problémy s dědičnosti, přístupnosti metod tříd
- **SDA** – anomálie definice stavu – přepisující metoda má jinou def-množinu než přepisovaná metoda – definice v metodě h(), užití v j(), potomek ale přepíše h() a ***nedefinuje***, užití v metodě j() nepřepíše
- **ITU** – nekonzistentní užití

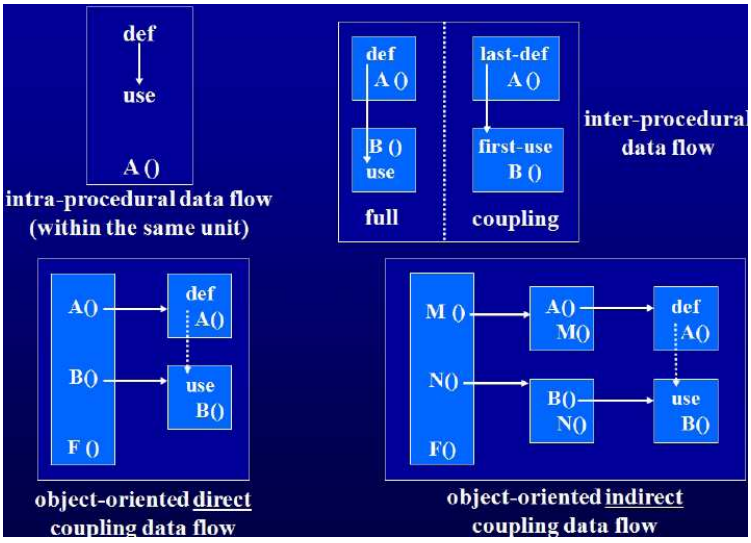


- **SDIH** – nekonzistence stavu – podobně jako SDA, potomek ale předefinuje něco v prarodiči, jeho předchůdce má metodu, která užívá chybně definované proměnné
- **ACB1** – konstruktor volá metodu, ten je přepsán v potomci, ta metoda používá něco, co ještě není definováno (konstruktor nedokončen)



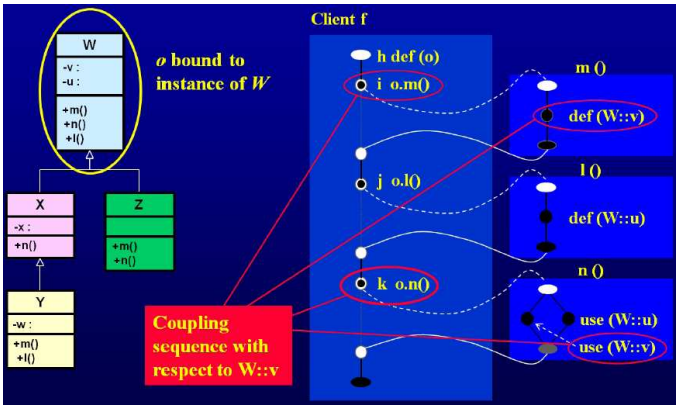


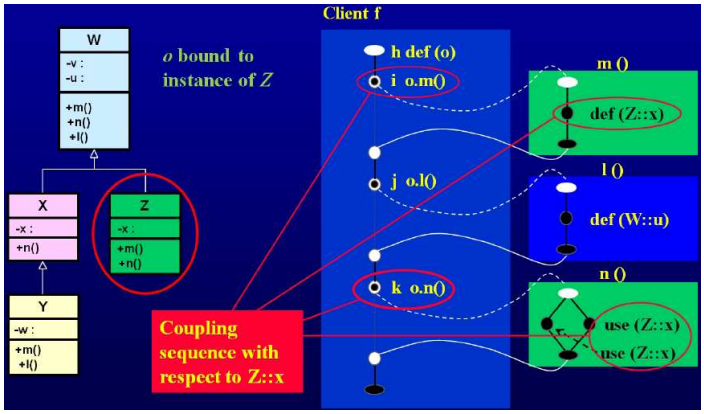
- **SVA** – přepsání metody v prarodiči, v rodiči je přidán přepis metody později a způsobí, že potomek zavolá místo metody v prarodiči metodu v rodiči.



(b) Testování párových sekvencí

- Typy testování – intra, inter – metod, intra inter tříd
- sekvence reprezentují interakce stavových prostorů, indentifikace bodů integrace a testovacích požadavků
- last-def, first-use
- množina polymorfních volání je množina metod, které mohou být potenciálně provedeny jako výsledek volání metody v rámci dané instance (metody od všech tříd v hierarchiích dědění)
- příklad vazební sekvence na obr.





11. Spolehlivost softwaru

- **Spolehlivost = pravděpodobnost**, že systém bude vykonávat zamýšlenou fci v daných operačních podmínkách po specifickou dobu
- **Modely** spolehlivosti se používají **k odhadu** – objektivní vyjádření, plánování zdrojů (sleduje se počet defektů v poměru s počtem řádků) za daný časový interval

(a) Základní metriky kvality softwaru

Metriky **produktu** – popisuji charakteristiky jako je velikost, komplexity, návrhové vlastnosti, výkonnost, úroveň kvality

Procesní metriky – mohou byt využity při zlepšovaní vývoje softwaru a procesu údržby

- efektivita odstraňovaní defektu během vývoje
- vzor přírůstku defektu během testovaní
- doba odezvy procesu oprav

Metriky **projektu** – popisuji charakteristiky projektu jeho provedení

- počet vývojářů SW
- vývoj personálu během životního cyklu softwaru
- cena, rozvrh, produktivita

Metriky **kvality SW** – podmnožiny SW metrik

- ***finálního produkt*** – MTTF, MTBF, intenzita defektů, velikost SW, hlášené problémy zákazníkem, spokojenost zákazníka
- ***průběhu procesu*** – *zaznamenání přírůstků defektů*: hustota (defekt na KLOC), vzor přírůstků, profil odstraňování, efektivnost odstraňování
- ***údržby* –** počet ohlášených problémů nevyřízených (k datumu), BMI (backlog management index) = procentuální poměr mezi uzavřené problémy a přírůstkem problémů, metrika vadných oprav za časový interval

(b) Statické modely spolehlivosti vývoje softwaru TVS11

- používá atributy projektu nebo programových modulů k odhadu počtu defektu v SW
- parametry modelu jsou odhadovány na základě řady předchozích projektu
- Weibullova distribuce, Rayleighův model
- rychlost defektu pozorovaných během vývojového procesu je positivně korelovaná s rychlosti defektu v poli nasazení
- čím více defektů je objeveno a odstraněno dříve, tím méně jich zůstane na pozdější fáze
- predikce je platná na základě přesnosti a spolehlivosti vstupních dat
- stupeň změny výstupů modelu vzhledem k možnosti fluktuací vstupů

(c) Dynamické modely spolehlivosti vývoje softwaru TVS11

- používá průběžného vývoje vzoru defektu k odhadu spolehlivosti finálního produktu
- parametry dynamických modelů jsou odhadovány na základě mnoha údajů zaznamenaných o hodnoceném produktu k danému datu
- *modely růstu spolehlivosti* se obvykle odvozuji z dat fáze formálního testovaní
    - Během tohoto testovaní po fázi vývoje, kdy se objevují selhaní, identifikují a opravují defekty, se SW produkt stává stabilnější a jeho spolehlivost roste s časem.
- Exponenciální model (speciální případ Weibullova distribuce)
- Model doby mezi selhání, počtu vad …
- J-M, LW, G-O,M-O, zpožděný S, inflexní S