**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

FRANTIŠEK CABADAJ

**Vývoj automatizovaného testovacieho systému pre kontrolu komunikácie medzi SAP a externými úložiskami**

Vedúci práce: doc. Ing. Marek Kvet, PhD.

Registračné číslo: 556425

Žilina, 2020

**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA ......................................

BAKALÁRSKA/DIPLOMOVÁ/

DIZERTAČNÁ PRÁCA

ŠTUDIJNÝ ODBOR:

MENO PRIEZVISKO

**Názov práce**

Žilinská univerzita v Žiline

Fakulta ..................

Školiace pracovisko..............

Žilina, 2016

#### Úvodné pokyny pre použitie šablóny

Väčšina nadpisov šablóny je previazaná na dokument Pokyny\_pre\_vypracovanie\_ZP.docx. Sú v ňom podrobnejšie informácie o vypracovaní záverečnej práce. Linky budú správne fungovať vtedy a len vtedy keď si uložíte dokument so záverečnou prácou do toho istého priečinka, v ktorom sa nachádza uvedený dokument. Priamo z tejto šablóny nefungujú.

#### (Pri vytváraní svojho dokumentu pokyny pre použitie šablóny vymažte!)

#### [Čestné vyhlásenie](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Čestné_vyhlásenie)

Podľa národnej legislatívy nie je v záverečnej práci povinné.

#### [Poďakovanie](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Poďakovanie)

Nie je povinné, vysvetlenie a príklad poďakovania

[ABSTRAKT V ŠTÁTNOM JAZYKU](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Abstrakt)

CABADAJ, František: Vývoj automatizovaného testovacieho systému pre kontrolu komunikácie medzi SAP a externými úložiskami. [Bakalárska práca] – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky;

PRIEZVISKO, Meno: *Názov témy záverečnej práce*. [Bakalárska/ Diplomová/ Dizertačná práca]. – Názov univerzity. Názov fakulty; Názov školiaceho pracoviska. – Školiteľ/Vedúci: (tituly pred menom) Meno a priezvisko (tituly za menom) – Stupeň odbornej kvalifikácie: bakalár/magister/iný. – Mesto: skratka fakulty a univerzity, rok predloženia. Počet strán (napr. 35 s.)

[Vysvetlenie a príklad.](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Príklad_abstrakt)

[**Kľúčové slová**](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Kľúčové_slová)**:**  (jednoslovné alebo viacslovné termíny, ktoré charakterizujú vecný obsah práce)

[ABSTRAKT V CUDZOM JAZYKU](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Abstrakt_CJ)

[Vysvetlenie.](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Abstrakt_CJ)

**Key words:**

[Obsah](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Obsah)

Zoznam obrázkov 8

Zoznam tabuliek 9

Zoznam skratiek 10

Úvod 11

1 Teoretické východiská 13

1.1 Životný cyklus informačného systému https://raygun.com/blog/software-development-life-cycle/ 13

1.2 Testovanie 17

1.2.1 Funkčné (functional) testovanie 17

1.2.2 Ne-funkčné (Non-functional) testovanie 19

1.2.3 Automatizované testovanie https://testguild.com/automation-testing/ 20

Manuálne testovanie 22

1.3 SAP 23

1.3.1 Jazyk ABAP 27

2 Ciele práce 30

3 Praktická časť 31

3.1 Analýza externých úložísk a identifikácia scenárov 32

3.1.1 Apache Hive https://aws.amazon.com/big-data/what-is-hive/ 33

3.2 Návrh automatizovaného testovacieho systému 36

3.3 Implementácia automatizovaného testovacieho systému 39

3.3.1 Implementácia interface pre triedy pracujúce s testovacími scenármi 39

3.3.2 Implementácia abstraktnej triedy pre testovacie scenáre 39

3.3.3 Implementácia triedy pre vytvorenie tabuľky na externom úložisku 41

3.3.4 Implementácia triedy pre čítanie dát z tabuľky na externom úložisku 42

3.3.5 Implementácia triedy pre zmenu existujúcej tabuľky 43

3.3.6 Implementácia triedy pre spúšťanie testovacích scenárov SM 44

3.4 Proces spustenia automatizovaného testovacieho systému 46

3.5 Vyhodnotenie dosiahnutých výsledkov 47

4 Metodika práce a metódy skúmania 48

5 Výsledky práce a diskusia 49

5.1 Výsledky práce 49

5.2 Diskusia 49

Záver 50

Zoznam použitej literatúry 51

Zoznam príloh 52

Prílohy 53

Príloha A: Názov prílohy 54

Príloha B: Obsah DVD 55

# [Zoznam obrázkov](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_obrázkov)

**Nenašli sa žiadne položky zoznamu obrázkov.**

# [Zoznam tabuliek](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_obrázkov)

**Nenašli sa žiadne položky zoznamu obrázkov.**

# [Zoznam skratiek](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_skratiek)

ANSA Automated Network Simulation and Analysis

BFD Automated Network Simulation and Analysis

Cisco IOS Cisco Internetwork Operating System

# [Úvod](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Úvod)

Úvodný text sa píše v štýle odseku textu Normálny. Ten je v tejto šablóne nastavený a neodporúčame ho meniť.

Zásady spracovania záverečnej práce sú uvedené [TU](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Zásady_spracovania_ZP).

Formálna úprava záverečných prác je uvedená [TU](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Formálna_úprava_ZP)

#### Pokyny pre používanie šablóny

V šablóne sú pripravené štyri typy nadpisov:

* **Nadpis 1** je číslovaný a začína vždy na novej stránke (dá sa priradiť klávesom **F5**)
* **Nadpis 2** nadväzuje číslovaním na Nadpis 1 (**F6**)
* **Nadpis 3** nadväzuje číslovaním na predchádzajúce nadpisy (**F7**)
* **Nadpis 4** je bez čísla (**F8**)

V šablóne sú pripravené ďalšie štýly odseku textu, pri ktorých odporúčame nemeniť ich nastavené vlastnosti:

* **Normálny** je určený na písanie bežného textu (**F4**)
* **Obrázok** je zarovnaný na stred a určený pre obrázky a ich popis (**F10**)
* **Skratky** je určený na písanie zoznamu skratiek použitých v dokumente
* **Literatúra** je určený na písanie zoznamu literatúry (ešte nie je pripravený)
* **Tabuľka** je určený na vkladanie textu do tabuliek
* **Typ práce** je určený na vypísanie typu práce do hlavičky

#### Pokyny pre prácu s ilustráciami

Ilustrácie sú obrázky obsahujúce grafy, diagramy, mapy, schémy a pod. Nie je potrebné rozlišovať rozličné typy ilustrácií, stačí, ak sa všetky označia ako „Obrázok”.

#### Vloženie ilustrácie

Obrázky a tabuľky vkladáme v tejto šablóne takto:

1. Karta **Vložiť** → **Obrázok**
2. Na obrázku klikneme na pravé tlačidlo myši → **Vložiť popis**
3. Do poľa **Popis** dopíšeme názov obrázku
4. Na obrázok a jeho popis použijeme štýl **Obrázok** (**F10**)

V texte sa na vytvorené obrázky a tabuľky odkazujeme krížovými odkazmi. Krížový odkaz na obrázky a tabuľky vytvoríme takto:

1. Nastavíme sa kurzorom tam, kde sa má objaviť odkaz,
2. Na karte **Vložiť** → **Krížový odkaz**
3. V dialógovom okne pre **Krížový odkaz** v poli **Typ odkazu** vyberieme **Tabuľka** alebo **Obrázok**
4. V poli **Vložiť odkaz** vyberieme **Iba menovka a číslo**
5. Zo zoznamu popisov vyberieme tabuľku alebo obrázok a zaškrtneme **Hypertextový odkaz**
6. Klikneme na **Vložiť**

#### Aktualizácia krížových odkazov a popisov obrázkov

Pokiaľ v texte vymažete alebo pridáte ilustráciu, tak je potrebné prečíslovanie ilustrácií a krížových odkazov:

1. Vyberte celý dokument (**Ctrl + A**)
2. Stlačte kláves **F9**

Obrázky by mali byť kreslené v rovnakom štýle s popisom v jazyku práce. Podrobnejší popis pre prácu s ilustráciami nájdete [tu](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Ilustrácie).

#### Pokyny pre prácu s tabuľkami

Každá tabuľka musí mať poradové číslo a titulok, umiestnený zvyčajne nad tabuľkou, pričom titulok je zarovnaný na pravú stranu tabuľky. Tabuľky by mali byť rovnako naformátované s rovnakým vzhľadom buniek.

Podrobnejší popis pre prácu s tabuľkami nájdete [tu](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Tabuľky).

#### Zoznam použitej literatúry

Bibliografické odkazy vkladajte cez kartu **Referencie,** zoskupenie nástrojov **Citácie a bibliografia**.

# Teoretické východiská

## Životný cyklus informačného systému <https://raygun.com/blog/software-development-life-cycle/>

Softvér je komplexný produkt, ktorý je vyvíjaný a dodávaný zákazníkovi pomocou série krokov. Ako iné produkty aj vývoj softvéru začína s myšlienkou. Táto myšlienka sa postupne stáva dokumentom alebo prototypom, čo záleží na forme metódy ktorá sa na vývoj používa.

Či už je to dokument, diagram alebo funkčný softvér, artefakt[1] vytvorený v jednom kroku sa stáva vstupom do ďalšieho. Nakoniec je softvér dodaný zákazníkovi. Postupnosti krokov, použitých v týchto metódach sa spoločne hovorí Software Development Lifecycle (SDLC).

Artefakt[1] – je jedným z mnohých druhov vedľajších produktov vyrobených počas vývoja softvéru ako napr. Unified Modeling Languare (UML) diagram.

Proces vývoja softvéru je nikdy nekončiaci cyklus. Prvé nasadenie softvérovej aplikácie je zriedkavo považované za dokončené. Takmer vždy sa vyskytnú chyby dizajnu alebo funkcie, čakajúce na opravu.

Chyby, ktoré boli zachytené používateľmi softvéru sa vracajú späť do procesu vývoja softvéru a stávajú sa novými požiadavkami na vylepšenie existujúcich funkcií. Práve z tohto dôvodu je SDLC najvšeobecnejší pojem pre metódy vývoja softvéru. Kroky procesu ako aj ich poradie je jedinečné pre každú metódu, majú však spoločné to, že zvyčajne bežia v cykloch, začínajú odznova s každým opakovaním.

„ Ak zlyháš pri plánovaní, plánuješ zlyhať“ – Benjamin Franklin

Je veľmi náročné vykonávať komplexnú tímovú prácu ako je vývoj softvéru bez nejakého druhu plánu. Každá metóda vývoja softvéru má svoje výhody a nevýhody. Existuje veľa diskusií o tom, ktorá z nich je najvhodnejšia pre konkrétny typ softvéru a ako merať jej úspech. Jedna vec je však istá, akýkoľvek plán je lepší ako žiaden.

Bez nejakého štruktúrovaného plánu, majú tímy vývoja softvéru tendenciu prejsť do stavu, kedy nevedia ako budú pokračovať. Projektoví manažéri netušia aký veľký krok sa urobil k dosiahnutiu cieľa projektu. Spoločnosť sa bez plánu nemá ako uistiť, či konečný produkt spĺňa ich požiadavky.

Formálne definovaná metóda pre vývoj softvéru vo forme SDLC dokáže zaistiť viacero výhod:

* Spoločný slovník pre každý krok
* Sú definované komunikačné kanály medzi vývojovým tímom a zúčastnenými stranami
* Jasne stanovená zodpovednosť medzi vývojármi, dizajnérmi, obchodnými analytikmi a projektovými manažérmi
* Jasne definované vstupné a výstupné parametre z jedného kroku do ďalšieho
* Deterministická definícia toho čo sa vykonalo, ktorá sa môže použiť na potvrdenie, či je krok skutočne dokončený

Tieto kroky sú zhruba rovnaké medzi jednotlivými metódami. Majú tendenciu vyskytovať sa v tomto poradí aj keď sa môžu navzájom spájať tak, že sa niekoľko krokov vykonáva paralelne.

Agilné metodiky majú tendenciu zoskupiť tieto kroky do opakujúceho sa cyklu. Vodopádová metóda zvykne podniknúť každý z týchto krokov postupne. Výstupy z jedného sa stanú vstupmi do nasledujúceho kroku.

**Obrázok 1. Životný cyklus informačného systému**

1. **Plánovanie**

Fáza plánovania môže zahŕňať:

* Plánovanie kapacít
* Plánovanie projektu
* Odhad nákladov
* Prideľovanie zdrojov (ľudských aj materiálnych)

Medzi výsledky tejto plánovacej fázy patria: plán projektu, rozvrh, odhad nákladov a požiadavky na obstarávanie. V najlepšom prípade projektoví manažéri a vývojári spolupracujú so zástupcami plánovania a bezpečnosti aby zabezpečili zastúpenie všetkých perspektív.

1. **Požiadavky**

Zákazník musí komunikovať s IT tímom pri stanovení požiadaviek na vývoj a vylepšenie požadovaného softvéru. Fáza požiadaviek zhromažďuje tieto požiadavky od zákazníka a Subject Matter Experts (SME).

1. **Dizajn**

Keď už máme jasne stanovené požiadavky, architekti a vývojári môžu začať navrhovať softvér. Proces navrhovania používa stanovené vzory na vývoj aplikačnej architektúry a softvéru. Architekti môžu používať architektúru ako The Open Group Architecture Framework (TOGAF) na zostavenie aplikácie z existujúcich komponentov, podporu opätovného použitia a štandardizácie.

Vývojári používajú overené návrhové vzory na konzistentné riešenie algoritmických problémov. Táto fáza môže zahŕňať aj niekoľko techník Rapid prototypingu, taktiež nazývaných Spike, ktoré slúžia na porovnanie riešení s cieľom nájsť to najvhodnejšie.

Výstup z tejto fázy zahŕňa:

* Návrhové dokumenty v ktorých sú uvedené vzory a komponenty vybrané pre projekt
* Kód vytvorený technikou Spike, ktorý sa používa ako východisko pre vývoj

1. **Vývoj softvéru**

Táto fáza vyprodukuje vyvíjaný softvér. V závislosti od metodológie môže byť táto fáza vykonávaná v časovo obmedzených šprintoch (Agilná metodika) alebo ako TODO. Bez ohľadu na metodiku by vývojové tímy mali čo najrýchlejšie vyprodukovať funkčný softvér. Zákazník by sa mal pravidelne zapájať do vývoju, aby sa splnili stanovené požiadavky. Výstupom z tejto fázy je funkčný testovateľný softvér.

1. **Testovanie**

Fáza testovania je v SDLC je pravdepodobne jednou z najdôležitejších, pretože bez riadneho testovania nie je možné dodať kvalitný softvér. Existuje široké spektrum testov potrebných na určenie kvality ako napríklad:

* Unit testovanie
* Integračné testovanie
* Testovanie výkonu
* Testovanie bezpečnosti

Najlepším spôsobom, ako zaistiť, aby sa testy nezanedbávali a vykonávali pravidelne, je ich automatizácia. Výstupom testovacej fázy je funkčný softvér pripravený na nasadenie do produkčného prostredia resp. k zákazníkovi.

1. **Nasadenie**

Fáza nasadenia je ideálne vysoko automatizovaná. Vo vyspelých spoločnostiach je táto fáza takmer neviditeľná, softvér je nasadený hneď ako je dokončený. V menej vyspelých spoločnostiach alebo v niektorých viac regulovaných odvetviach tento proces vyžaduje určité manuálne schválenia. Avšak aj v týchto prípadoch je najlepšie aby samotné nasadenie bolo plne automatizované. Nástroje Aplication Release Automation (ARA) sa používajú v stredných a veľkých podnikoch na automatizáciu nasadzovania aplikácií do produkčných prostredí. Systémy ARA sú zvyčajne integrované s nástrojmi na kontinuálnu integráciu. Výstupom z tejto fázy je nasadenie funkčného softvéru do produkčného prostredia.

1. **Údržba**

Fáza údržby je akoby začiatkom konca. Životný cyklus informačného systému tu nekončí. Softvér sa musí neustále monitorovať aby sa zabezpečila správna prevádzka. Chyby a nedostatky zistené v produkcií sa musia nahlásiť a spracovať, čo často vedie k tomu, že sa vyžaduje dodatočná práce na projekte. Oprava chýb nemusí pretekať celým cyklom, je však potrebný aspoň skrátený postup, aby sa zabezpečilo, že oprava nespôsobí ďalšie problémy známe ako regresia.

## Testovanie

Ako sa už spomínalo pri fáze testovania v SDLC, testovanie je jedným z najdôležitejších krokov k vývoju funkčného softvéru. Práve z tohto dôvodu existuje široké spektrum testov. <https://www.edureka.co/blog/functional-testing-vs-non-functional-testing/>

** Obrázok 2. Typy softvérového testovania**

Typy softvérového testovania sa delia do dvoch základných skupín a to funkčné (functional) a ne-funkčné (non-functional) testovanie.

### Funkčné (functional) testovanie

Funkčné testovanie je typ softvérového testovania, pri ktorom sa systém testuje na základe funkčných požiadaviek alebo špecifikácií, ako sú technické detaily, manipulácia a spracovanie údajov a ďalšie špecifické funkcie. Možno si myslíte, že funkčné testovanie sa týka iba testovania funkcie (metódy) nejakého modulu alebo triedy, čo ale nie je celkom pravda. Testuje sa, či správne funguje určitá časť celkového systému.

**Typy funkčného testovania:**

**Unit testovanie**

Unit je najmenšia možná testovateľná časť akéhokoľvek softvéru. Zvyčajne má jeden alebo viacero vstupných parametrov a jeden výstupný parameter. Je to úroveň softvérového testovanie, kde sú testované individuálne unity. Hlavným cieľom je overiť, či každý komponent softvéru pracuje tak ako by mal.

**Integračné (integration) testovanie**

Je to úroveň softvérového testovania, kde individuálne unity sú kombinované do testovacích skupín. Hlavným cieľom tohto typu testovania je odhaliť chyby pri interakcií medzi integrovanými unitami.

**Systémové (System) testovanie**

Je úroveň softvérového testovania, kde sa testuje celkový integrovaný softvér. Cieľom tohto testu je vyhodnotiť súlad systému so stanovenými požiadavkami. Je to séria rôznych testov, ktorých jediným účelom je precvičiť celkové fungovanie systému.

**Interface testovanie**

Je to úroveň softvérového testovania, ktorá overuje komunikáciu jedného softvéru s druhým.

**Regresné (Regression) testovanie**

Overuje či zmeny v kóde nemajú dopad na existujúcu funkcionalitu produktu.

**User-acceptance testovanie**

Je úroveň softvérového testovania kde sa testuje prijateľnosť systému. Hlavným účelom tohto testu je vyhodnotiť súlad systému s jeho obchodnými požiadavkami a posúdiť, či je prijateľný na odovzdanie zákazníkovi. Kontroluje, či softvér dokáže zvládnuť požadované úlohy v scenároch s praktickým využitím.

### Ne-funkčné (Non-functional) testovanie

Ne-funkčné testovanie kontroluje ne-funkčné aspekty ako výkonnosť, využiteľnosť, spoľahlivosť atď. softvérovej aplikácie, ktoré sa netestujú vo funkčnom testovaní. Ne-funkčné testovanie pomáha overiť pripravenosť systému.

Definuje skôr spôsob fungovania systému ako jeho konkrétne správanie. Čo je opakom toho o čo sa snaží funkčné testovanie. V zásade, ne-funkčné testovanie vykonáva a vyhodnocuje všetky ne-funkčné parametre, ktoré nie sú zahrnuté do funkčného testovania, ako je rýchlosť, škálovateľnosť, bezpečnosť a efektívnosť aplikácie. Toto testovanie robí aplikáciu robustnou a pripraví ju TODO.

**Typy ne-funkčného (non-functional) testovania:**

**Dokumentačné (documentation) testovanie**

Pomáha odhadnúť požadované úsilie na testovanie a sledovať požiadavky. Softvérová dokumentácia zahŕňa testovací plán, testovacie scenáre a sekciu požiadaviek. Testuje zdokumentované artefakty.

**Inštalačné (installation) testovanie**

Je typ práce pri zabezpečovaní kvality v softvérovom priemysle, ktorá sa zameriava na to, čo budú zákazníci musieť urobiť, aby mohli úspešne nainštalovať a nastaviť nový softvér. Kontroluje, či je aplikácia úspešne nainštalovaná a či pracuje podľa očakávaní. Tento proces môže obsahovať úplné alebo čiastočné inštalácie. TODO

**Výkonnostné (performance) testovanie**

Je definované ako typ softvérového testovania, ktoré je používané na zabezpečenie toho, aby aplikácie pracovali dobre pri očakávanom pracovnom zaťažení. Výkonnostné testovanie sa považuje za srdce ne-funkčného testovania. Ďalej sa delí na niekoľko typov.

* Záťažové (load) testovanie - vyhodnocuje správanie systému pri postupnom zvyšujúcom sa pracovnom zaťažení.
* Stress (dôrazové) testovanie – vyhodnocuje správanie systému na hranici alebo za hranicou zvládnuteľného pracovného zaťaženia.
* Endurance (vytrvalostné) testovanie - vyhodnocuje správanie systému pri veľkej dlho trvajúcej pracovnej záťaži.
* Spike testovanie – vyhodnocuje správanie systému pri náhlom zvýšení pracovnej záťaže.

**Testovanie spoľahlivosti (Reliability)**

Zaručuje, že produkt bude bezchybný a spoľahlivý pre v tom na čo bol navrhnutý. Ide o testovanie aplikácie tak, aby sa chyby objavili pred zavedením systému

### Automatizované testovanie <https://testguild.com/automation-testing/>

Automatizované testovanie je technika, ktorá sa používa na zvýšenie rýchlosti overovania alebo iných opakovaných úloh v životnom cykle informačného systému. Z tohto dôvodu, aby ušetrili čas, sa veľa spoločností snaží zobrať manuálne testovacie scenáre a previesť ich na automatizované testovacie scenáre.

Automatizovaný testovací nástroj potom vykoná kroky testovacieho scenára automaticky bez ľudského pričinenia. Taktiež automatizovaný testovací nástroj používa programátorský prístup aby napodobnil interakciu používateľa s aplikáciou.

Prečo testovať automaticky? (<https://testguild.com/automation-testing/>) Môžu byť otázky?

So zvýšenou rýchlosťou, ktorou sa softvér v dnešnej dobe vyvíja, je automatické testovanie nevyhnutnou súčasťou vývoja nového softvéru. Praktiky ako priebežná integrácia, vývojárska praktika ktorá od vývojárov vyžaduje integrovať kód do zdieľaného úložiska niekoľko krát denne, vyžadujú testy ktoré bežia rýchlo a spoľahlivo. Veľa manuálneho overovania zabráni dosiahnutiu požadovanej rýchlosti vývoja softvéru. Je takmer isté, že v dnešnom modernom vývojovom prostredí, by sme bez automatizovania neuspeli.

Hoci hlavným dôvodom prečo sa tímy snažia testovať automatizovane je aby ušetrili spoločnosti čas a peniaze, dôležité je tiež dodať vývojárom spätnú väzbu aby boli pri odovzdaní kódu čo najskôr informovaný o tom, že zmena kódu ktorú odovzdali niečo pokazila. Ďalšie dôvody pre automatizované testovanie sú:

* Overenie novších verzií softvéru
* Oslobodenie testerov od triviálnych scenárov aby sa mohli sústrediť na komplexnejšie testovacie scenáre
* Väčšie pokrytie testov
* Znovu-použiteľnosť
* Rýchlejšie nasadenie softvéru
* Šetrí čas
* Dodáva rýchlu spätnú väzbu pre vývojárov ohľadom chýb v softvéri

Teória je taká, že automatizované testy ušetria spoločnostiam čas a peniaze. Vyzerá to však tak, že veľa ľudí neberie do úvahy čas a peniaze, ktoré treba investovať do udržania stabilného testovacieho systému.

Keďže na tvorbu automatických testov sa používajú programovacie jazyky, automatizácia sa stáva samostatným projektom na vývoj. Čo sa v skutočnosti robí je, že sa vyvíja časť softvéru na testovanie inej časti softvéru, preto je potrebné investovať rovnaké úsilie do vývoju automatických testov. Dodržujú sa rovnaké procesy a osvedčené postupy ako by sa použili pri vývoji akéhokoľvek iného projektu.

Automatizované testovanie je náročné, presne ako väčšina ostatných projektov na vývoj softvéru. Taktiež prezentuje veľa rovnakých problémov výziev. Ak sa k automatizovanému testovaniu bude pristupovať ako k druhotriednemu vývoju, bude mať problémy s údržbou a spoľahlivosťou z dlhodobého hľadiska.

Proces automatizovaného testovania sa dá zhrnúť do týchto krokov:

 **Obrázok 3. Proces automatizovaného testovania**

**Príprava**

Najskôr je potrebné pripraviť sa a pochopiť ciele funkčného testovania, aké testovacie dáta sú potrebné a čo treba overiť.

**Písanie**

Zmena požiadaviek na automatizované riešenie. Určenie počiatočného a koncového stavu pre každý test. Testy by mali byť úplne nezávislé jeden od druhého. Je potrebné pridať dostatočné kontroly na uistenie sa, že sa aplikácia správa podľa zadaných špecifikácií. Každý test by mal mať konkrétny účel.

**Realizácia**

Realizácia testov by mala byť spoľahlivá. Odporúča sa spustiť každý test aspoň tri krát za sebou predtým ako sa skontroluje kód.

**Vyhodnotenie**

Overenie, či automatizované testy robia to, čo sa od nich čaká. Pomocou manuálnych testov sa overí, či sa testy správajú podľa požiadaviek.

**Komunikácia**

Každý člen tímu by mal byť informovaný o výsledkoch testov. Pochybné testy by mali byť opravené čo najskôr, inak vzniká riziko, že tím bude výsledky testov ignorovať.

**Opakovanie/Spravovanie**

Ak sa ukáže pochybný test, je potrebné ho prerobiť aby sa stal viac spoľahlivým. Najdôležitejšie je však vymazať akékoľvek testy, ktoré nie sú spoľahlivé a neboli opravené v danom časovom rámci.

Automatické testovanie je teda využívané pre uľahčenie práce vývojárov a odľahčenie záťaže na testovacie oddelenie. Existujú však prípady, kedy automatické testovanie nedokáže pokryť všetky scenáre a musia sa testovať manuálne. Preto pri vývoji nového softvéru vo firmách, ktoré nemajú testovací tím, robia vývojári manuálne testy predtým ako sa nový softvér nasadí u zákazníka.

### Manuálne testovanie

Existujú scenáre, ktoré nie je možné vykonať automaticky alebo čas strávený vývojom takéhoto scenára by presiahol jeho užitočnosť. Jedná sa väčšinou o kód Graphical user interface (GUI), kde by vývoj automatického scenáru takmer s istotou pôvodný presiahol čas strávený vývojom GUI. V takomto prípade testuje scenáre testovací tím alebo samotný vývojár, ktorý kód napísal.

Je však niekoľko dôvodov prečo môže byť manuálne testovanie problematické:

* Používa veľa prostriedkov – na manuálne testovanie potrebujeme testovací tím, ktorý môže byť finančne náročný
* Je časovo náročné – celkové pokrytie testovacích scenárov manuálne zaberá veľa času
* Tester niekedy nemusí pokryť celý testovaný scenár – určité scenáre vyžadujú schopnosti programátorov
* Kvôli opakujúcej sa povahe, testeri môžu pri manuálnom vykonávaní zanedbať alebo prehliadnuť niektoré dôležité kroky čo môže viesť k nekonzistentnému stavu testu.

## SAP

What is SAP? <https://www.guru99.com/what-is-sap-definition-of-sap-erp-software.html>

Je to európska medzinárodná spoločnosť založená Dietmarom Hoppom, Hans-Wernerom Hectorom, Hassom Plattnerom, Klausom Tschiraom a Clausom Wellenreutherom v roku 1972. Celosvetovo od roku 2010 má SAP viac ako 140 000 inštalácií, viac než 25 podnikových riešení pre špecifické priemyselné odvetvia a viac ako 75 000 zákazníkov v 120 krajinách.

SAP je skratka pre System aplications and Products je jednotka na ERP trhu v spracovaní dát. Je to štvrtá najväčšia softvérová spoločnosť na svete. SAP R/3 systém je balík biznis softvéru, ktorý je navrhnutý pre integráciu všetkých oblastí podnikania Spoločnosť vyvíja softvérové riešenia pre riadenie obchodných operácií a vzťahov so zákazníkmi.

Po veľkom úspechu R/3, vytvoril SAP AG viac a viac niche\* (Ako sa postaviť k slovíčkam ktoré je vhodné nepreložiť resp. spraviť nejakú legendu) softvér ako napríklad Customer Relationship Management (CRM), SRM, XI (aktuálne nazývaný aj ako Process Integration skratkou PI) čím sa znova ukázala kvalita softvéru tejto spoločnosti.

Všetky podnikové procesy sú vykonávané v jednom SAP systéme a zdieľajú spoločné informácie.

<https://searchsap.techtarget.com/definition/SAP?fbclid=IwAR22o4XxOhTOdOar3gePgFADmEt3Wb_kbEV4HqH7u1ItcwA3OhA8kKCV2I>

Odhaduje sa, že 77% peňazí vymieňaných prostredníctvom globálnych obchodných transakcií sa dostane do kontaktu so systémom SAP. Väčšinu zákazníkov tvoria malé a stredné podniky. Spoločnosť ponúka lokálne, cloudové a hybridné modely na nasadenie s tým, že možnosť cloud computing je to na čo sa chce spoločnosť v budúcnosti sústrediť.

V roku 2011, spoločnosť uviedla na trh SAP HANA, in-memory\* databázovú platformu, ktorá je na čele budúcej stratégie spoločnosti. HANA bola hlavným projektom pre SAP, ktorý uviedol, že má v úmysle nahradiť databázou HANA tradičné databázy, ktoré SAP používa pre svoje obchodné aplikácie.

**Enterprise resource planning (ERP)**

SAP SE je jedným z najväčších dodávateľov softvéru pre Enterprise Resource Planing \*(ERP) a súvisiacich podnikových aplikácií. ERP systém spoločnosti umožňuje zákazníkom viesť ich obchodné procesy vrátane účtovníctva, predaja, výroby, ľudských zdrojov a financií v integrovanom prostredí. Integrácia zabezpečuje tok informácií z jedného komponentu SAPu (skloňovanie cudzích slov môže byť?) do druhého bez potreby redundantného\* zadávania údajov a pomáha vymáhať finančné, procesné a právne kontroly. Umožňuje tiež efektívne využívanie zdrojov, vrátane pracovnej sily, strojov a výrobných kapacít.

SAP ERP systém, nazývaný SAP ERP Central Component (SAP ECC), je spoločný termín pre funkčné a technické moduly spoločnosti SAP, ktoré umožňujú podnikom riadiť obchodné procesy prostredníctvom zjednoteného systému. ECC je miestna verzia SAP, ktorá je zvyčajne implementovaná v stredných a veľkých spoločnostiach. Pre menšie spoločnosti ponúka SAP svoju Business One ERP platformu.

SAP ERP má rozdielne hlavné moduly, ktoré sú rozdelené do funkčných a technických modulov, z ktorých každý má submoduly\*.

Funkčné moduly SAP systému zahŕňajú:

* Human Capital Management ([SAP HCM](https://searchhrsoftware.techtarget.com/definition/human-capital-management-HCM))
* Production Planning ([SAP PP](https://searchsap.techtarget.com/definition/SAP-Production-Planning))
* Materials Management ([SAP MM](https://searchsap.techtarget.com/definition/SAP-Materials-Management-MM))
* Project System ([SAP PS](https://searchsap.techtarget.com/definition/SAP-Project-System-PS))
* Sales and Distribution ([SAP SD](https://searchsap.techtarget.com/definition/SAP-Sales-and-Distribution-SAP-SD))
* Plant Maintenance ([SAP PM](https://searchsap.techtarget.com/definition/SAP-Plant-Maintenance-PM))
* Finance and controlling ([SAP FICO](https://searchsap.techtarget.com/definition/SAP-FICO-SAP-Finance-and-SAP-Controlling))

Quality Management ([SAP QM](https://searchsap.techtarget.com/definition/SAP-Quality-Management-QM))

**História spoločnosti SAP**

SAP začal v roku 1972 piatimi bývalými zamestnancami IBM v Mannheime v Nemecku. Pôvodným cieľom pre SAP bolo poskytnúť zamestnancom možnosť využiť bežnú korporátnu\* databázu na komplexný rozsah aplikácií v reálnom čase.

V roku 1973, SAP vydal R/1, systém finančného účtovníctva. R/1 bežal na IBM serveroch a diskovom operačnom systéme (DOS). Mal jednostupňovú architektúru, v ktorej boli prezentačná, aplikačná a databázová vrstva na jednej platforme.

V roku 1979, SAP vydal R/2, systém s centrálnym počítačom (mainframe), ktorý poskytoval spracovanie údajov v reálnom čase v rámci účtovníctva, výroby, zásobovacieho reťazca a ľudských zdrojov. R/2 používal dvojvrstvovú architektúru, kde prezentačná vrstva bola na jednej platforme a aplikačná a databázová vrstva boli na druhej. R/2 pomohlo podporiť rast spoločnosti SAP a predajca rozšíril svoju klientelu na približne 200 spoločností.

V roku 1992, SAP vydal R/3, ktorý reprezentoval zmenu z \*mainframe na klient-server model a z dvojvrstvovej architektúry na trojvrstvovú architektúru, v ktorej prezentačná, aplikačná a databázová vrstva boli umiestnené osobitne?? R/3 bol kritický produkt pre SAP, ktorý preniesol túto spoločnosť na svetovú scénu.

**Obrázok 4. 3 vrstvová architektúra**

### Jazyk ABAP

<https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm>

ABAP je programovací jazyk vyvinutý spoločnosťou SAP pre vývoj obchodných aplikácií v prostredí SAP. ABAP podporuje objektovo orientovaný programovací model, založený na triedach a rozhraniach a procedurálny programovací model, založený na funkčných moduloch a podprogramoch.

**ABAP Dictionary**

ABAP Dictionary (ďalej len DDIC) je trvalé úložisko metadát dátových typov systému ABAP. Rovnako ako v jazyku ABAP, definovateľné dátové typy v DDIC môžu byť elementárne, štruktúrované alebo tabuľkové. Referenčné dátové typy je tiež možné vytvoriť. Štruktúry v DDIC majú dôležitú úlohu pri popisovaní databázových tabuliek v databáze.

**Klasické objekty v ABAP DDIC:**

**Dátové typy**

Globálne dátové typy sú viditeľné pre všetky objekty aktuálneho SAP systému a môžu byť nimi použité. Názov dátového typu môže mať maximálne 30 znakov, môže obsahovať písmená, čísla a znak „ \_ “.

Musí začínať písmenom alebo predponou (namespace prefix). Predpona môže mať 5-10 znakov, ktoré musia začínať a končiť lomkou (/.../). Predponu pre objekty SAP systémov patriacu SAP oddeleniam, partnerom a zákazníkom je možné vyžiadať od firmy SAP.

Dátové elementy sú elementárne dátové typy alebo referencie so sémantickými atribútmi. Atribúty dátového elementu sú definované buď priamo alebo pomocou domény.

Doména popisuje atribúty dátových elementov, ako je napríklad dátový typ alebo rozsah hodnôt. Doménu môže používať ľubovoľný počet dátových elementov, ale samotný dátový element ju nemusí používať. V iných objektoch SAP systému, konkrétne v ABAP programoch nie je možné priamo použiť doménu, je potrebné využiť dátový element, ktorý ju využíva.

Štruktúra v ABAP DDIC definuje štruktúrovaný dátový typ, ktorý obsahuje iné dátové typy ako komponenty. Tieto komponenty môžu byť elementárne dátové typy, referenčné dátové typy, štruktúrované typy alebo tabuľkové typy.

Tabuľkové typy sú komplexné typy, ktoré popisujú interné tabuľky v ABAPe. Tabuľkové typu by si nemali čitatelia pomýliť s databázovými tabuľkami, ktoré popisujú tabuľku v databáze.

**Databázové tabuľky**

Tabuľka je objekt v ABAP DDIC, ktorý označuje databázovú tabuľku. Pokiaľ ide o jej dátový typ, databázová tabuľka je plochá štruktúra, pre ktorú je možné popri atribútoch dátového typu definovať ďalšie technické atribúty. Fyzická databázová tabuľka je vytvorená po jej aktivácií na databáze. ABAP program spracováva databázovú tabuľku ako štruktúru aj ako databázovú tabuľku. To znamená, že tabuľka môže byť použitá ako šablóna pre štruktúrované dátové objekty a tiež sprístupnená pomocou ABAP SQL. Dátové typy pre jazyk ABAP sú zobrazené v nasledujúcej tabuľke.

| **Dictionary Type** | **Meaning** | **Maximum Length n** | **ABAP Type** |
| --- | --- | --- | --- |
| DEC | Calculation/amount field | 1-31, in Tabellen 1-17 | P((n+1)/2) |
| INT1 | Single-byte integer | 3 | Internal only |
| INT2 | Two-byte integer | 5 | Internal only |
| INT4 | Four-byte integer | 10 | I |
| CURR | Currency field | 1-17 | P((n+1)/2) |
| CUKY | Currency key | 5 | C(5) |
| QUAN | Quantity | 1-17 | P((n+1)/2) |
| UNIT | Unit | 2-3 | C(n) |
| PREC | Obsolete data type | 2 | Internal only |
| FLTP | Floating point number | 16 | F(8) |
| NUMC | Numeric text | 1-255 | N(n) |
| CHAR | Character | 1-255 | C(n) |
| LCHR | Long Character | 256-max | C(n) |
| STRING | String of variable length | 1-max | STRING |
| RAWSTRING | Byte string of variable length | 1-max | XSTRING |
| DATS | Date | 8 | D |
| ACCP | Accounting period YYYYMM | 6 | N(6) |
| TIMS | Time HHMMSS | 6 | T |
| RAW | Byte string | 1-255 | X(n) |
| LRAW | Long byte string | 256-max | X(n) |
| CLNT | Client | 3 | C(3) |
| LANG | Language | internal 1, external 2 | C(1) |

**Tabuľka 1. Dátové typy jazyka ABAP**

<https://help.sap.com/doc/saphelp_nw73ehp1/7.31.19/en-US/fc/eb3138358411d1829f0000e829fbfe/content.htm?no_cache=true>

**ABAP program (Report)**

<https://www.tutorialscampus.com/tutorials/sap-abap/sap-abap-report-programming.htm>

<https://www.sapnuts.com/courses/core-abap/classical-reports/classical-report-events.html>

Program v ABAP-e je prezentácia údajov, vytvorená so špecifickým zámerom poskytovať informácie v organizovanej štruktúre. Program je vykonávaný na základe udalostí, nie riadok za riadkom.

<https://www.guru99.com/all-about-abap-report-programming.html>

Programy v ABAP-e sú používané keď je potrebné vybrať a spracovať dáta z viacerých tabuliek, keď sa program musí stiahnuť zo SAP-u do Excelu a keď je potrebné odoslať dáta prostredníctvom emailu.

**ABAP trieda (Class)** <https://www.tutorialspoint.com/sap_abap/sap_abap_classes.htm>

Trieda sa používa na špecifikovanie formy objektu a kombinuje reprezentáciu údajov a metódy manipulácie s týmito údajmi do jedného elegantného balíka. Dáta a funkcie v rámci triedy sa nazývajú členovia triedy (members).

<https://www.tutorialspoint.com/sap_abap/sap_abap_function_modules.htm>

**ABAP funkčný modul (Function Module)**

Funkčné moduly tvoria veľkú časť SAP systému, pretože SAP už roky formuje kód pomocou funkčných modulov, ktoré umožňujú opakované použitie kódu. Funkčné moduly sú podprogramy, ktoré obsahujú množinu opakovane použiteľných príkazov so vstupnými a výstupnými parametrami. SAP systém obsahuje veľký počet preddefinovaných funkčných modulov, ktoré je možné volať z ľubovoľného ABAP programu alebo triedy.

# Ciele práce

Cieľom tejto bakalárskej práce je vytvoriť funkčný systém automatizovaných testov, pomocou ktorého sa bude kontrolovať komunikácia systému SAP s vybranými externými úložiskami dát (Apache Hive, AWS Redshift, Oracle , MSSQL).

Obsah práce možno zhrnúť nasledovne:

1. Analýza vybraných úložísk a identifikácia vhodných scenárov na automatizáciu testovania,

2. Zber a vyhodnotenie požiadaviek na vytváraný systém automatického testovania,

3. Návrh a implementácia softvérového riešenia v jazyku ABAP tak, aby sa jednotlivé testy vykonávali vo vopred určenom čase a výsledky testov aby sa uchovávali na vopred definovanom mieste a v presne určenej forme,

4. Návrh a implementácia testov, tvorba testovacích scenárov,

5. Vyhodnotenie dosiahnutých výsledkov,

6. Dokumentácia riešenia vo forme UML.

# [Praktická](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Ciele_práce) časť

Predtým ako začneme vytvárať nástroj na automatizované testovanie a implementovať konkrétne testovacie scenáre, potrebujeme zistiť, ktoré scenáre pre nás budú užitočné, aby sme zabránili zbytočnému predlžovaniu vývoja automatizovaného testovacieho systému. Keď sa pozrieme na problematiku testovacích scenárov z pohľadu zákazníka, vieme bližšie určiť, ktorým smerom by sme sa mali vyberať. Väčšina zákazníkov našej firmy robí operácie nad nejakými externými úložiskami.

Pre výber vhodných testovacích scenárov sa môžeme pozrieť na scenáre, ktoré zákazníci každodenne používajú. Operácie nad externými úložiskami, môžeme zhrnúť do troch bodov a to:

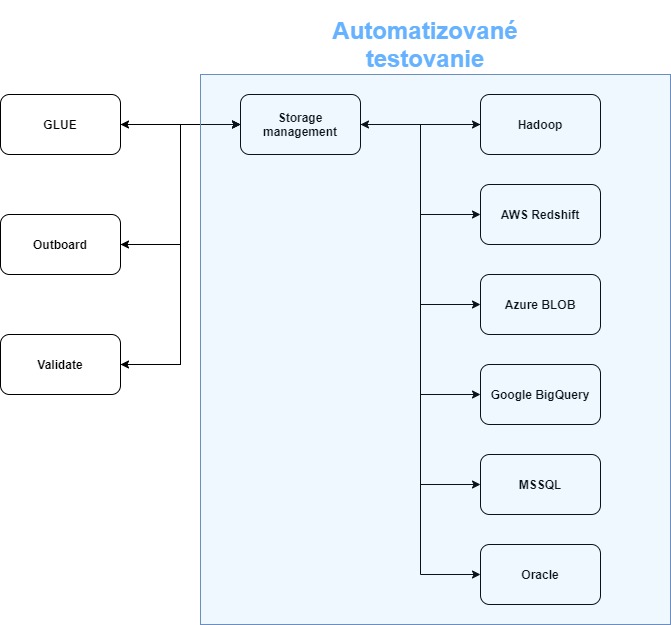
* Zapisovanie dát – vytvorenie tabuľky, prípadne súboru na externom úložisku a následný zápis dát
* Čítanie dát – načítanie dát z tabuľky, prípade súboru z externého úložiska
* Modifikácia dát – úprava existujúcich tabuliek, súborov na externom úložisku

Každý z našich zákazníkov si vyberá, ktoré externé úložiská chce používať. Či už je to podľa technológie, ktorú dané externé úložisko využíva alebo je to podľa cenníku konkrétnych externých úložísk. Naša firma aktuálne podporuje operácie nad týmito externými úložiskami: Apache Hive, Azure BLOB, Azure Data Lake, MSSQL, Oracle, Google Cloud storage, Google BigQuery, AWS Redshift.

Manuálne testovanie kombinácií všetkých vymenovaných úložísk so všetkými scenármi je časovo a taktiež technicky náročné. Z tohto dôvodu sme sa rozhodli pre vývoj automatizovaného testovacieho systému, ktorý výrazne uľahčí kontrolovanie komunikácie pre určené scenáre medzi systémom SAP a externými úložiskami.

## Analýza externých úložísk a identifikácia scenárov

Podľa scenárov zákazníka, ktoré sme vyššie identifikovali sú pre naše testovacie scenáre dôležité operácie zapisovania, čítania a modifikácie dát. Na tieto operácie sa budeme sústrediť pri vytváraní scenárov pre testovací systém.

Na obrázku č. X je znázornené používanie produktu Storage management (SM). Produkty GLUE, Outboard a Validate priamo využívajú SM pre komunikáciu s externými úložiskami, z ktorých sú niektoré znázornené na obrázku. Pri vytváraní testovacích scenárov sa budeme zameriavať na modro zvýraznenú časť obrázku, teda na komunikáciu SM s externými úložiskami.

**Obrázok 5. Komunikácia produktov s externými úložiskami**

Na základe zákazníckych scenárov zápisu, čítania a modifikácie dát si ukážeme syntax SQL príkazov nad databázou externého úložiska. SQL operácie nad databázou externého úložiska Apache Hive použijeme ako všeobecný príklad pre testovacie scenáre.

### Apache Hive <https://aws.amazon.com/big-data/what-is-hive/>

Apache Hive je distribuovaný systém skladovania údajov, odolný voči poruchám, ktorý umožňuje analýzu v rozsiahlom merítku. Dátový sklad poskytuje centrálne úložisko informácií, ktoré možno ľahko analyzovať, aby sa dali robiť rozhodnutia založené na informáciách z dát. Hive umožňuje používateľom čítať, zapisovať a spravovať petabyty dát s využitím SQL.

Hive je založený na Apache Hadoop, čo je open-source rámec (framework), ktorý sa používa na efektívne ukladanie a spracovanie veľkého množstva údajov. Výsledkom je, že Hive je úzko integrovaný s Hadoopom a je navrhnutý tak, aby dokázal rýchlo spracovať petabyty dát. Čo robí Hive unikátnym je jeho schopnosť dotazovať sa na veľké množstvá údajov pomocou Apache Tez alebo MapReduce cez rozhranie podobnému Structured Query Language (SQL).

Bol vytvorený, aby umožnil ľuďom, ktorí nie sú programátori ale sú oboznámení s SQL, pracovať s petabajtami dát pomocou rozhrania podobného SQL s názvom HiveQL. Tradičné relačné databázy sú určené na malé a stredné množiny údajov ale nespracovávajú dobre obrovské množiny údajov. Hive namiesto toho používa dávkové spracovanie, takže pracuje rýchlo vo veľmi veľkej distribuovanej databáze. Hive transformuje query HiveQL na úlohy MapReduce alebo Tez, ktoré sú spustené v rámci distribuovaného plánovania úloh Apache Hadoop, Yet Another Resource Negotiator (YARN). Dopytuje údaje uložené v distribuovanom úložisku, ako Hadoop Distributed File System (HDFS) alebo Amazon S3. Hive ukladá svoje databázové a tabuľkové metadáta do metastoru, čo je databázou alebo súborom zálohovaný sklad, ktorý umožňuje ľahkú abstrakciu a zisťovanie údajov.

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL>

Apache Hive podporuje SQL operácie na databáze, ktoré sme vybrali ako vhodné pre scenáre automatizovaného testovacieho systému. Operácie sú uvedené v nasledujúcej tabuľke.

|  |  |
| --- | --- |
| CREATE | Database, Table |
| DROP | Database, Table |
| ALTER | Database,Table |
| DESCRIBE | Database, Table |

**Tabuľka 2. Operácie nad Apache Hive**

**Create table - vytvorenie tabuľky**

**CREATE TABLE** vytvorí tabuľku s daným menom. V prípade, že tabuľka s daným menom už existuje, príkaz vyhodí chybu (Table already exists). Tejto chybe sa dá predísť doplnkom príkazu **IF NOT EXISTS**, ktorý vytvorí tabuľku len v prípade, že tabuľka s týmto menom ešte neexistuje.

Mená tabuľky a ich stĺpce sú citlivé na veľké a malé písmená. Vo verziách Hive 0.13 a neskôr, mená stĺpcov môžu obsahovať akékoľvek znaky Unicode, avšak bodka (.) a dvojbodka (:) spôsobujú chyby pri vykonávaní príkazu. Z tohto dôvodu sú tieto dva znaky zakázané.

Doplnok **COMMENT** pridá komentár k danému stĺpcu alebo celkovo k tabuľke.

Doplnok **PARTITIONED BY** umožňuje vytvoriť partíciu na tabuľke. Tabuľka môže mať jednu alebo viac partícií na stĺpcoch a pre každú kombináciu hodnôt v stĺpcoch sa vytvorí samostatný dátový adresár.

Hive taktiež podporuje ukladanie tabuľky v rôznych formátoch, napríklad:

* STORED AS TEXTFILE – uloží tabuľku ako textový súbor
* STORED AS SEQUENCEFILE – uloží tabuľku ako komprimovaný súbor
* STORED AS JSONFILE – uloží tabuľku vo formáte JSON

SQL query pre vytvorenie tabuľky na Hive s názvom tabulka1:

CREATE TABLE tabulka1

(viewTime INT,

userid BIGINT,

      page\_url STRING,

referrer\_url STRING,

      ip STRING COMMENT 'IP Address of the User')

  COMMENT 'This is the page view table'

  PARTITIONED BY(dt STRING, country STRING)

  STORED AS SEQUENCEFILE;

**Drop table - odstránenie tabuľky**

**DROP TABLE** vymaže metadáta ako aj dáta, ktoré daná tabuľka obsahuje. Ak máme nakonfigurovaný kôš dáta sa v skutočnosti presunú doňho, ale metadáta sú vymazané definitívne. V prípade, že nemáme nakonfigurovaný kôš, dáta sa vymažú.

SQL query pre vymazanie tabuľky s názvom tabulka1:

DROP TABLE tabulka1;

**Alter table – Zmena tabuľky**

**ALTER TABLE** príkaz umožňuje zmeniť štruktúru existujúcej tabuľky. Môžeme pridávať stĺpce, zmeniť vlastnosti tabuľky alebo ju premenovať.

Doplnok **RENAME TO** umožňuje meniť názov tabuľky. SQL príklad pre zmenu názvu tabuľky z „tabulka1“ na „tabulka2“:

ALTER TABLE tabulka1 RENAME TO tabulka2;

Doplnok **SET TBLPROPERTIES** môžeme použiť na pridanie vlastných metadát do tabuľky. Aktuálne vlastnosti ako posledný používateľ, ktorý vykonal posledné zmeny (last\_modified\_user) a čas poslednej zmeny na tabuľke (last\_modified\_time) sú automaticky pridávané a spravované Hiveom. Používatelia môžu pridávať ďalšie vlastnosti do príkazu. Na zistenie týchto informácií o tabuľke, môžeme použiť príkaz **DESCRIBE EXTENDED TABLE**.

SQL query pre zmenu komentáru tabuľky s názvom tabulka2:

ALTER TABLE tabulka2 SET TBLPROPERTIES (‘comment’ = novy\_komentar);

Príkazom **ALTER** taktiež môžme pridávať, premenovať, meniť alebo vymazať partície, pomocou doplnku **PARTITION.** Na pridanie partície do tabuľky môžeme použiť príkaz **ALTER TABLE ADD PARTITION**. Hodnoty partícií by mali byť v úvodzovkách iba ak sa jedná o hodnoty typu string. Časť Location musí byť adresár v ktorom sú dátové súbory uložené.

SQL query pre pridanie partície pre tabuľku tabulka1:

ALTER TABLE tabulka1 ADD PARTITION (stlpec = 'value1') location 'loc1';

Pre odstránenie partície sa používa príkaz **ALTER TABLE DROP PARTITION**. Tento príkaz odstráni dáta a metadáta pre danú partíciu. Ak máme nakonfigurovaný kôš, dáta sa presunú do koša, ale metadáta sú odstránené.

SQL príklad pre odstránenie partície na tabuľke s názvom tabulka1:

ALTER TABLE tabulka1 DROP PARTITION (dt='2008-08-08', country='us');

## Návrh automatizovaného testovacieho systému

Pri návrhu automatizovaného testovacieho systému musíme dbať na znovu-použiteľnosť jednotlivých častí systému. Keďže tento systém bude využívať viacero produktov našej firmy, prvým krokom bolo navrhnúť nástroj, ktorý bude slúžiť na spúšťanie testovacích scenárov a bude fungovať pre každý z našich produktov.

Produkt, ktorého testovacie scenáre sme implementovali v rámci tejto bakalárskej práce sa nazýva Storage management (SM). Je to balík implementácií všetkých externých úložísk a doplnkových funkcií, ktoré naša firma ponúka zákazníkom pre ukladanie dát na externých úložiskách.

**Program pre všeobecné spúšťanie testovacích scenárov**

Tento program sme nazvali /DVD/BTEST (ďalej len BTEST). Využíva tabuľku /DVD/BTEST\_SCEN (ďalej len SCEN), v ktorej budú uložené všetky typy testovacích scenárov. Jeden typ testovacieho scenára predstavuje produkt, na ktorom sa bude testovanie vykonávať.

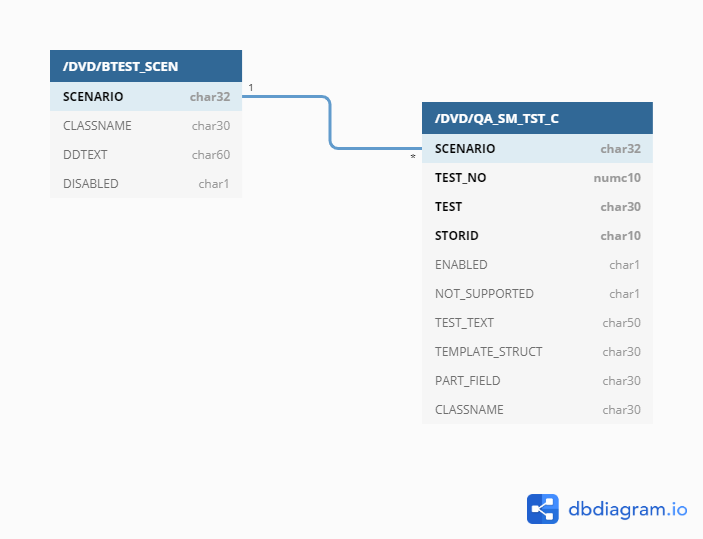
**Štruktúra tabuľky pre typy testovacích scenárov (SCEN)**

* SCENARIO (CHAR32) – kľúč, predstavuje typ testovacieho scenára
* CLASSNAME (CHAR30) – názov triedy, zodpovednej za načítanie testovacích scenárov
* DDTEXT (CHAR60) – popisný text typu testovacieho scenára
* DISABLED (CHAR1) – identifikácia platnosti typu scenára

**Konfiguračná tabuľka pre parametre testovacích scenárov**

Pre testovanie produktu SM budú parametre testovacích scenárov uložené v tabuľke s názvom /DVD/QA\_SM\_TST\_C (ďalej len TST).

**Štruktúra tabuľky TST**

* SCENARIO (CHAR32) – kľúč, predstavuje typ testovacieho scenáru, v tomto prípade je to testovanie SM na tabulárnych externých úložiskách (SM\_TAB\_STORAGE)
* TEST\_NO (NUMC10) – kľúč, číslo testu (1,2,…)
* TEST (CHAR30) – kľúč, kód testu (CREATE\_TABLE)
* STORID (CHAR10) – kľúč, interný názov premennej, ktorá reprezentuje externé úložisko ( H\_CNN\_MSH - Hive )
* ENABLED (CHAR1) – signalizuje či testovací scenár bude pustený (‘X’ alebo ‘ ’)
* NOT\_SUPPORTED (CHAR1) – signalizuje, či je daný testovací scenár aktuálne podporovaný (‘X’ alebo ‘ ’)
* TEST\_TEXT (CHAR50) – popisný text testu (Create table)
* TEMPLATE\_STRUCT (CHAR30) – názov štruktúry, ktorá sa používa ako štruktúra tabuľky pri testovaní ( /DVD/QA\_ALL\_DT)
* PART\_FIELD (CHAR30) – názov stĺpca, nad ktorým sa bude robiť partícia
* CLASSNAME (CHAR30) – názov triedy, ktorá vykonáva logiku testovacieho scenára napr. pre vytvorenie tabuľky je trieda /DVD/QA\_SM\_T\_CRT\_TAB

**Obrázok 6. Vzťah medzi tabuľkami SCEN a TST.**

**Stĺpec TEMPLATE\_STRUCT (CHAR30)**

Stĺpec TEMPLATE\_STRUCT v tabuľke konkrétnych testovacích scenárov /DVD/QA\_SM\_TST\_C predstavuje názov štruktúry, ktorá sa používa ako štruktúra tabuľky pri testovaní s názvom /DVD/QA\_ALL\_DT (ďalej len ALL\_DT). Táto štruktúra obsahuje takmer všetky možné dátové typy, ktoré je možné použiť v jazyku ABAP aby sme pri testovaní pokryli čo najväčšie spektrum hodnôt. Dátové typy, ktoré tabuľka obsahuje sme vyberali podľa tabuľky č.1.

Každý jeden typ testovacieho scenáru, ktorý bude zaznamenaný v tabuľke SCEN, bude mať svoj balík, ktorý bude obsahovať:

* triedu pre spúšťanie konkrétnych testovacích scenárov, ktoré sú popísané v časti Analýza externých úložísk a identifikácia scenárov
* abstraktnú triedu pre konkrétne testovacie scenáre
* konkrétne implementácie tried scenárov zdedených od abstraktnej triedy

Abstraktná trieda a trieda pre spúšťanie scenárov budú rozšírené o interface /DVD/BTEST\_IF\_SCENARIO s metódami INIT, TEST a CLEANUP. Metóda INIT bude mať za úlohu inicializáciu premenných, prípadne pripravenie potrebných dát a objektov pre testovanie. Metóda TEST bude vykonávať logiku testovacieho scenáru. Metóda CLEANUP bude mať za úlohu čistenie, teda odstránenie všetkých dočasných objektov, ktoré boli vytvorené pre účel testovacieho scenáru.

**Program pre ľahšiu manipuláciu s testovacími scenármi**

Ako sme už spomínali, program BTEST bude používať viacero produktov našej firmy. Preto sme vytvorili program /DVD/QA\_BTEST\_WRAPPER (ďalej len WRAPPER), ktorý ako aj podľa názvu, zaobaľuje program BTEST, poskytuje grafické rozhranie schopné spúšťať individuálne scenáre a slúži na rozdelenie typov testovacích scenárov podľa produktov ako aj na pridávanie individuálnych testovacích scenárov.

Pre pridanie testovacieho scenáru bude v programe wrapper tlačidlo add test, ktoré zobrazí parametre tabuľky pre daný typ testovacieho scenáru a po vyplnení týchto parametrov pridá záznam do tabuľky TST.

## Implementácia automatizovaného testovacieho systému

### Implementácia rozhrania pre triedy pracujúce s testovacími scenármi

Rozhranie pre testovacie scenáre sme nazvali /DVD/BTEST\_IF\_SCENARIO (ďalej len IF\_SCENARIO). Má tri metódy a to: INIT, TEST a CLEANUP. Každá z týchto metód používa ako parameter objekt s názvom IREF\_LOG typu /DVD/BTEST\_CL\_TCASES\_LOG (ďalej len TCASES\_LOG), ktorý slúži na zaznamenávanie informácií o testovacích scenároch. Objekt pre zaznamenávanie informácií obsahuje dva atribúty a to:

* SCENARIO – typ testovacieho scenáru
* T\_LOG – je štruktúra, obsahujúca dôležité informácie o stave testu, ako indikátor či bol test úspešný alebo neúspešný, dátum a čas testu ako aj detailný popis chyby v prípade zlyhania testovacieho scenára

Objekt zaznamenáva informácie o testoch pomocou metód SUCCESS a FAILURE. Metóda SUCCES je na zaznamenávanie úspešných testov a metóda FAILURE na zaznamenávanie neúspešných testov.

### Implementácia abstraktnej triedy pre testovacie scenáre

Abstraktnú triedu pre testovacie scenáre sme nazvali /DVD/QA\_SM\_CL\_TAB\_TEST\_ABS (ďalej len ABS). Trieda implementuje interface IF\_SCENARIO, ktorého metódy sú predefinované v konkrétnych triedach pre testovacie scenáre.

**Metóda CONSTRUCTOR**

Obsahuje vstupný parameter is\_test\_config, čo je štruktúra tabuľky TST. Konštruktor sa volá pri vytváraní inštancie tejto triedy s povinnosťou vyplniť tento parameter. Má za úlohu naplniť hodnoty atribútov, z ktorých väčšinu získa zo štruktúry is\_test\_config, ako napríklad názov STORID (mv\_storid), názov štruktúry tabuľky TEMPLATE\_STRUCT (mv\_temple\_struct) a stĺpec, nad ktorým chceme robiť partície PART\_FIELD (mv\_partition). Naviac vytvára vždy unikátny názov tabuľky na testovanie a to pomocou metódy GET\_UNIQUE\_TABNAME, ktorá vráti unikátny názov tabuľky podľa aktuálneho dátumu a času (mv\_sm\_tabname).

**Metóda na spracovanie výnimky**

Metódu na spracovanie výnimky sme nazvali PROCESS\_EXCEPTION. Obsahuje dva vstupné parametre, výnimku a objekt pre zaznamenávanie informácií testu IREF\_LOG. Spracuje text výnimky a zaznamená ho do objektu IREF\_LOG.

**Metóda pre získanie stĺpcov tabuľky**

Metódu pre získanie stĺpcov tabuľky sme nazvali GET\_TABLE\_FIELDS. Vstupným parametrom je názov tabuľky a výstupným parametrom sú stĺpce požadovanej tabuľky. Využíva štandardný funkčný modul systému SAP pre získanie stĺpcov tabuľky a hodnoty uloží do výstupného parametra.

**Metóda pre inicializáciu partície**

Metódu pre inicializáciu partície sme nazvali INIT\_PARTITIONING. Úlohou tejto metódy je skontrolovať atribút mt\_dd\_fields, ktorý obsahuje stĺpce testovacej tabuľky a zistiť či táto tabuľka obsahuje hodnotu PART\_FIELD vyplnenú v konfigurácií testovacieho scenáru. V prípade, že metóda tento stĺpec nenájde, vyhodí výnimku a testovací scenár je ukončený.

**Metóda pre vytvorenie tabuľky v ABAP DDIC**

Metódu pre vytvorenie tabuľky v ABAP DDIC sme nazvali CREATE\_DDIC\_TABLE. Má dva vstupné parametre a to: názov štruktúry, podľa ktorej sa vytvorí tabuľka v DDIC a meno tabuľky, pod ktorým sa tabuľka vytvorí. Pomocou SAP funkčného modulu metóda vytvorí tabuľku v DDIC.

**Metóda pre vygenerovanie dát**

Metódu pre vygenerovanie dát sme nazvali GENERATE\_DATA. Jej vstupným parametrom je názov DDIC tabuľky do ktorej sa dáta budú generovať. Ako výstupný parameter je tabuľka s vygenerovanými dátami, ktorá má štruktúru podľa názvu DDIC tabuľky.

**Metóda pre porovnanie dát**

Metóda pre porovnanie dát COMPARE\_DATA má dve tabuľky ako vstupné parametre, ktoré bude porovnávať. Ako prvé metóda porovnáva počet záznamov v tabuľke pomocou ABAP funkcie LINES. Ak sa počet záznamov v tabuľkách odlišuje, vyhodí sa výnimka. Ak je počet záznamov rovnaký, záznamy z tabuliek sa postupne po jednom čítajú a porovnávajú sa hodnoty záznamov.

**Metóda pre porovnanie stĺpcov**

Metóda pre porovnanie stĺpcov COMPARE\_FIELDS má tri vstupné parametre a to názov tabuľky iv\_tabname, názov externého úložiska iv\_storid a štruktúru na porovnanie it\_field\_example. Metódou SM GET\_DDIC\_FIELDS, na základe názvu tabuľky a názvu externého úložiska, zistíme pomocou SAP funkčného modulu štruktúru tabuľky na externom úložisku. Zistenú štruktúru potom porovnávame so štruktúrou it\_field\_example, ktorú pošleme ako vstupný parameter. V prípade, že sa štruktúry nezhodujú, metóda vyhodí výnimku.

### Implementácia triedy pre vytvorenie tabuľky na externom úložisku

Triedu pre vytvorenie tabuľky sme nazvali /DVD/QA\_SM\_T\_CRT\_TAB. Má za úlohu vytvoriť tabuľku na externom úložisku. Je zdedená od triedy ABS, čo znamená, že zdieľa jej atribúty, metódy a takisto implementuje interface IF\_SCENARIO.

**Predefinovanie metódy INIT**

INIT zavolá metódu GET\_TABLE\_FIELDS so vstupným parametrom názov tabuľky, ktorý sme získali z konfiguračných parametrov testovacieho scenáru v konštruktore. Pre pokrytie hodnôt, ktoré sa môžu vyskytovať u zákazníka, používame vo všetkých testovacích scenároch tabuľku ALL\_DT. Výstupný parameter sa uloží do atribútu pre stĺpce mt\_dd\_fields. Následne sa zavolá metóda INIT\_PARTITIONING v prípade, že je atribút mv\_partition vyplnený.

**Predefinovanie metódy TEST**

TEST najskôr zavolá metódu SM s názvom CREATE\_TABLE, so vstupnými parametrami: mv\_storid, mv\_sm\_tabname, mt\_dd\_field a mv\_partition. Táto metóda podľa vstupných parametrov pošle na úložisko query na vytvorenie tabuľky CREATE TABLE s názvom mv\_sm\_tabname, so stĺpcami určenými tabuľkou mt\_dd\_field na externom úložisku podľa mv\_storid a ak bol zadaný parameter na partície, tak podľa mv\_parition vytvorí partíciu. Pri testovaní vytvorenia tabuľky nepotrebujeme zapisovať dáta, stačí nám vytvorenie.

Nasleduje kontrola, či sa tabuľka na externom úložisku vytvorila pomocou metódy SM s názvom TABLE\_EXISTS. So vstupnými parametrami metóda skontroluje, či sa na externom úložisku mv\_storid nachádza tabuľka s názvom mv\_sm\_tabname tak, že vykoná DESCRIBE query. Pokiaľ query prejde, tabuľka existuje a pokračuje sa ďalej.

Metóda TEST taktiež testuje negatívny scenár. Znova zavolá metódu SM CREATE\_TABLE s rovnakými vstupnými parametrami s tým, že očakáva, že toto volanie zlyhá s chybou. V prípade, že je volanie metódy neúspešné, metóda odchytí výnimku a test je úspešne ukončený.

**Predefinovanie metódy CLEANUP**

CLEANUP volá metódu SM s názvom DROP\_TABLE, so vstupnými parametrami mv\_storid a mv\_sm\_tabname, ktorá vymaže tabuľku na danom externom úložisku s daným názvom pomocou query DROP TABLE. V tomto štádiu testovací scenár končí.

### Implementácia triedy pre čítanie dát z tabuľky na externom úložisku

Triedu pre čítanie dát z tabuľky sme nazvali /DVD/QA\_SM\_T\_READ\_TAB. Má za úlohu vytvoriť tabuľku na externom úložisku, zapísať do nej vygenerované dáta, prečítať tieto dáta a porovnať prečítané dáta s dátami, ktoré sa do nej zapisovali. Je zdedená od triedy ABS.

**Predefinovanie metódy INIT**

Prvým krokom je vytvoriť tabuľku v ABAP DDIC pomocou metódy CREATE\_DDID\_TABLE, ktorá bude slúžiť pre kontrolu dát. Následne sa volá metóda GENERATE\_DATA, z ktorej výstup sa uloží do lokálnej premennej lt\_dd\_data. INIT rovnako ako v triede pre vytváranie tabuľky volá metódu GET\_TABLE\_FIELDS a metódu INIT\_PARTITIONING ak je atribút mv\_partition vyplnený. Ďalším krokom je samotné vytvorenie tabuľky na externom úložisku metódou SM CREATE\_TABLE rovnako, ako v predefinovanej metóde TEST pri scenári vytvorenia tabuľky. Dáta uložené v premennej lt\_dd\_data sa uložia do vytvorenej tabuľky na externom úložisku a zavolá sa metóda SM COMMIT, ktorá zavolá príkaz commit nad databázou daného externého úložiska.

**Predefinovanie metódy TEST**

Metóda TEST najskôr prečíta dáta z tabuľky na externom úložisku pomocou metódy SM GET\_NEXT\_PACKAGE a uloží tieto dáta do lokálnej premennej lt\_sm\_data. Následne načíta dáta pomocou príkazu SELECT z tabuľky uloženej v ABAP DDIC do lokálnej premennej lt\_dd\_data. Zavolá sa metóda COMPARE\_DATA, ktorá má ako vstupné parametre lt\_sm\_data a lt\_dd\_data. Ak porovnanie dát prebehne úspešne, test je úspešne ukončený.

**Predefinovanie metódy CLEANUP**

Metóda CLEANUP pomocou SAP funkčného modulu vymaže tabuľku z DDIC. Následne pomocou metódy SM DROP\_TABLE vymaže tabuľku z externého úložiska.

### Implementácia triedy pre zmenu existujúcej tabuľky

Triedu sme nazvali /DVD/QA\_SM\_T\_ALT\_TAB. Má za úlohu zmeniť existujúcu tabuľku na externom úložisku. Je zdedená od triedy ABS.

**Predefinovanie metódy INIT**

INIT rovnako ako v triede pre vytváranie tabuľky volá metódu GET\_TABLE\_FIELDS a metódu INIT\_PARTITIONING ak je atribút mv\_partition vyplnený. Následne sa vytvorí tabuľka na externom úložisku pomocou metódy SM CREATE\_TABLE rovnako, ako v predefinovanej metóde TEST pri scenári vytvorenia tabuľky.

**Predefinovanie metódy TEST**

Metóda je rozdelená na dve časti úpravy existujúcej tabuľky. Prvá časť je vymazanie stĺpcov z tabuľky. Keďže pri každom testovacom scenári používame tabuľku ALL\_DT využijeme to v náš prospech. Vyberieme 10 stĺpcov na vymazanie z tabuľky, ktorú sme vytvorili v metóde INIT, uložíme ich do lokálnej premennej lt\_field\_del a pošleme ich do metódy SM ALTER\_TABLE spolu s názvom tabuľky a názvom externého úložiska. Zvyšné stĺpce, uložíme do lokálnej premennej lt\_field\_example, ktoré by mali tabuľke na externom úložisku ostať. Metóda ALTER\_TABLE vykoná query ALTER TABLE nad danou tabuľkou a externým úložiskom a vymaže stĺpce uložené v lt\_field\_del. Následne využijeme metódu pre porovnanie stĺpcov COMPARE\_FIELDS kde pošleme názov tabuľky, názov externého úložiska a premennú lt\_field\_example, ktorá obsahuje stĺpce, ktoré testovacej tabuľke ostali. Ak všetko prebehne úspešne, pokračujeme na druhú časť testu.

V druhej časti budeme vymazané stĺpce do tabuľky vkladať naspäť. Do lokálnej premennej lt\_field\_example uložíme stĺpce z atribútu mt\_dd\_field, ktorý obsahuje všetky stĺpce testovacej tabuľky. Z prvej časti vieme, ktoré stĺpce tabuľky sme vymazali, uložíme si ich do premennej lt\_fields\_ins. Zavoláme metódu SM ALTER TABLE, kde pošleme stĺpce na vloženie lt\_fields\_ins spolu s názvom tabuľky a názvom externého úložiska. Metóda vykoná query ALTER TABLE nad danou tabuľkou a externým úložiskom a vloží do nej stĺpce uložené v lt\_field\_ins. Následne využijeme metódu pre porovnanie stĺpcov COMPARE\_FIELDS kde pošleme názov tabuľky, názov externého úložiska a premennú lt\_fieds\_example, ktorá obsahuje už všetky stĺpce tabuľky. Ak porovnanie dát prebehne úspešne, test je úspešne ukončený.

**Predefinovanie metódy CLEANUP**

CLEANUP pomocou metódy SM DROP\_TABLE vymaže testovaciu tabuľku na danom externom úložisku.

### Implementácia triedy pre spúšťanie testovacích scenárov SM

Triedu pre spúšťanie testovacích scenárov produktu SM sme nazvali /DVD/QA\_SM\_CL\_TAB\_STORAGE (ďalej len STORAGE), ktorá implementuje interface IF\_SCENARIO. Táto trieda má svoj názov spolu s typom testovacieho scenára, v tomto prípade je to SM, v zázname tabuľky SCEN v stĺpci CLASSNAME, ktorej inštanciu vytvára program BTEST. STORAGE má dva atribúty:

* mt\_test\_config – štandardná tabuľka typu tabuľky TST
* ms\_test\_config – štrukrúra typu TST

**Predefinovanie metódy INIT**

INIT za pomoci štandardnej SAP funkcionality zistí svoj názov, vyberie záznam z tabuľky SCEN, kde zistí hodnotu stĺpca SCENARIO a uloží si ju do lokálnej premennej lv\_scenario. Následne vyberie všetky záznamy z tabuľky TST podľa hodnôt SCENARIO = lv\_scenario, ENABLED = ‘X’ a NOT\_SUPPORTED < > ‘X’ do atribútu mt\_test\_config.

**Predefinovanie metódy TEST**

Metóda TEST obsahuje LOOP cyklus nad tabuľkou mt\_test\_config, ktorý pri každej iterácií uloží záznam do štruktúry ms\_test\_config. Vnútri cyklu sa vytvorí inštancia triedy uložená v štruktúre ms\_test\_config podľa stĺpca CLASSNAME s názvom lo\_test, ktorá zodpovedá konkrétnemu testovaciemu scenár. Pri vytváraní inštancie sa volá CONSTRUCTOR abstraktnej triedy TEST\_ABS kde vstupný parameter is\_test\_config = ms\_test\_config. Následne inštancia lo\_test zavolá svoje predefinované metódy INIT A TEST a CLEANUP. V prípade, že v žiadnej z týchto metód nenastane chyba, test prebehol úspešne. V prípade zlyhania testu, metóda odchytí výnimku a zaznamená chybu do objektu IREF\_LOG a cyklus pokračuje na ďalší záznam z tabuľky mt\_test\_config.

Metóda CLEANUP neobsahuje žiadnu implementáciu, pretože o čistenie dát a záznamov sa starajú konkrétne implementácie testovacích scenárov.

**Implementácia programu pre spúšťanie testovacích scenárov**

Program BTEST slúži sa spúšťanie testovacích scenárov. Obsahuje parametre so\_scen, p\_debug, p\_notify, p\_subj a so\_email. Parameter so\_scen môže obsahovať typ testovacieho scenáru, čiže produkt, na ktorom sa bude testovať. So\_scen môže obsahovať viacero hodnôt. P\_debug slúži sa debugovanie testov tak, že po spustení programu zastaví vykonávanie a otvorí debuger. P\_notify je začiarkovacie políčko, ktoré signalizuje programu BTEST, či bude výsledky testov posielať emailom. P\_subj je popis testovania, prednastavený na hodnotu „Automatic test results“. A nakoniec parameter so\_email slúži na vyplnenie emailovej adresy, na ktorú sa výsledky testov pošlú. So\_email takisto môže obsahovať viacero hodnôt.

Po vyplnení týchto parametrov BTEST spustí hlavnú logiku programu. Vytvorí inštanciu objektu pre zaznamenávanie informácií lo\_log, typu TCASES\_LOG. Z tabuľky SCEN pomocou príkazu SELECT vytiahne záznamy do lokálnej premennej lt\_scen (tabuľka typu SCEN) s podmienkou, WHERE scenario IN so\_scen, čo znamená, že sa vytiahnu všetky záznamy, kde sa SCENARIO zhoduje z hodnotami v so\_scen. Každý záznam z tabuľky v stĺpci CLASSNAME obsahuje názov triedy, ktorá sa stará o spúšťanie testovacích scenárov napr. trieda STORAGE. BTEST robí LOOP cyklus na tabuľke lt\_scen, kde pri každej iterácií vytvorí inštanciu triedy podľa stĺpca CLASSNAME a zavolá metódy INIT, TEST a CLEANUP. Po ukončení LOOP cyklu, BTEST odošle výsledky testov na emailové adresy zadané v parametri so\_email.

**Implementácia programu pre ľahšiu manipuláciu s testovacími scenármi**

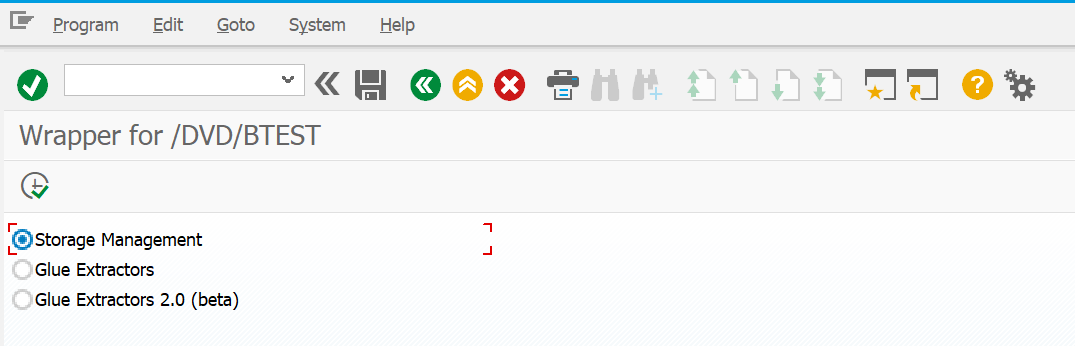
Program WRAPPER poskytuje grafické rozhranie na vyberanie testovacích scenárov. Obsahuje parametre p\_sm, p\_glext a p\_ext2, podľa ktorých sa vyplní premenná gv\_scen napríklad pre p\_sm bude mať gv\_scen hodnotu „SM\_TAB\_STORAGE“. Podľa výberu jedného z parametrov pomocou začiarkovacích políčok (check box) WRAPPER zobrazí zoznam testovacích scenárov, ktoré sú uložené v konfiguračnej tabuľke. Pre produkt SM je to tabuľka TST.

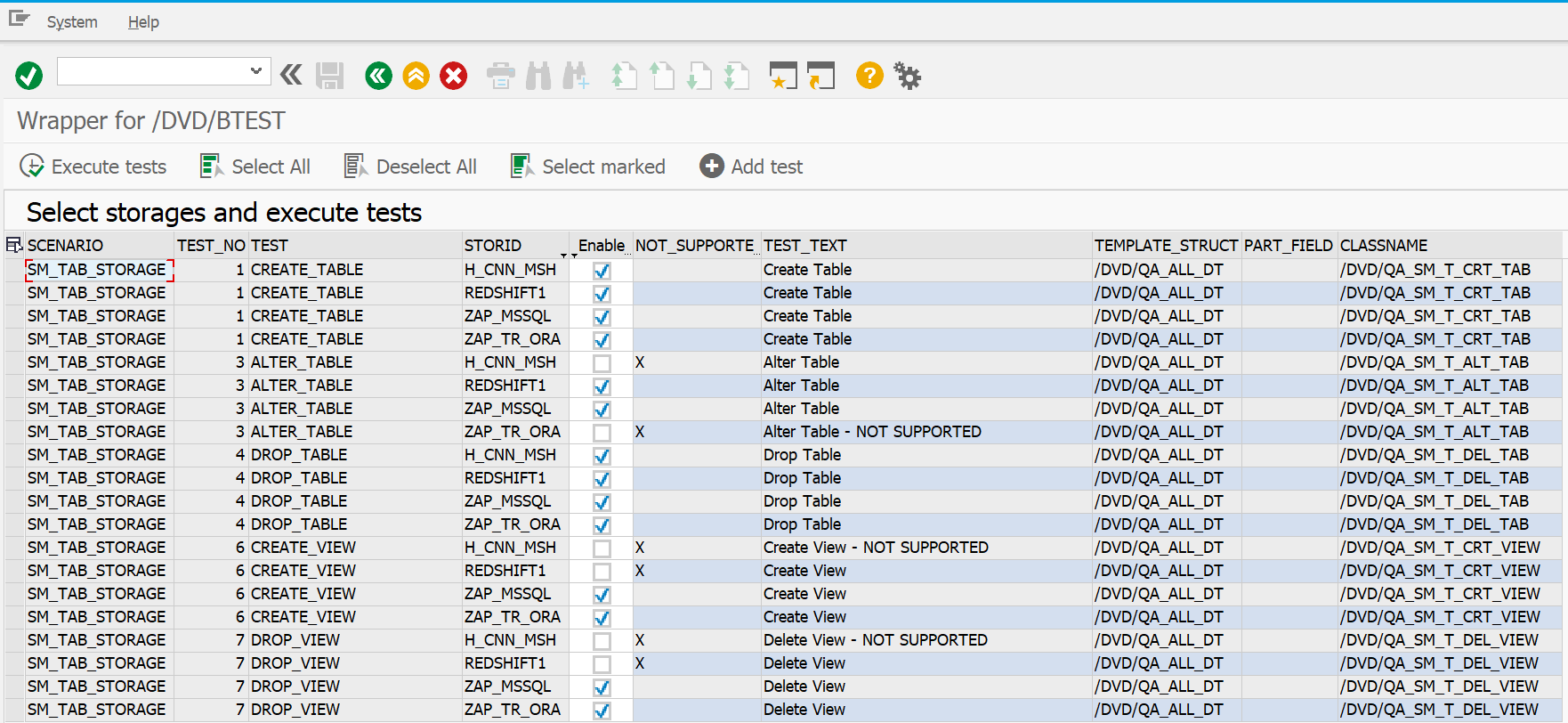
Po výbere testovacích scenárov zo zoznamu, WRAPPER stlačením tlačidla Execute tests najskôr uloží zmeny urobené v zozname do konfiguračnej tabuľky. Následne zavolá a spustí vykonávanie programu BTEST s vyplnenými parametrami so\_scen = gv\_scen, p\_debug = ‘ ’, p\_notify = ‘X’, p\_subj = ‘BTEST executed from QA wrapper’. Parameter so\_email je prednastavený na hodnotu skupinového emailu, kde sú všetci z našej firmy koho sa udržiavanie produktu SM týka.

## Proces spustenia automatizovaného testovacieho systému

**Jednorazové spustenie**

Spustenie automatizovaného testovacieho systému začína spustením programu WRAPPER, kde si môžeme vybrať z podporovaných produktov na testovanie prostredníctvom začiarkovacích tlačidiel (radio buttons). Ako môžeme vidieť na obrázku č. X, program aktuálne podporuje výber testovacích scenárov pre produkt SM a produkt Glue.



Stlačením tlačidla execute sa dostaneme k zoznamu testovacích scenárov uložených pre daný produkt, v tomto prípade sú to scenáre pre SM, čo môžeme vidieť na obrázku číslo X.

V zozname si môžeme pomocou začiarkovacích políčok (check box) v stĺpci Enabled vybrať testovacie scenáre, ktoré chceme aby boli spustené. Tlačidlo Select All označí pre všetky záznamy stĺpec Enabled ako začiarknutý. Tlačidlo Deselect All naopak označí pre všetky záznamy stĺpec Enabled ako nezačiarknutý. GUI tohto programu nám taktiež povoľuje označiť myšou viacero scenárov naraz a po stlačení tlačidla Select marked sa pre označené scenáre začiarkne stĺpec Enabled.

Následným stlačením tlačidla Execute tests sa vyplnia parametre programu BTEST a zobrazí sa jeho grafické rozhranie, čo môžeme vidieť na obrázku X.

**Konvencie pomenovaní pre firmu Datavard s.r.o. v SAP systémoch**

Väčšina firiem pracujúcich v SAP systémoch má určité konvencie pre pomenovania objektov. Naša firma má od spoločnosti SAP pre pomenovania dátových typov, tabuliek, štruktúr atď. rezervovanú predponu “/DVD/”. Z tohto dôvodu všetky objekty začínajú touto predponou.

V rámci vzájomnej dohody medzi zamestnancami našej firmy máme takisto určité pravidlá pre pomenovávanie lokálnych premenných, atribútov atď. Prvé písmeno signalizuje dosah alebo úlohu premennej. Lokálna premenná má prvé písmeno ( „l“ ) a globálna ( „g“ ). Pri parametroch metód majú vstupné parametre prvé písmeno „i“ a výstupné „e“. Druhé písmeno signalizuje dátový typ premennej. Platí:

* lv\_... (variable) – premenné jednoduchého typu (STRING, CHAR, INT4)
* gs\_... (structure) – premenné štruktúrovaného typu
* it\_... (table) – premenné tabuľkového typu
* eo\_... (object) – premenné pre inštanciu triedy

## Vyhodnotenie dosiahnutých výsledkov

# [Metodika práce a metódy skúmania](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Metodika_práce)

# [Výsledky práce a diskusia](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Výsledky_práce)

## Výsledky práce

Podrobný popis postupov podľa metodiky riešenia.

## Diskusia

Výsledky, ktoré boli dosiahnuté riešením.

# [Záver](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Záver)

# [Zoznam použitej literatúry](file:///C:\Users\DonChameleón\Desktop\Pokyny_pre_vypracovanie_ZP.docx#Zoznam_použitej_literatúry)

[1] RAYGUN, “The SLDC”, Dave Swersky [Online]. Available:

<https://raygun.com/blog/software-development-life-cycle/>. [Cit. ]

# [Zoznam príloh](C:\\Users\\DonChameleón\\Desktop\\Pokyny_pre_vypracovanie_ZP.docx" \l "Prílohy)

**Príloha A** Názov

**Príloha B** Názov

# 

# Prílohy

## Príloha A: Názov prílohy

Každá ďalšia príloha začína na novej strane.

## Príloha B: Obsah DVD

Priložené DVD obsahuje:

* Práca v elektronickej podobe (formát PDF)