ZKS

# Otázky

Generování testu z modelu aplikace - kombinace vstupních dat, urovne testovacího pokrytí, pairwise testing - Popsal jsem model jako graf - vrcholy stavy aplikace/rozhodovací body, hrany jako akce sloužící pro přechod mezi stavy. Pak jsem definoval MCC, MC/DC a Pairwise (CC a DC jsem mu nakonec rekl taky, ale nechtel to), popsal Pairwise a ukazal na prikladech. Celkove Bureš dost lpěl na formalnejsich definicích

# Obecné

**7 principů testování**

* *Testování ukáže přítomnost defektů* - Nelze dokázat nepřítomnost defektů
* *Vyčerpávající testování* je nemožné - Analýza rizik, stanovení priorit
* *Včasné testování*: Začít testovat asap, konkrétní zaměření, Boehmův první zákon
* *Shlukování defektů*: Malé množství modulu obsahuje většinu defektů
* *Pesticidní paradox:* Je třeba revidovat testy a psát nové, stejné testy časem nenaleznou nové defekty
* Testování je závislé na kontextu: Testovat z různých pohledů, v různých kontextech
* Falešná představa o neexistenci omylů

**Blackbox testing**

* Testovací scénáře vytvořeny na základě specifikace/dokumentace, není známa vnitřní struktura
* Např. funkční scénáře na základě use case

**Whitebox testing**

* Testovací scénáře na základě vnitřní struktury
* Např. statická analýza kódu

**Greybox testing**

* Kombinace whitebox a blackbox

**Verifikace**

* Proces vyhodnocování, zda produkt konkrétní vývojové fáze odpovídá požadavkům definovaným na začátku fáze
* Vyvíjíme produkt správně (podle specifikace)?

**Validace**

* Vyhodnocování SW v průběhu vývoje nebo na konci, jestli odpovídá zadaným požadavkům
* Vyvíjíme správný produkt (co klient požaduje)?

**Test basis**

* Dokumentace (nebo všechny informace), na jejímž základě jsou vytvářeny testovací scénáře

**Test level**

* Skupina testovacích aktivit, které jsou organizované a řízené jako jeden blok, mají samostatný cíl

**UAT**

* Uživatelské akceptační testy

**Boehmův první zákon**

* Chyby jsou nejčastější ve fázích požadavků a návrhu, čím později je odhalíme, tím bude jejich oprava dražší
* Nevýhoda V-modelu - drahé chyby způsobené špatnými požadavky jsou odhaleny až ke konci testování (waterfall)
* ->W-model, každé fáze přípravy systému odpovídá kontrolní fáze

**Regrese**

* Zanášení chyb do systému vlivem oprav nebo změn

**Test coverage**

* Jaká část SW a jeho funkcí je pokryta testovacími scénáři, vyjádřeno v %

# Statické testování

* Testování komponenty nebo systému na úrovni specifikace/implementace bez spouštění systému
* Efektivní metoda, ale vyžaduje počáteční investici
* Validace jednotlivých částí návrhu proti předchozím artefaktům (levá část v-modelu, nebo revize v w-modelu) - návrh vůči požadavkům, technický návrh vůči návrhu, implementaci vůči technickému návrhu atp.
* Validace konzistence artefaktu uvnitř
* Možnosti:
  + Pro specifikaci - různé typy revizí, quality gates, kontroly konzistence
  + Pro implementaci - statická analýza (zahrnuje code review), quality gates

## Revize

(vzestupně podle stupně formálnosti)

**Informal review**

* Není přímo řízené /moderování, code review/revizi analytických výstupů

**Walkthrough**

* Autor prochází artefakt se skupinou spolupracovníků, kteří jej připomínkují

**Technical review**

* Formální typ revize řízený moderátorem
* Dokumentovaný, předem definovaný proces, Revidující se připravují předem
* Cílem může být kromě nalezení defektů také rozhodnutí konkrétní technické otázky

**Inspection**

* Přesně definovaný proces, použití metrik, příprava předem, kontrolní seznam
* Povinný seznam zjištění, definovaný proces na jejich zpracování

## Quality gates

* Revize je součástí vývojového cyklu jako povinná součást přechodu mezi fázemi
* Může být zkombinována se statickým testováním

**Formální**

* Proces nemůže pokračovat, dokud artefakt nemá požadovanou kvalitu
* **+** jasná pravidla, transparentní proces
* **-** obcházení/hackování quality gate, možné zdržení procesu, degradace procesu pod tlakem, může sklouznout do kontroly formálních náležitostí - nedostatek pozornosti pro obsah

**Kolaborativní**

* Meeting producentů artefaktu s odběrateli, odběratelé se ptají, producenti vysvětlují
* Obě strany se informují předem (o artefaktu/otázkách odběratelů)
* Zápis - oprava dokumentace/nové issues
* **+** efektivita, proces se nemusí zastavit, ale je třeba vyřešit issues
* **-** jak nakombinovat účastníky a vytvořit agendu, aby to bylo relevantní pro všechny, nutná vhodná komunikace otázek

## Implicitní revize

* V průběhu přípravy testů je kontrolována konzistence specifikace
* V průběhu návrhu testů:
  + *Test životního cyklu dat* - konzistence CRUD matice
  + *Procesní test* - test analytik vytvoří nezávisle activity diagram a kontroluje nekonzistence, srovnává s verzí v návrhu a konzultuje rozdíly
  + *Test přechodu stavů* - analogie s předchozím
* V průběhu vytváření testovací strategie:
  + *Metoda BDTM* - srovnání vzájemné konzistence a pokrytí test goals, požadavků na systém, nadefinovaných procesů a modulů/funkcionality

## Statická analýza kódu

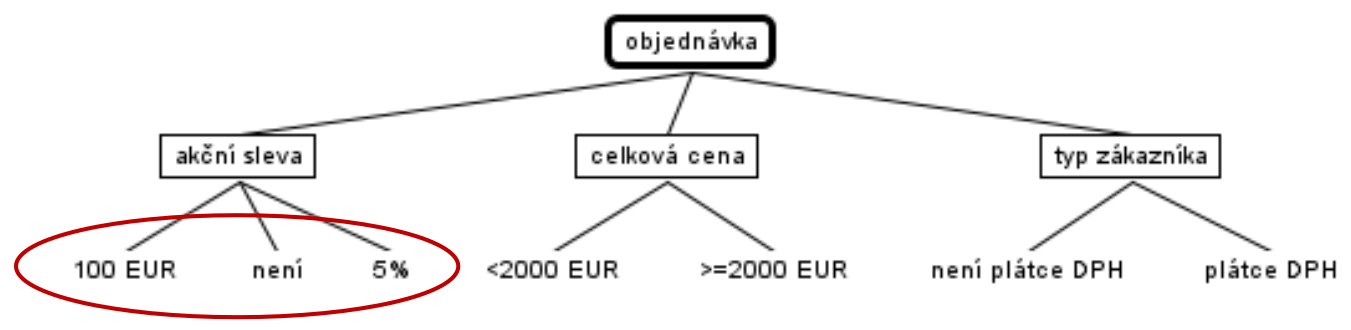
* Analýza bez spouštění kódu
* Detekce potenciálních chyb, formální zjištění vlastností kódu

# Automatické testování (TODO)

* Kdy to má cenu?
  + Spouštění na různých platformách, N opakování, málo času na regresní/smoke testování, intenzivní testování
* Vytvoření něco stojí, je třeba zvážit, jestli se tvorba automatických testů vyplatí
* Testy musí být automatizovatelné, nutná údržba
* **Test automation pyramid**
  + Unit > Service > UI

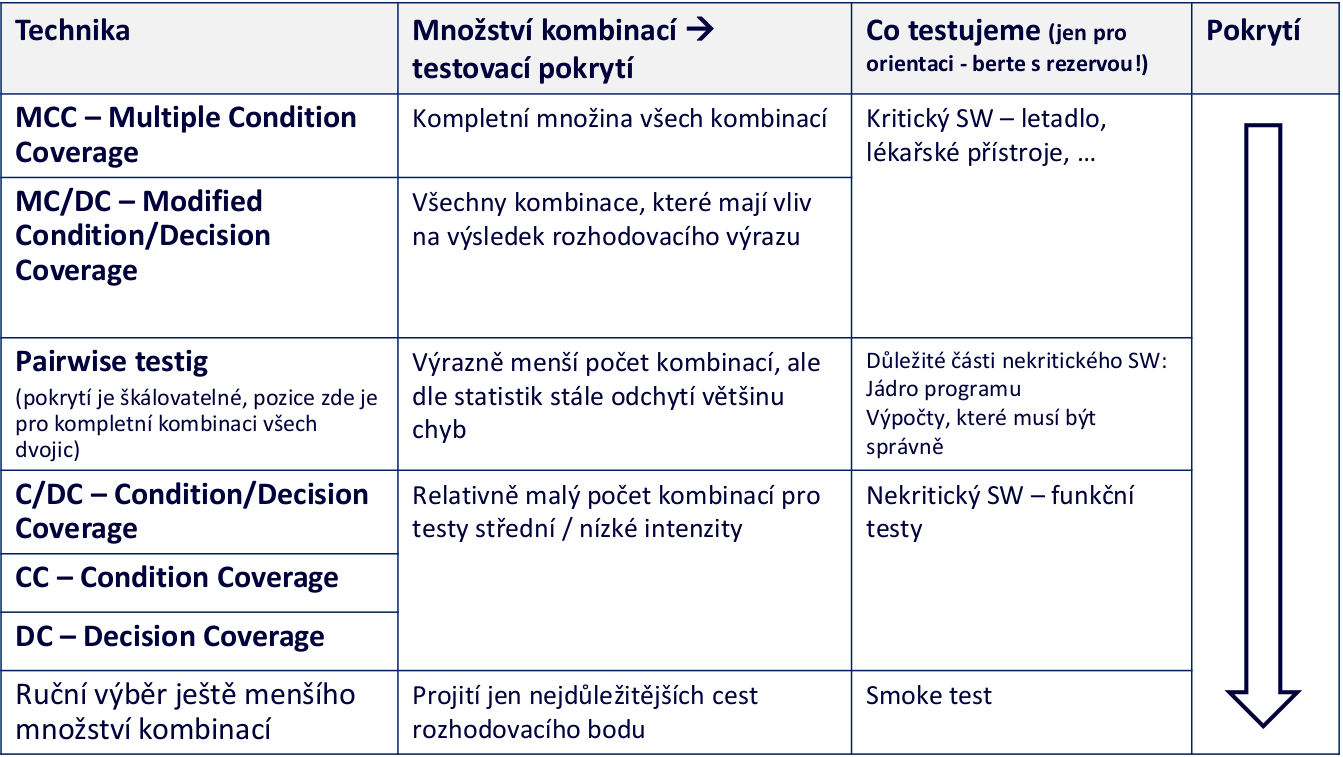
# Příprava kombinací testovacích dat

* **Smoke test**
  + Sada testů prověřují nejdůležitější funkcionalitu, cílem je ověřit, jestli nejdůležitější části aplikace fungují
* **Test condition**
  + Obecný element nebo událost, kterou je možné ověřit testovacím(i) scénáři
  + Např. Funkce, transakce, strukturální element
* Akce a přechody v aplikace lze vnímat jako UML activity diagram/digraf
  + Test condition = rozhodovací bod
* **Klasifikační strom**
  + Lze vytvořit pro každý rozhodovací bod v aplikaci v různých úrovních
  + Uzly = jednotlivé vstupy pro rozhodovací bod (form field, část podmínky)
  + Listy = třídy ekvivalence



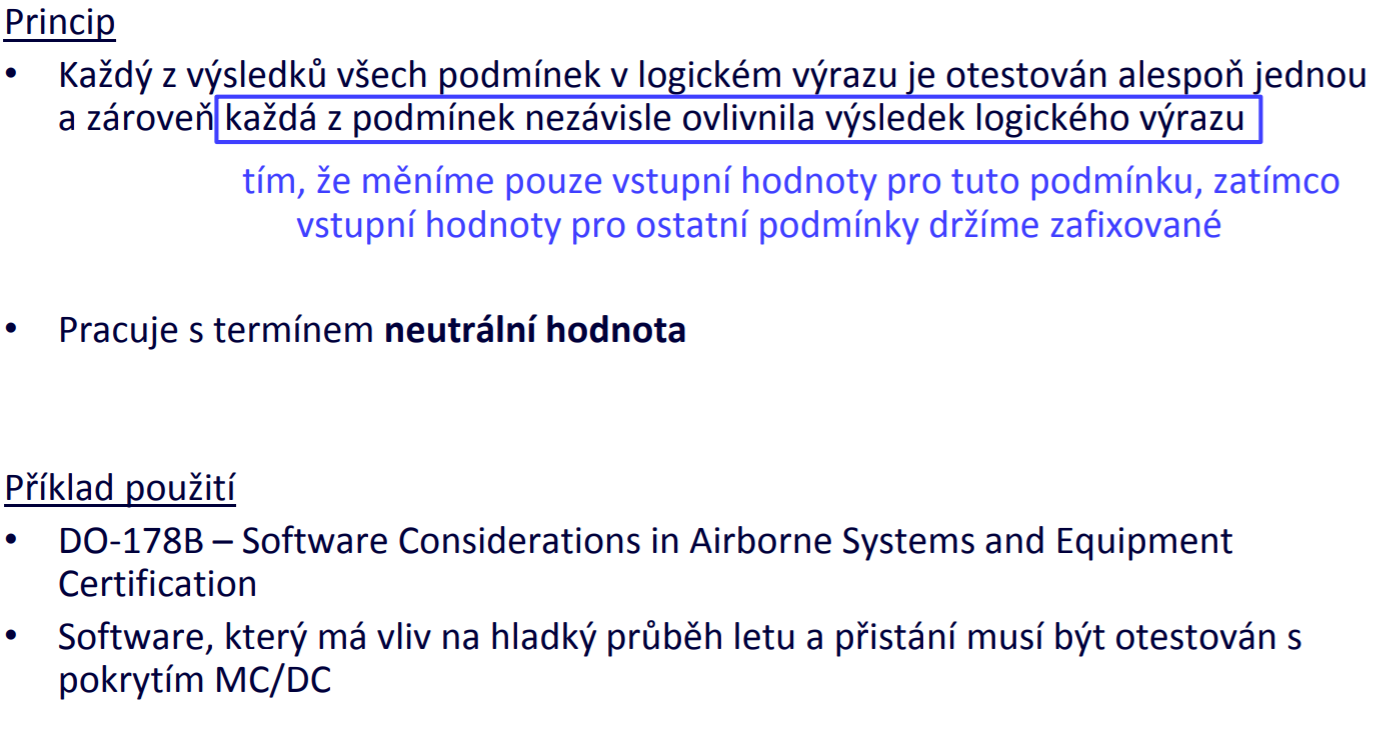
* **Třídy ekvivalence (EC, equivalence class)**
  + Vstupy mohou mít velké množství různých hodnot
  + Systém se má chovat pro všechny hodnot z konkrétní třídy ekvivalence stejným způsobem
  + Typy - podle vstupu:
    - *Interval* - hranice intervalu = mezní podmínky
      * Např. částka, věk, číslo účtu
    - *Diskrétní hodnoty* - nejsou mezní podmínky
      * Např. metoda platby, položka v menu, select box
  + Podle validity dat:
    - Validní EC - situace, které systém zpracuje standardním způsobem
    - Nevalidní EC - systém by měl zareagovat korektním ošetřením chyby
      * Nevalidní z technického pohledu - např. datový typ, nevalidní formát čísla účtu apod.
      * Nevalidní z business pohledu - např. číslo účtu, které neexistuje, číslo mimo definovaný rozsah
  + Pravidla konzistence:
    - Třída ekvivalence **nesmí být prázdná**
    - Dvě třídy ekvivalence musí mít prázdný průnik (trošku lidsky: nesmí se překrývat)
    - Sjednocení všech tříd ekvivalence musí být původní interval/množina
  + Určení tříd ekvivalence
    - Diskrétní hodnoty - analýza požadavků a specifikace
    - Interval - analýza požadavků a specifikace pomocí techniky mezních podmínek
  + **Mezní podmínka** = hodnota, která tvoří hranici mezi EC, Testování mezní podmínky M, M + I a M - I. Přesnost inkrementu dána datovým typem, nebo specifikací.

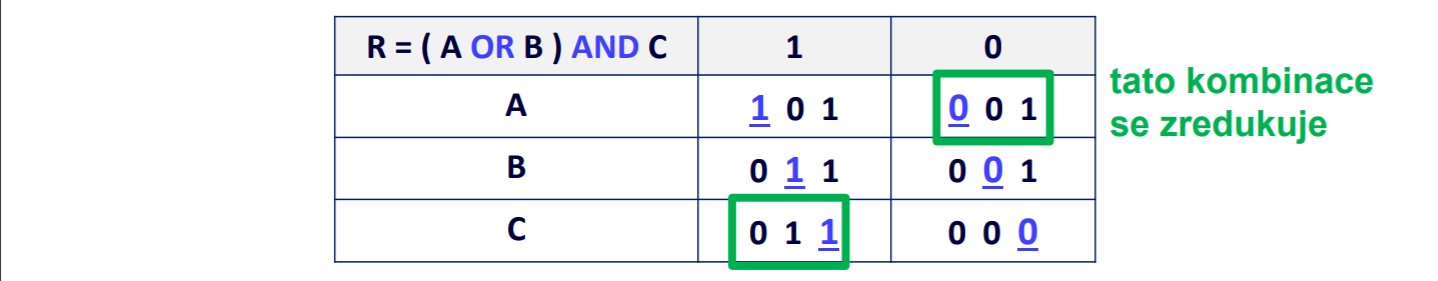
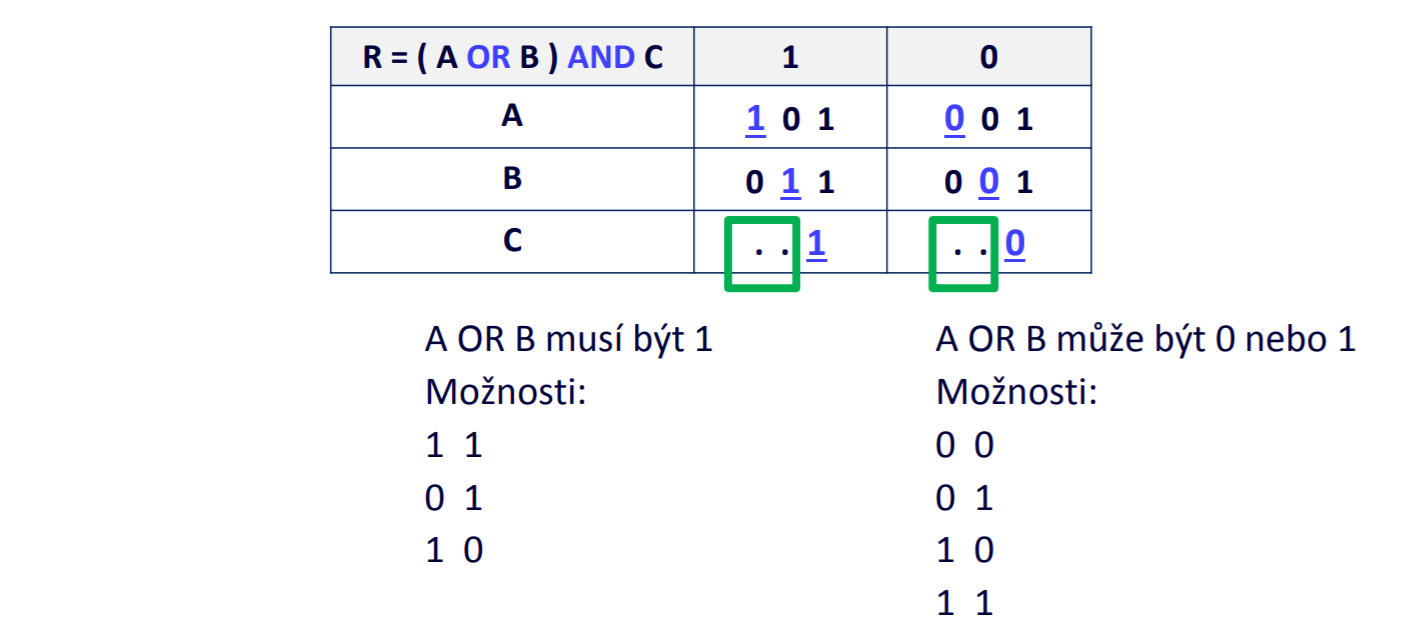
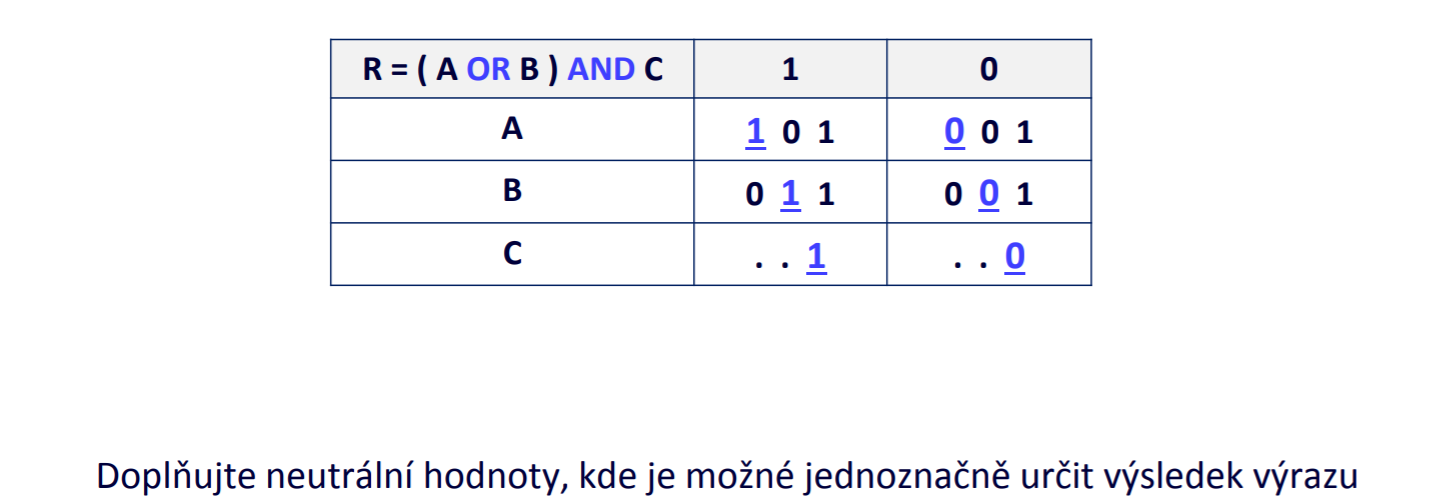
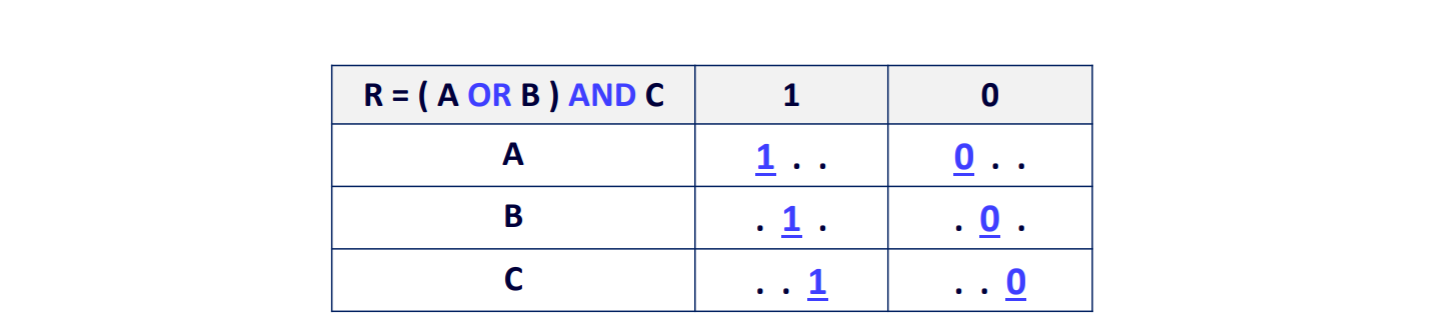
## Kombinace vstupních hodnot

****

* **DC** - každý z výsledků celého logického výrazu je otestován alespoň jednou
* **CC** - každý z výsledků všech podmínek ( **a < b** and **c < d** ) výrazu je otestován alespoň jednou
* **C/DC** - každý z výsledků všech podmínek a výsledků celého logického výrazu je otestován alespoň jednou (Kombinace předchozích dvou)
* **MCC** - všechny možné kombinace hodnot podmínek, pro logický výraz 2počet podmínek, pro klasifikační strom produkt počtu EC ve vstupech
* **MC/DC** - každý z výsledků všech podmínek v logickém výrazu je otestován alespoň jednou a zároveň každá z podmínek nezávisle ovlivnila výsledek logického výrazu (tj. zafixuju hodnoty ostatních parametrů)
  + Neutrální hodnota podle logického výrazu (1 pro AND, 0 pro OR)
* **Pairwise testing (n-wise)**
  + Princip: pokud je v SW defekt, nejčastěji se aktivuje hodnotou jednoho konkrétního vstupu nebo kombinací dvou vstupů
  + Vyberu konkrétní dvojice vstupů a pokryju všechny kombinace EC
  + Počet kombinací podle konfigurace párů a počtu tříd ekvivalence
    - (počet EC v prvním vstupu) \* (počet EC v druhém vstupu) \* ...
  + Škálovatelnost - dvojice pro všechny kombinace vstupů, jen pro nejdůležitější, použití trojic, kombinace dvojic a trojic
  + Nástroje pro vytváření kombinací - Allpairs, CTE XL
* **Výběr několika kombinací**
  + Nízké testovací pokrytí
  + Je třeba hlubší analýza mezních podmínek + důležité průchody z business analýzy
  + Hlavní pozitivní průchod, nejčastější průchod, průchod s největším objemem obchodní aktivity

### MC/DC

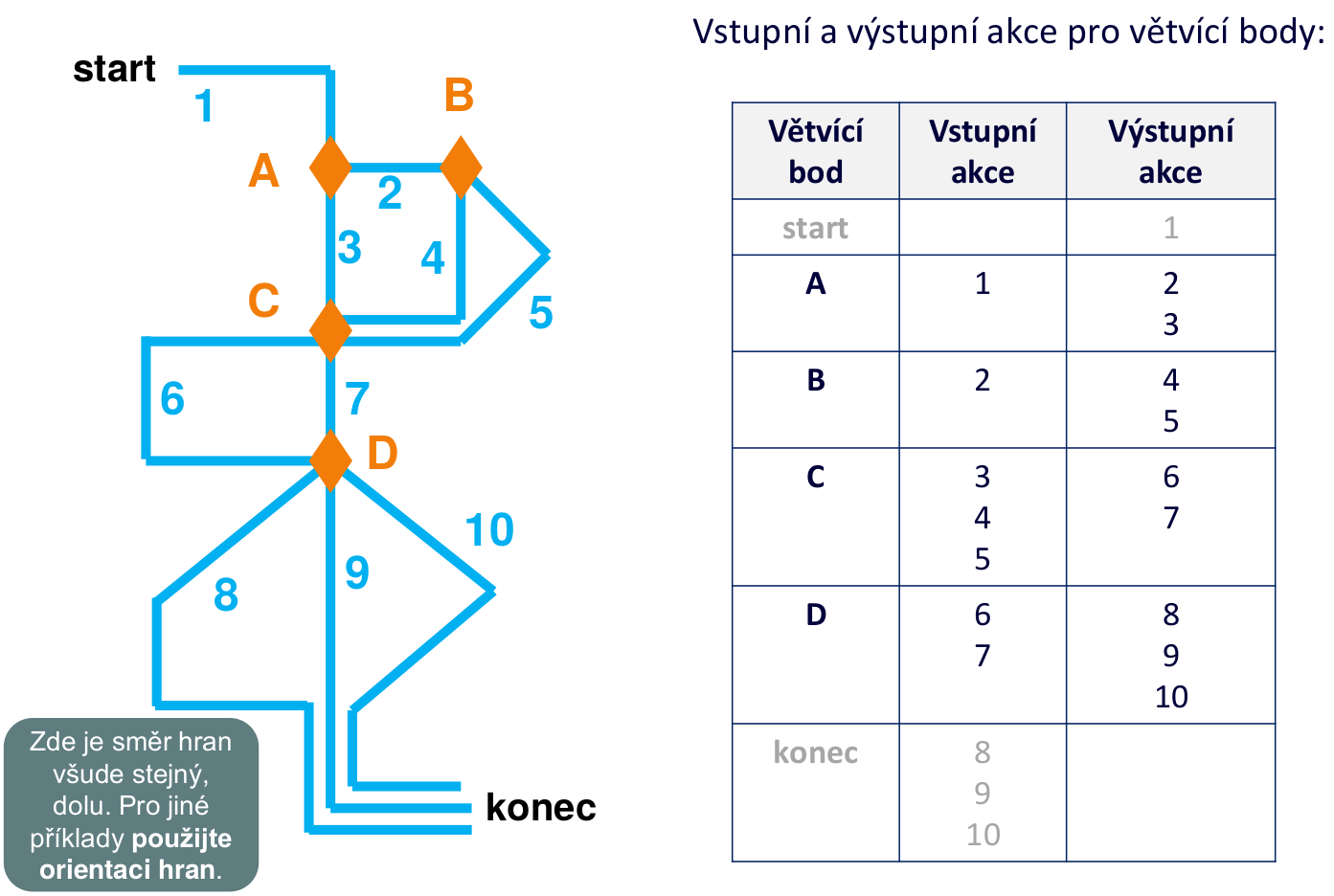


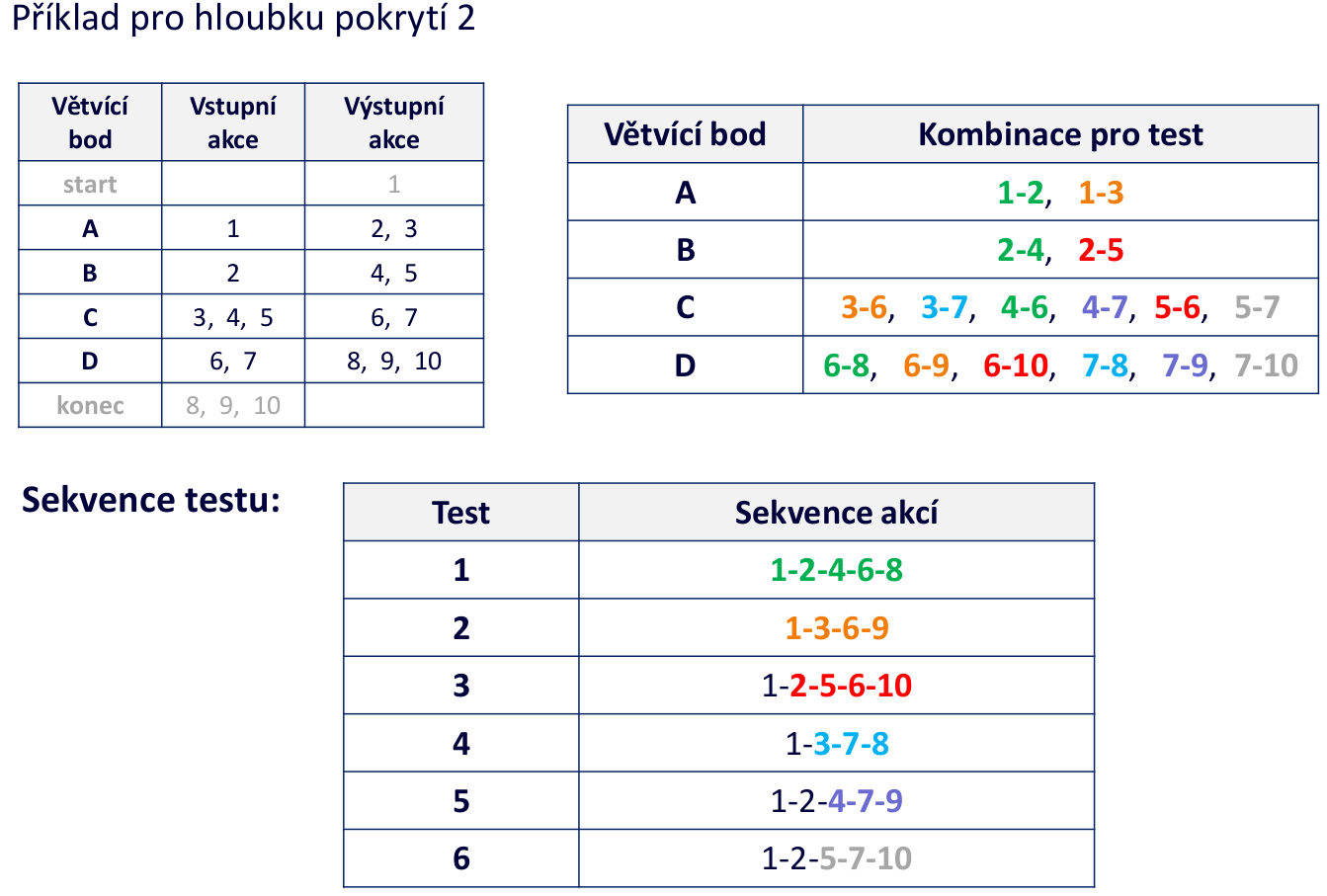


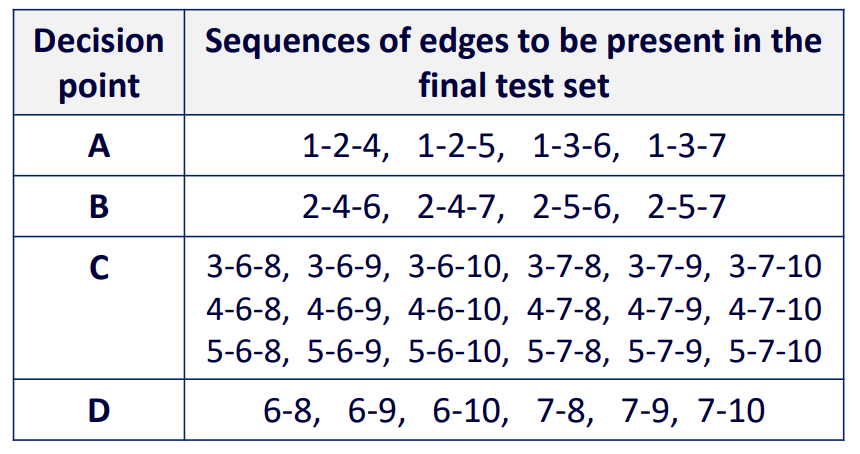


# Testování procesů

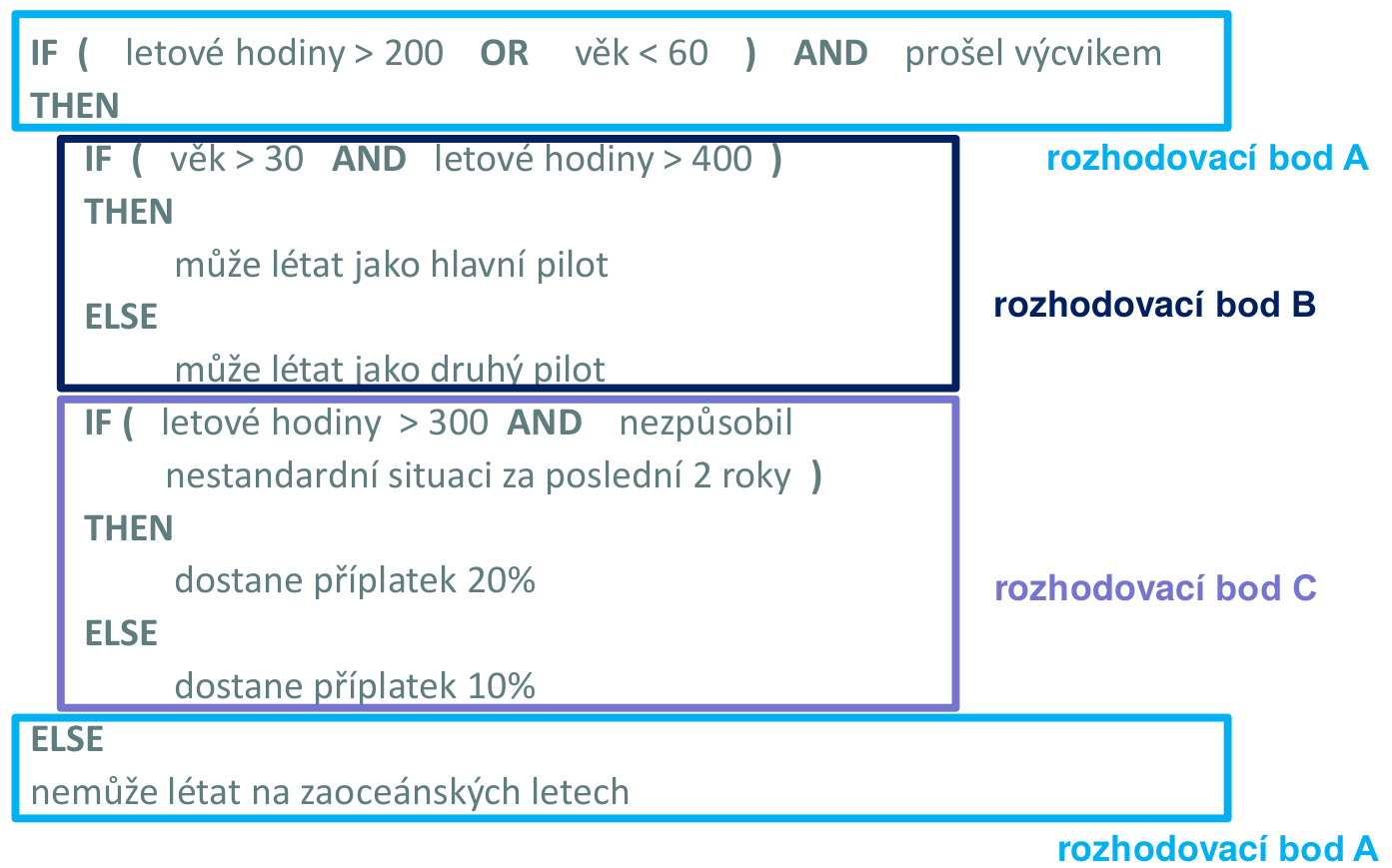
* **Path Based Testing** 
  + Kombinace sekvence akcí pro maximální účinnost
  + UML activity diagram, pro zjednodušení pouze větvící body
  + **Hloubka pokrytí N (TDL - test depth level)** - jistota, že jsou prokryty všechny kombinace N na sebe navazujících akcí
    - Hloubka pokrytí 2 = pro každý uzel provedeme všechny kombinace vstupních akcí do uzlu a výstupních akcí z uzlu
  + Zkombinujeme kombinace akcí za sebou do výsledných testů
    - Minimalizace počtu kroků testu, pokrytí všech akcí
    - Pokud máme na výběr (kombinace již byla pokryta), preferujeme důležitější/více rizikovou cestu

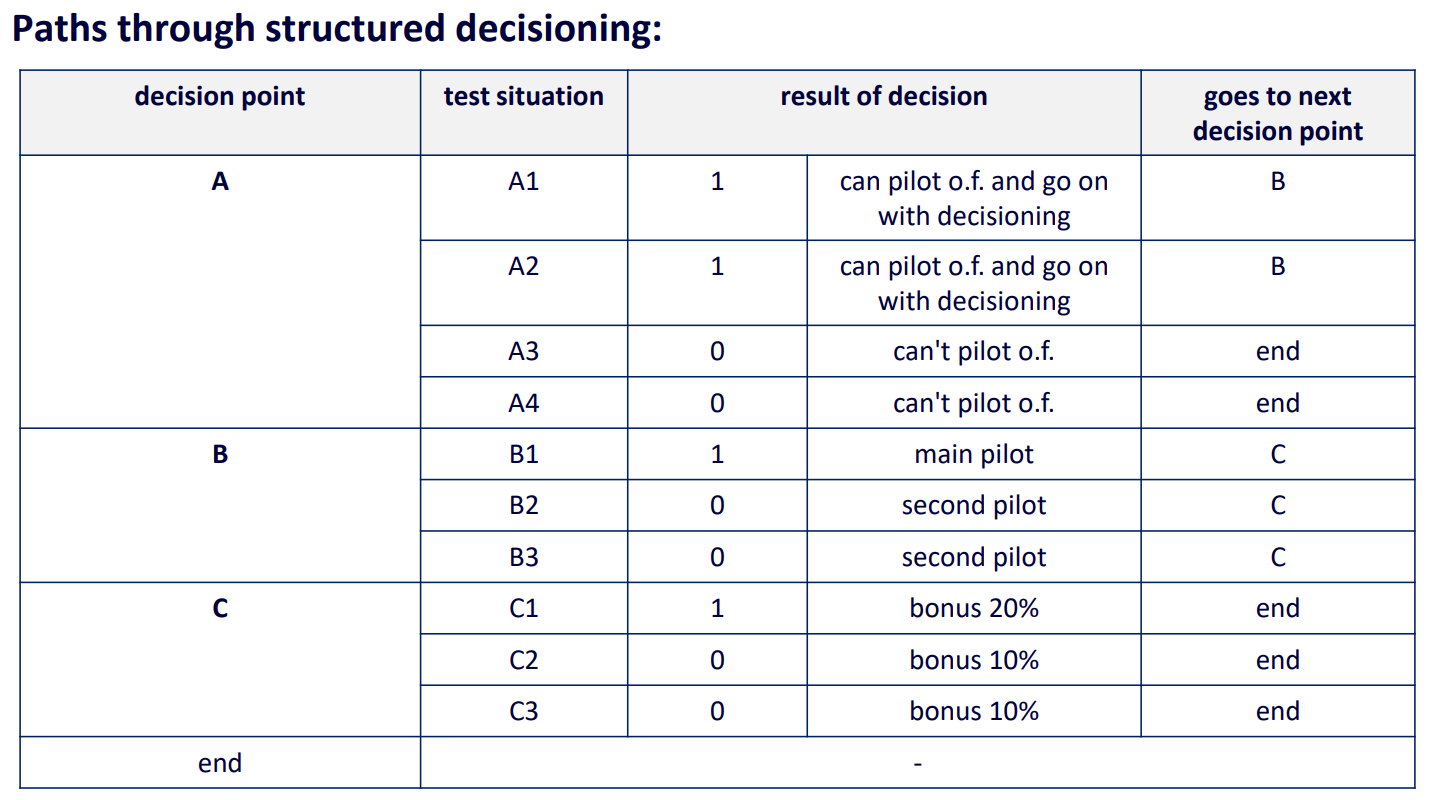


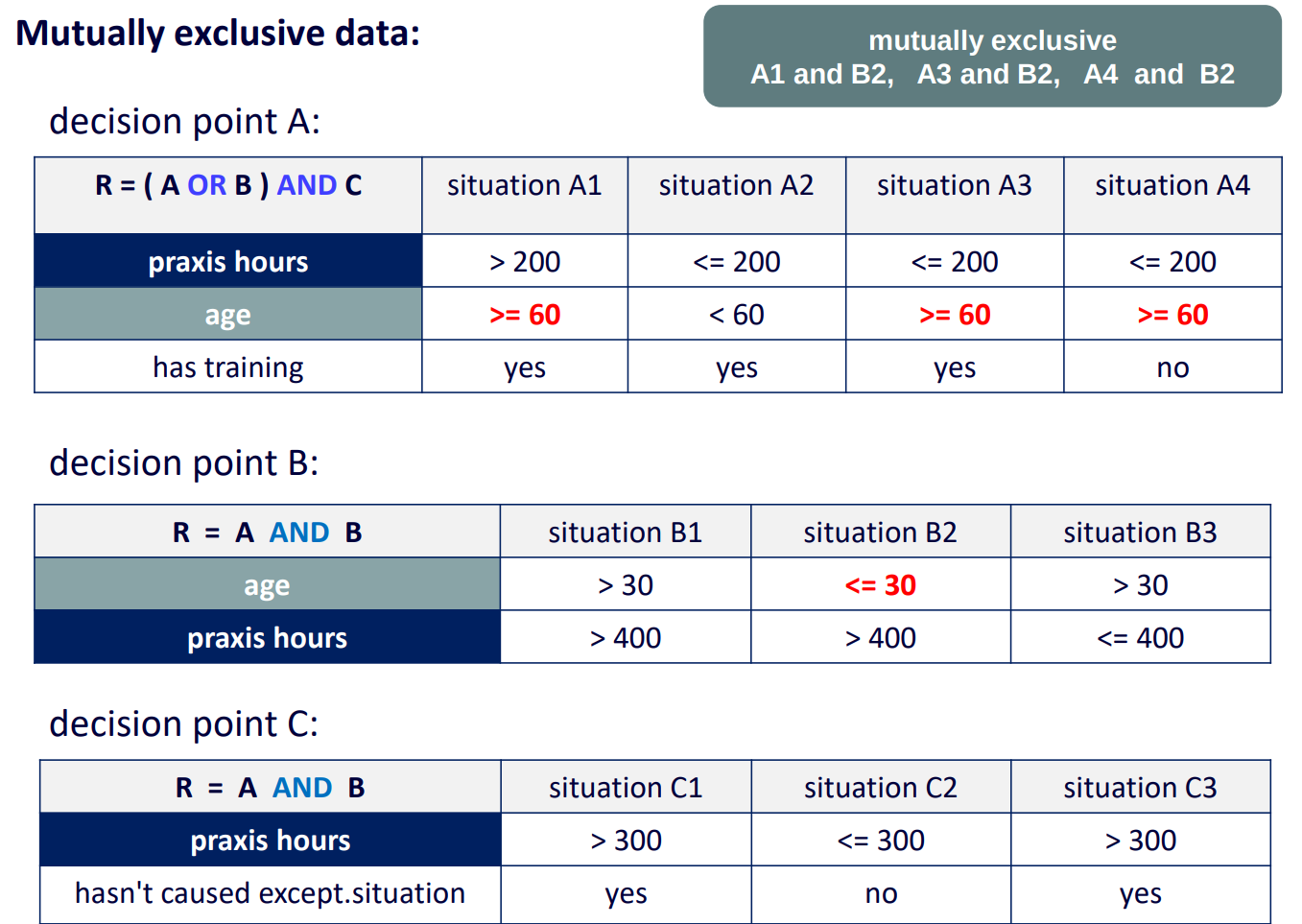


TLD = 3

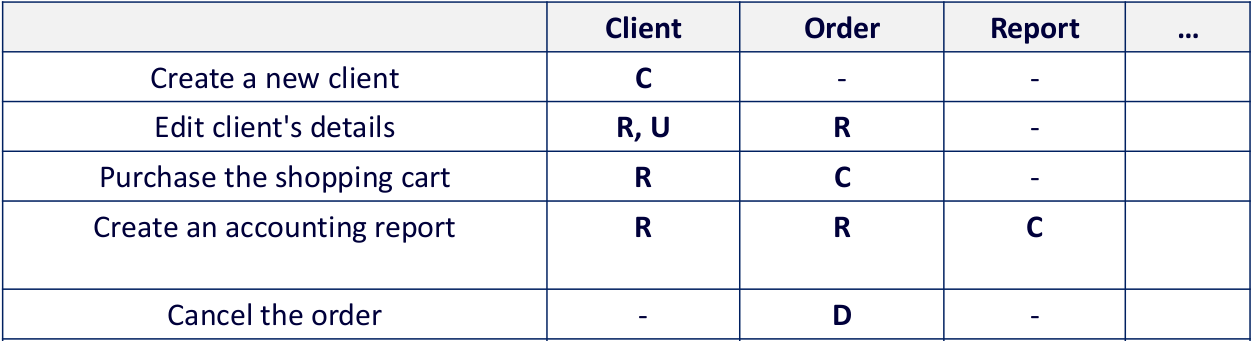
* **Elementary comparison test**
  + Kombinace obou technik - sekvence akcí a kombinace vstupů
  + Každá podmínka ovlivnila každou možnou cestu ve složeném výrazu
  + Velké vnořené logické výrazy







* + Kombinace vstupních dat v bodech A, B, C podle úrovně pokrytí (MC/DC, CC, ...) - parametry voleny tak, abychom prošli všechny cesty
  + Některé vstupy jsou společné, vzájemně výlučná data
* **Exploratory testing**
  + Paralelní učení, návrh testů a jejich provádění
  + *Free testing* - testeři volně procházejí aplikaci bez testovacích scénářů, hlásí chyby, průchod se nezaznamenává, sporná kontrola dokumentace, silně závisí na morálce testerů
  + *Exploratory testing* - Je zaznamenáno, co bylo otestováno, obecná technika pro testování, pokud není k dispozici dostatečná test basis
    - Záznam cesty - vytváření test cases v průběhu testů, navázání na dokumentaci, pokud existuje, v opačném případě rekonstrukce struktury aplikace (model) a navázání test cases na ní
  + Není doložené pokrytí aplikace testy
* **Model of system under test (SUT)**
  + Digraf G = (D, A), D = decision points, A = množina akcí (hrany)
  + Definován start a end decision point
  + Akce a je buďto abstrakce jednoho kroku v SUT, nebo lineární sekvence akcí, která nemá alternativní cestu
  + *Test case t* je sekvence decision points a akcí
* **Test requirements**
  + Klasicky - pod-cesty v grafu G
  + Alternativně - více úrovní, přidání vah do grafu podle priority akce
* **(Prioritize when selecting from alternative subcombinations)**
  + Prioritizace hran HIGH/MEDIUM/LOW
  + Pod Kombinace podle test depth level
  + Pokud při kombinaci scénářů opakuju již použitou hranu, použiju tu, která má vyšší prioritu
    - pro TDL>1: porovná se **priorita podkombinací** vypočítaná podle vzorce
    - *P = (součet vah hran v podkombinaci) / (počet hran v podkombinaci)*
    - váha pro HIGH je 5, váha pro MEDIUM je 3, váha pro LOW je 1
* kombinacemi tak, aby šly vykonat
* Testování procesů na základě grafu nemusí být dostačující.Např. není testována konzistence dat
* **Data lifecycle test**
  + Testování sekvence CRUD (read a update typicky opakováno vícekrát)
  + Celý životní cyklus entity totiž není typicky reprezentován jedním procesem
  + Mezi procesy a entitami je vazba M:N (proces pracuje s více entitami a naopak)
  + Možné defekty - některé z CRUD operací nejsou vůbec implementovány, nebo se nechovají korektně, nekonzistentní chování soft delete, …
  + *CRUD matice* - zobrazuje vazby procesů na odpovídající operace nad entitami

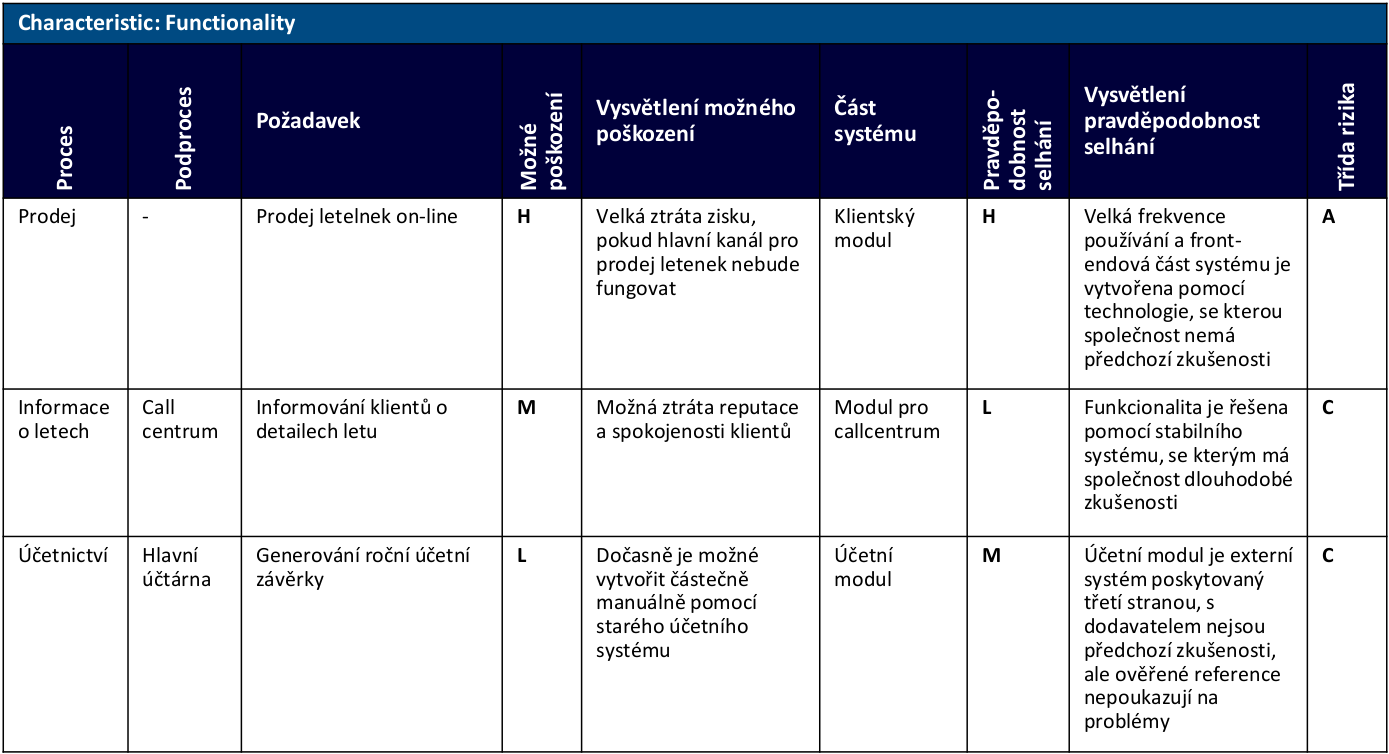


* + Test úplnosti - statický test, kontroluje, zda mají všechny entity všechny 4 operace
    - Pokud ne, nemusí to znamenat defekt
  + Test datové konzistence - dynamický test
    - Kontrola, zda nějaká funkce nenechává data v nekonzistentním stavu
    - Stavíme scénáře z operací s CRUD operacemi
    - Začínáme s C, následně kombinace R a U a konec s D
    - Po každém C, U, D použít R
      * Jednou pro nejdůležitější funkce - *basic coverage*
      * Vícekrát pro různé zobrazovací funkce - *complete R coverage*
  + Můžeme omezit operace podmínkami (např. Uživatel může být smazán pouze pokud má zaplacené všechny objednávky)
    - Podmínka byla splněna a akce byla vykonána
    - Podmínka nebyla splněna a systém neumožní vykonat akci
* **Process cycle test (TMap)**
  + Strukturální test - postaven na toku v aplikaci, ne na základě návrhu/specifikace
  + <http://wawewi.com/cover/usecasetesting2en.html>

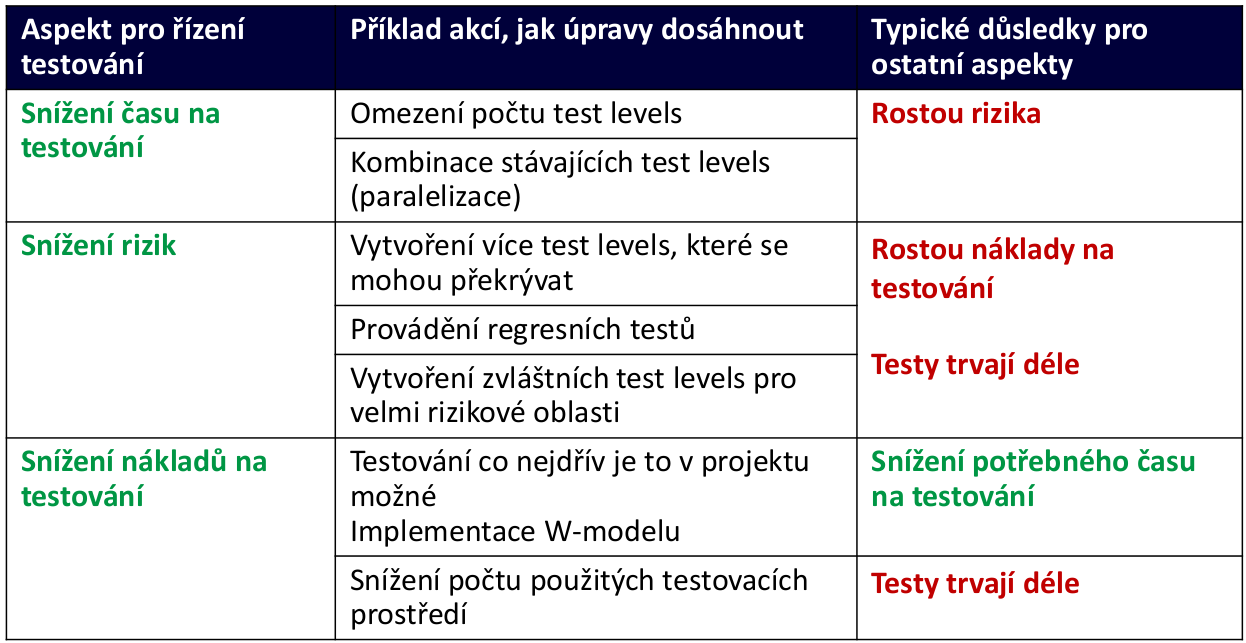
# 

# Testovací strategie

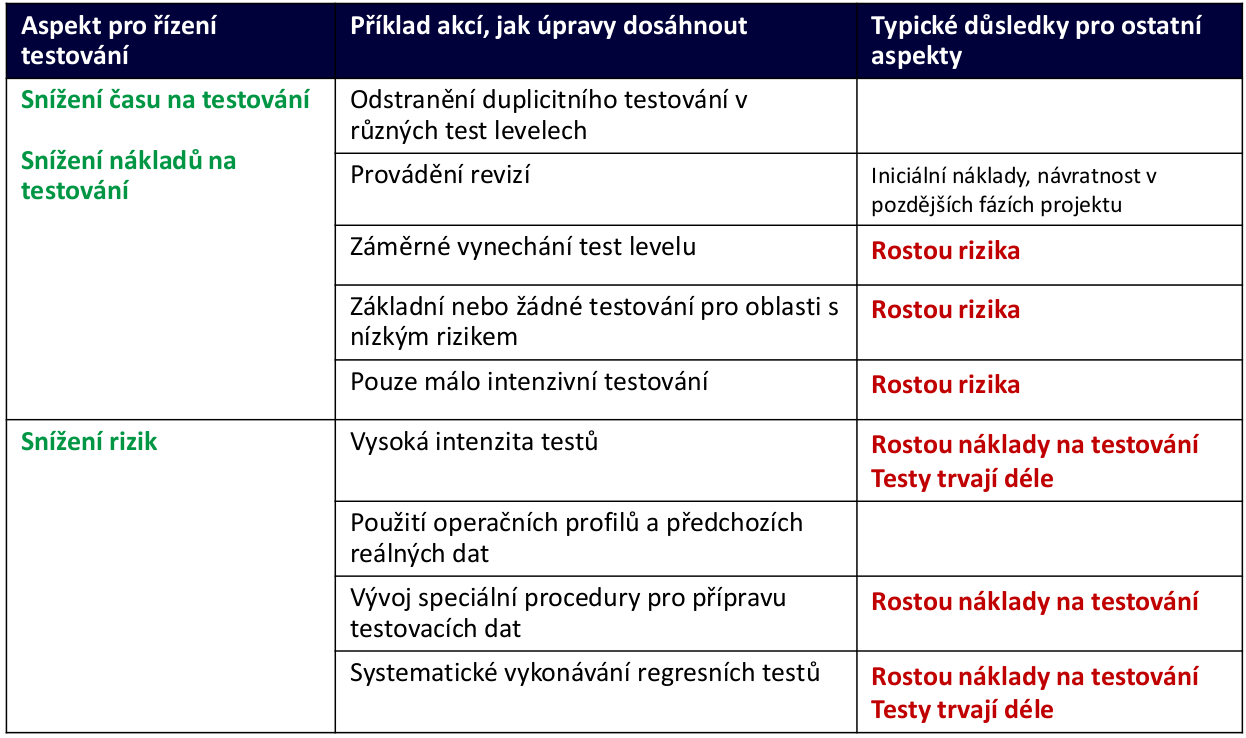
* **Řízení testů**
  + Podprojekt v procesu vývoje SW, má svůj projektový plán
  + Práce test managera je podobná práci project managera
  + Optimalizace aspektů - kvantita, kvalita, termíny, náklady, morálka týmu
* **Entry criteria** **(vstupní podmínky)**
  + Sada podmínek, které musí být splněny, aby mohla aktivita začít
* **Exit criteria (výstupní podmínky)**
  + Sada podmínek a kritérií, které musí být splněny, aby byla aktivita považována za splněnou
* **Testovací strategie**
  + High level popis test levels a testů prováděných v rámci těchto levels na úrovni programu (program se skládá z více projektů)
  + Efektivní řízení testů - systém pro určení priorit (analýza rizik) -> intenzita testů, trasovatelnost, rychlá orientace pro zjištění stavu testování/rizik/aplikace
  + Metoda Business Driven Test Management (BDTM) - TMAP
    - Pro větší testovací projekty, škálovatelná i na menší
* **Test approach**
  + Implementace testovací strategie pro specifický projekt
  + Zahrnuje typicky cíle testování, zhodnocení rizik, test design techniky, typy testů
* **Test plan**
  + Dokument, který popisuje scope, přístup, zdroje a harmonogram testovacích aktivit
  + Popisuje části systému na testování a jejich funkce, úkoly, test design techniky, případný záložní plán
* **Master Test Plan (MTP)**
  + Test plan typicky pro více test levels
  + Poslední čtyři pojmy jsou používány dost vágně
  + MTP je třeba průběžně aktualizovat
* **Test goals**
  + Nezávislý popis, co je cílem testování (“Systém musí umět ...”)
  + BDTM eviduje overall test goals (od investora/managera) a department test goals (typicky od budoucích uživatelů systému)
  + Test goals mohou odhalit rozpory v očekáváních investora a budoucích uživatelů
  + Nastavení konkrétnějších cílů testů s účastníky
  + Lze hierarchicky strukturovat
* **Popis testovaného systému**
  + Požadavky, vazba na procesy/části systému, seznam funkcí systému (nutný)
  + Toto dáme do vztahu s charakteristikami kvality
    - Správná funkcionalita, výkon systému, bezpečnost, UX
  + Kontrola konzistence - jsou testo goals, které nejsou pokryté požadavky/naopak, je implementována funkcionalita, která není v požadavcích apod.
* **Určení priorit**
  + Metoda **Product Risk Analysis (PRA)**
    - Podle priorit je určena intenzita testování
    - Analýza pravděpodobnosti selhání a míry poškození v případě selhání
  + Pravděpodobnost selhání
    - V úvahu je třeba vzít frekvenci používání funkcionality, zkušenost týmu s technologií, problematická integrace, historická chybovost, …
  + Možné poškození
    - Typicky z pohledu zadavatele
    - Ztráta objemu obchodu, pokuty, ztráta reputace, soudní spory, …
  + Na základě priorit je určena **třída rizika**
    - Typicky tři stupně (A, B, C)
  + Záznam do jedné tabulky



* + Kontrola konzistence - vazba rizik na test goals, není významný rozpor mezi tím, co vidí investoři a uživatelé jako hlavní test goals a reálnou rizikovostí na základě analýzy?
  + Neúplné informace
    - Odhad priority nastavit na medium, pokud chybí jedna informace (pravděpodobnost selhání/míra poškození), pokud chybí obě, není možný odhad
    - Nebo expertní odhad
    - Každopádně je třeba označit odhad jako nepřesný
* **Test levels**
  + Na jaké úrovni budou prováděny testy
  + Nejprve definice test levels
    - Např. Revize/Vývojářské testy/Systémové testy/UAT/Testy v produkci
  + Pro jednotlivé části systému/funkce je určena intenzita testování (jaké typy testů budou použity a jaká bude jejich intenzita)
  + Úprava test levels podle aspektů pro řízení testování:



* + Úprava intenzity testování podle aspektů pro řízení testování:



* Další kroky při přípravě testovací strategie:
  + Přiřazení test design technik odpovídající intenzitě testování
  + Odhady pracnosti
  + Revize
  + Rozvedení do detailního plánu testů - aktivity, milníky, zdroje
* Komunikace v průběhu s project managerem
* Informovanost předem o testovacím plánu

# 

# Jednotkové testy

* Základní struktura Arrange -> Act -> Assert (Given/When/Then)
* Rozdělení do packages, aby kopírovaly strukturu zdrojových souborů
* Test Suite - sada unit testů, spustitelná samostatně jako celek
* Mockování
* Výjimky - @Test(expected = IllegalArgumentException.class)
  + Pro asserty try/catch, nezapomenout na fail, pokud nebyla vyhozena očekávaná výjimka
* Parametrizace vstupních dat - Kolekce v kódu, testovací DB data, soubor

# Testovací strategie

**Plán testování** považuji v návrhu postupu testování softwaru za stěžejní dokument pro celý proces. Při realizaci návrhu jej doporučuji vytvořit jako první dokument před samotným začátkem testování softwaru. Při realizaci postupu testování softwaru musí být jakousi „biblí“ pro každého člena testovacího týmu a musí obsahovat odpovědi na veškeré otázky ohledně procesu testování daného software. Pokud na nějakou otázku z jakýchkoliv důvodů nemůže být odpověď uvedena přímo v dokumentu, bude obsahovat odkazy na místa (nebo osoby), kde se odpověď nalézá.

Plán testování obsahuje především rozsah postupu testování software, [definuje druhy a kategorie testů](http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu/), stanovuje harmonogram prací.

Plán testování musí minimálně obsahovat:

* **Cíle testování** – definice toho, co očekáváme přesně od provedení testů nad softwarem. Někdy to může být jen ověření nové funkčnosti, jindy naopak kompletní otestování celé aplikace.
* **Seznam plánovaných oblastí k testování** – Přestože by cílem testování při realizaci měly být všechny funkce software, je nutné vytvořit seznam těchto oblastí. K těmto oblastem také navrhuji přiřadit prioritu, která bude určovat, v jakém pořadí budou oblasti otestovány.
* **Kategorie testů** – [kategorie testů](http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu/), které budou využity během testovacího cyklu. Případně jejich využití v různých úrovních testování
* **Seznam testovacích případů** – seznam unikátních identifikátorů testovacích případů. Navrhuji je seřadit podle pořadí, v jakém budou spouštěny
* **Definice rizik pro testování** – definice hlavních rizik, které mohou znemožnit testování. Jejich podrobný seznam s ohodnocením a návrhem řešení těchto situací je zpracován v analýze rizik
* **Požadavky na testovací data** – definice dat, které jsou nezbytná pro provádění testů.

