

**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 RIADENIA A INFORMATIKY**

**BAKALÁRSKA PRÁCA**

**INFORMATIKA**

**FRANTIŠEK CABADAJ**

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

Žilinská univerzita v Žiline

Fakulta Riadenia a Informatiky

Školiace pracovisko: Katedra Informatiky

Žilina, 2020

**Pod'akovanie**

Chcem poďakovať vedúcemu práce doc. Ing. Marekovi Kvetovi, PhD., svojim kolegom zo spoločnosti Datavard a najmä svojmu manažérovi zo spoločnosti Datavard, Ing. Róbertovi Adamcovi za odbornú pomoc, usmernenie a cenné rady, vďaka ktorým som úspešne napísal túto prácu.

ŽILINSKÁ UNIVERZITA V ŽILINE, FAKULTA RIADENIA A INFORMATIKY.

ZADANIE TÉMY BAKALÁRSKEJ PRÁCE.

Študijný program: Informatika

Meno a priezvisko

František Cabadař

Osobné číslo

556425

**Názov práce v slovenskom aj anglickom jazyku**

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

Development of automatized testing system for checking the communication between SAP and external storages

**Zadanie úlohy, ciele, pokyny pre vypracovanie**

(Ak je málo miesta, použite opačnú stranu)

**Cieľ bakalárskej práce:**

Cieľom 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:**

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.

Meno a pracovisko vedúceho BP: doc. Ing. Marek Kvet, PhD., KI, ŽU

Meno a pracovisko tutora BP:

---

vedúci katedry  
(dátum a podpis)

Zadanie zaregistrované dňa 06. 12. 2019 pod číslom 832/2019 podpis \_\_\_\_\_

**ABSTRAKT V ŠTÁTNOM JAZYKU**

CABADAJ, František: *Vývoj automatizovaného testovacieho systému pre kontrolu komunikácie medzi SAP a externými úložiskami*. [Bakalárska práca] – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky; Katedra Informatiky. – Vedúci: doc. Ing. Marek Kvet, PhD. – Stupeň odbornej kvalifikácie: Bakalár. – Žilina: FRI ŽU v Žiline, 2020. Počet strán: 48

Bakalárska práca sa venuje vývoju automatizovaného testovacieho systému v jazyku ABAP, ktorý kontroluje komunikáciu medzi systémom SAP a externými úložiskami. V teoretickej časti sa práca zaoberá životným cyklom informačného systému so zameraním na testovanie. Porovnáva funkčné a nefunkčné testovanie a informuje o typoch testovania. Krátko rozoberá objekty v jazyku ABAP. V praktická časť sa zaoberá identifikáciou testovacích scenárov, na základe zákazníckych scenárov a analýzou SQL operácií nad databázou externého úložiska Apache Hive. Venuje sa návrhu automatizovaného testovacieho systému ako aj jeho implementácií, ktoré sú obohatené o UML a iné diagramy pre lepšie pochopenie fungovania systému. Ďalej spusteniu individuálnych testovacích scenárov ako aj nastaveniu automatického spúšťania testovacích scenárov a vyhodnoteniu výsledkov testovacích scenárov pomocou e-mailu, čo bolo cieľom práce.

**Kľúčové slová:** SAP, testovanie, automatizácia, externé úložiská, ABAP, SQL, Apache Hive

**ABSTRAKT V CUDZOM JAZYKU**

CABADAJ, František: Development of automatized testing system for checking the communication between SAP and external storages [Bachelor thesis] – University of Žilina. Faculty of management science and informatics; Department of Informatics. – Supervisor: doc. Ing. Marek Kvet, PhD. – Qualification level: Bachelor. – Žilina: FRI ŽU in Žilina, 2020. Total number of pages: 48

The bachelor thesis focuses on the development of automatized testing system in ABAP language, which checks the communication between SAP and external storages. In the theoretical part of the thesis deals with the software development life cycle with focus on testing. It compares functional and non-functional testing and informs about the types of testing. Short part is dedicated to ABAP objects. The practical part focuses on identifying testing scenarios, based on scenarios which customers use. It analyzes SQL operations on Apache Hive storage database. It focuses on design of the automated testing system as well as its implementation which are enriched with UML and other diagrams for better understanding how the system works. Furthermore, it deals with launching individual testing scenarios as well as setting up automated launch of testing scenarios and evaluating the results from testing scenarios using e-mail, which was the goal of this thesis.

**Key words:** SAP, testing, automation, external storages, ABAP, SQL, Apache Hive

## Obsah

<b>Zoznam obrázkov .....</b>	<b>1</b>
<b>Zoznam tabuliek .....</b>	<b>2</b>
<b>Zoznam skratiek .....</b>	<b>3</b>
<b>Úvod .....</b>	<b>4</b>
<b>1 Teoretické východiská .....</b>	<b>5</b>
1.1 Životný cyklus informačného systému .....	5
1.2 Testovanie .....	9
1.2.1 Funkčné (functional) testovanie .....	9
1.2.2 Ne-funkčné (Non-functional) testovanie .....	11
1.2.3 Automatizované testovanie .....	12
1.2.4 Manuálne testovanie .....	14
1.3 SAP.....	15
1.3.1 Enterprise resource planning (ERP) .....	16
1.3.2 História spoločnosti SAP.....	17
1.3.3 Jazyk ABAP .....	17
<b>2 Praktická časť .....</b>	<b>21</b>
2.1 Analýza externých úložísk a identifikácia scenárov .....	21
2.1.1 Apache Hive .....	22
2.2 Návrh automatizovaného testovacieho systému.....	25
2.2.1 Konvencie pomenovaní spoločnosti Datavard v SAP systémoch.....	25
2.2.2 Program pre všeobecné spúšťanie testovacích scenárov .....	26
2.2.3 Rozhranie pre triedy pracujúce s testovacími scenármi .....	28
2.2.4 Program pre ľahšiu manipuláciu s testovacími scenármi .....	29
2.2.5 Zhrnutie návrhu .....	29
2.3 Implementácia automatizovaného testovacieho systému.....	30
2.3.1 Implementácia rozhrania pre triedy pracujúce s testovacími scenármi...30	
2.3.2 Implementácia abstraktnej triedy pre testovacie scenáre .....	31
2.3.3 Implementácia triedy pre vytvorenie tabuľky na externom úložisku.....	33
2.3.4 Implementácia triedy pre čítanie dát z tabuľky na externom úložisku....	34
2.3.5 Implementácia triedy pre zmenu existujúcej tabuľky .....	34
2.3.6 Implementácia triedy pre volanie testovacích scenárov SM .....	35
2.3.7 Implementácia programu pre spúšťanie testovacích scenárov .....	36
2.3.8 Implementácia programu pre ľahšiu manipuláciu s testovacími scenármi	37
2.4 Proces spustenia automatizovaného testovacieho systému .....	37
2.4.1 Jednorazové spustenie .....	37

2.4.2	Nastavenie pravidelného automatického spustenia testov.....	40
2.4.3	Ďalšie možnosti vývoja .....	42
<b>Záver</b>	.....	<b>43</b>
<b>Zoznam príloh</b>	.....	<b>44</b>
<b>Bibliografia</b>	.....	<b>45</b>
<b>Prílohy</b>	.....	<b>47</b>
Príloha A:	USB Flash disk.....	48



## Zoznam obrázkov

Obrázok 1 - Životný cyklus informačného systému .....	6
Obrázok 2 - Typy softvérového testovania [2] .....	9
Obrázok 3 - Proces automatizovaného testovania [3].....	13
Obrázok 4 - 3 vrstvomá architektúra.....	17
Obrázok 5 - Komunikácia produktov s externými úložiskami .....	22
Obrázok 6 - Vzťah medzi tabuľkami SCEN a TST.....	28
Obrázok 7 - Návrh automatizovaného testovacieho systému .....	30
Obrázok 8 - Program WRAPPER, výber produktu.....	38
Obrázok 9 - Program WRAPPER, zoznam testovacích scenárov .....	38
Obrázok 10 - pridávanie testov.....	39
Obrázok 11- Program BTEST, grafické rozhranie .....	39
Obrázok 12 - Úspešné výsledky testovacích scenárov.....	40
Obrázok 13 - Neúspešné výsledky testovacích scenárov .....	40
Obrázok 14 - Nastavenia automatického spustenia testov .....	41

## Zoznam tabuliek

Tabuľka 1- Dátové typy jazyka ABAP [15] .....	19
Tabuľka 2 - SQL Operácie nad Apache Hive [14] .....	23

## **Zoznam skratiek**

UML	Unified Modeling Language
SDLC	Software Development Lifecycle
SME	Subject Matter Experts
TOGAF	The Open Group Architecture Framework
ARA	Application Release Automation
GUI	Graphical user interface
SAP	System applications and Products
CRM	Customer Relationship Management
PI	Process Integration skratkou
ERP	Enterprise resource planning
ECC	ERP Central Component
HCM	Human Capital Management
PP	Production Planning
MM	Materials Management
PS	Project System
SD	Sales and Distribution
PM	Plant Maintenance
FICO	Finance and controlling
QM	Quality Management
DDIC	Data Dictionary
SQL	Structured Query Language
SM	Storage management

## Úvod

Cieľom tejto bakalárskej práce je identifikovať testovacie scenáre zápisu, čítania a modifikácie dát pomocou zákazníckych scenárov, navrhnúť automatizovaný testovací systém v jazyku ABAP pre kontrolu komunikácie medzi systémom SAP a externými úložiskami a následne tento systém implementovať.

Testovací systém sa bude spúšťať každý deň aby mal tím spoločnosti Datavard, ktorý spravuje komunikáciu medzi systémom SAP a externými úložiskami každodenný prehľad o fungovaní tejto komunikácie.

# 1 Teoretické východiská

## 1.1 Životný cyklus informačného systému

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.[1]

Či už je to dokument, diagram alebo funkčný softvér, artefakt<sup>[1]</sup> 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). [1]

artefakt<sup>[1]</sup> – je jedným z mnohých druhov vedľajších produktov vyrobených počas vývoja softvéru ako napr. Unified Modeling Language (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. [1]

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. [1]

„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. [1]

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. [1]

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 [1]

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. [1]

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. [1]



Obrázok 1 - Životný cyklus informačného systému

## 1. Plánovanie

Fáza plánovania môže zahŕňať:

- Plánovanie kapacít
- Plánovanie projektu
- Odhad nákladov
- Pridelovanie 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]

## 2. 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]

## 3. 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. [1]

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. [1]

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

#### 4. Vývoj softvéru

Táto fáza vyprodukuje vyvíjaný softvér. V závislosti od metodológie môže byť táto fáza vykonávaná v časovo obmedzených šprintoch (Agilná metodika). 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]

#### 5. 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: [1]

- 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]

#### 6. 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 Application 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]

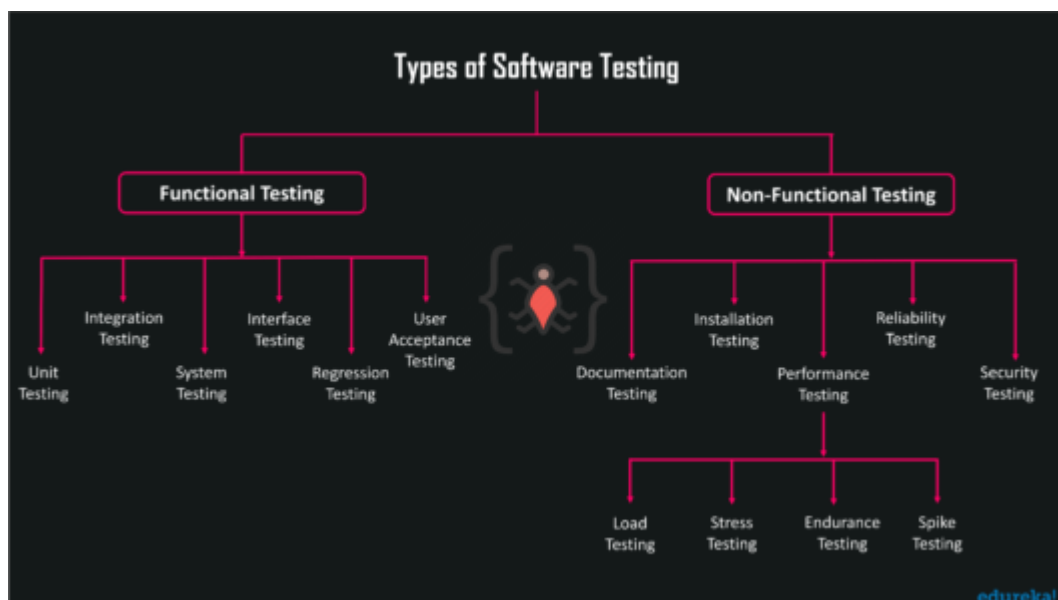


## 7. Údržba

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 produkcii sa musia nahlásiť a spracovať, čo často vedie k tomu, že sa vyžaduje dodatočná práca 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. [1]

### 1.2 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. [2]



Obrázok 2 - Typy softvérového testovania [2]

Typy softvérového testovania sa delia do dvoch základných skupín a to funkčné (functional) a ne-funkčné (non-functional) testovanie. [2]

#### 1.2.1 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. [2]

### **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 testovania, kde sú testované individuálne unity. Hlavným cieľom je overiť, či každý komponent softvéru pracuje tak ako by mal. [2]

#### **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. [2]

#### **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. [2]

#### **Interface testovanie**

Je to úroveň softvérového testovania, ktorá overuje komunikáciu jedného softvéru s druhým. [2]

#### **Regresné (Regression) testovanie**

Overuje či zmeny v kóde nemajú dopad na existujúcu funkcionálnu produkt. [2]

#### **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. [2]

### **1.2.2 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. [2]

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. [2]

#### **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. [2]

##### **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. [2]

##### **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. [2]

- Zátťažové (load) testovanie - vyhodnocuje správanie systému pri postupnom zvyšujúcom sa pracovnom zaťažení.
- Stress (dôrazové) testovanie – vyhodnocuje správanie systému na hranici alebo za hranicou zvláduteľného pracovného zaťaženia.
- Endurance (vytrvalostné) testovanie - vyhodnocuje správanie systému pri veľkej dlho trvajúcej pracovnej záťaži.
- Spike testovanie – vyhodnocuje správanie systému pri náhlom zvýšení pracovnej záťaže. [2]

### 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. [2]

#### 1.2.3 Automatizované testovanie

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. [3]

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. [3]

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. [3]

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 informovaní o tom, že zmena kódu ktorú odovzdali niečo pokazila. Ďalšie dôvody pre automatizované testovanie sú: [3]

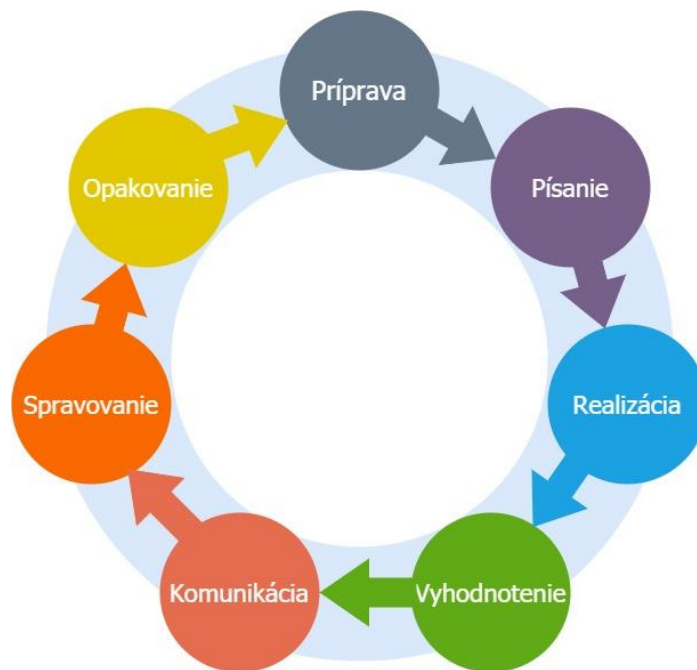
- Overenie novších verzií softvéru
- Oslobodenie testerov od triviálnych scenárov aby sa mohli sústrediť na komplexnejšie testovacie scenáre
- Väčšie pokrytie testov
- Znovu-použitelnosť
- Rýchlejšie nasadenie softvéru
- Šetrí čas
- Dodáva rýchlu spätnú väzbu pre vývojárov ohľadom chýb v softvéri [3]

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. [3]

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. [3]

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. [3]

Proces automatizovaného testovania sa dá zhrnúť do týchto krokov:



Obrázok 3 - Proces automatizovaného testovania [3]

### Príprava

Najskôr je potrebné pripraviť sa a pochopiť ciele funkčného testovania, aké testovacie dáta sú potrebné a čo treba overiť. [3]

**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. [3]

**Realizácia**

Realizácia testov by mala byť spoľahlivá. Odporúča sa spustiť každý test aspoň trikrát za sebou predtým ako sa skontroluje kód. [3]

**Vyhodnotenie**

Overenie, či automatizované testy robia to, čo sa od nich čaká. Pomocou manuálnych testov sa overí, či sa testy správajú podľa požiadaviek. [3]

**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. [3]

**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. [3]

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. [3]

**1.2.4 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 presiahol čas strávený vývojom GUI. V takomto prípade testuje scenáre testovací tím alebo samotný vývojár, ktorý GUI vyvinul. [3]

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 +ticelý testovaný scenár – určité scenáre vyžadujú schopnosti programátorov
- Kvôli opakujúcej sa povahe, testerí môžu pri manuálnom vykonávaní zanedbať alebo prehliadnuť niektoré dôležité kroky čo môže viesť k nekonzistentnému stavu testu. [3]

### 1.3 SAP

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. [4]

SAP je skratka pre System applications 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. [4]

Po veľkom úspechu R/3, vytvoril SAP AG viac a viac niche<sup>[2]</sup> (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. [4]

niche<sup>[2]</sup> – zameraný konkrétnu oblasť

Všetky podnikové procesy sú vykonávané v jednom SAP systéme a zdieľajú spoločné informácie. [4]

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. [5]

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. [5]

### **1.3.1 Enterprise resource planning (ERP)**

SAP SE je jedným z najväčších dodávateľov softvéru pre (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 SAP-u do druhého bez potreby zbytočného zadávania údajov a pomáha vymáhať finančné, procesné a právne kontroly. Umožňuje tiež efektívne využívanie zdrojov, vrátane pracovnej sily, strojov a výrobných kapacít. [5]

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. [5]

SAP ERP má rozdielne hlavné moduly, ktoré sú rozdelené do funkčných a technických modulov, z ktorých každý má submoduly. [5]

Funkčné moduly SAP systému zahŕňajú:

- Human Capital Management (SAP HCM)
- Production Planning (SAP PP)
- Materials Management (SAP MM)
- Project System (SAP PS)
- Sales and Distribution (SAP SD)
- Plant Maintenance (SAP PM)
- Finance and controlling (SAP FICO)
- Quality Management (SAP QM) [5]



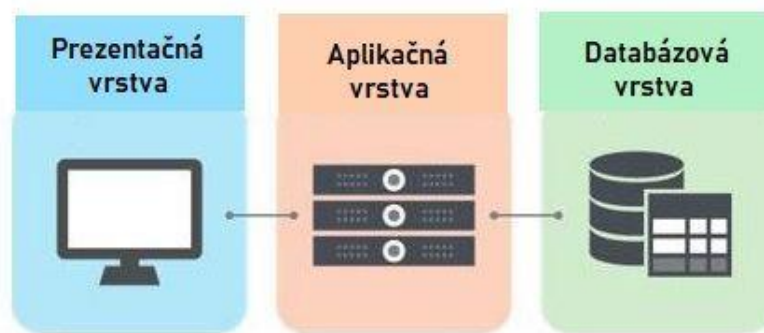
### 1.3.2 História spoločnosti SAP

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. [5]

V roku 1973, SAP vydal R/1, systém finančného účtovníctva. R/1 bežal na IBM serveroch a Disk Operating System (DOS). Mal jednostupňovú architektúru, v ktorej boli prezentačná, aplikačná a databázová vrstva na jednej platforme. [5]

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. [5]

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. [5]



Obrázok 4 - 3 vrstvová architektúra

### 1.3.3 Jazyk ABAP

Advanced Business Application Programming (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. [6]

#### ABAP Data Dictionary

ABAP Data 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. [6]

### **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 „\_“. [6]

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. [6]

Dátové elementy sú elementárne dátové typy alebo referencie so sémantickými atribútmi. Atribúty dátového elementu sú definované buď priamo alebo pomocou domény. [6]

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. [6]

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. [6]

Tabuľkové typy sú komplexné typy, ktoré popisujú interné tabuľky v ABAP-e. Tabuľkové typu by si nemali čitatelia pomýliť s databázovými tabuľkami, ktoré popisujú tabuľku v databáze. [6]

**Tabuľka 1- Dátové typy jazyka ABAP [15]**

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)

### 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 aktivácii 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. [6]

Dátové typy pre jazyk ABAP sú zobrazené v nasledujúcej tabuľke.

### **ABAP program (Report)**

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. Môže obsahovať grafické rozhranie. Podporuje takzvanú select option, čo je príkaz, ktorý sa používa, aby užívatelia mohli zadávať rozsah hodnôt alebo viacero jednotlivých hodnôt. [7] [8] [9]

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. [10]

### **ABAP trieda (Class)**

Trieda sa používa na špecifikovanie formy objektu a kombinuje reprezentáciu údajov a metódy manipulácie s týmito údajmi do jedného elegantného balíka. Atribútu dáta a funkcie v rámci triedy sa nazývajú členovia triedy (members). [11]

### **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. [12]

## 2 Praktická 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, prípadne súboru z externého úložiska
- Modifikácia dát – úprava existujúcich tabuliek, súborov na externom úložisku

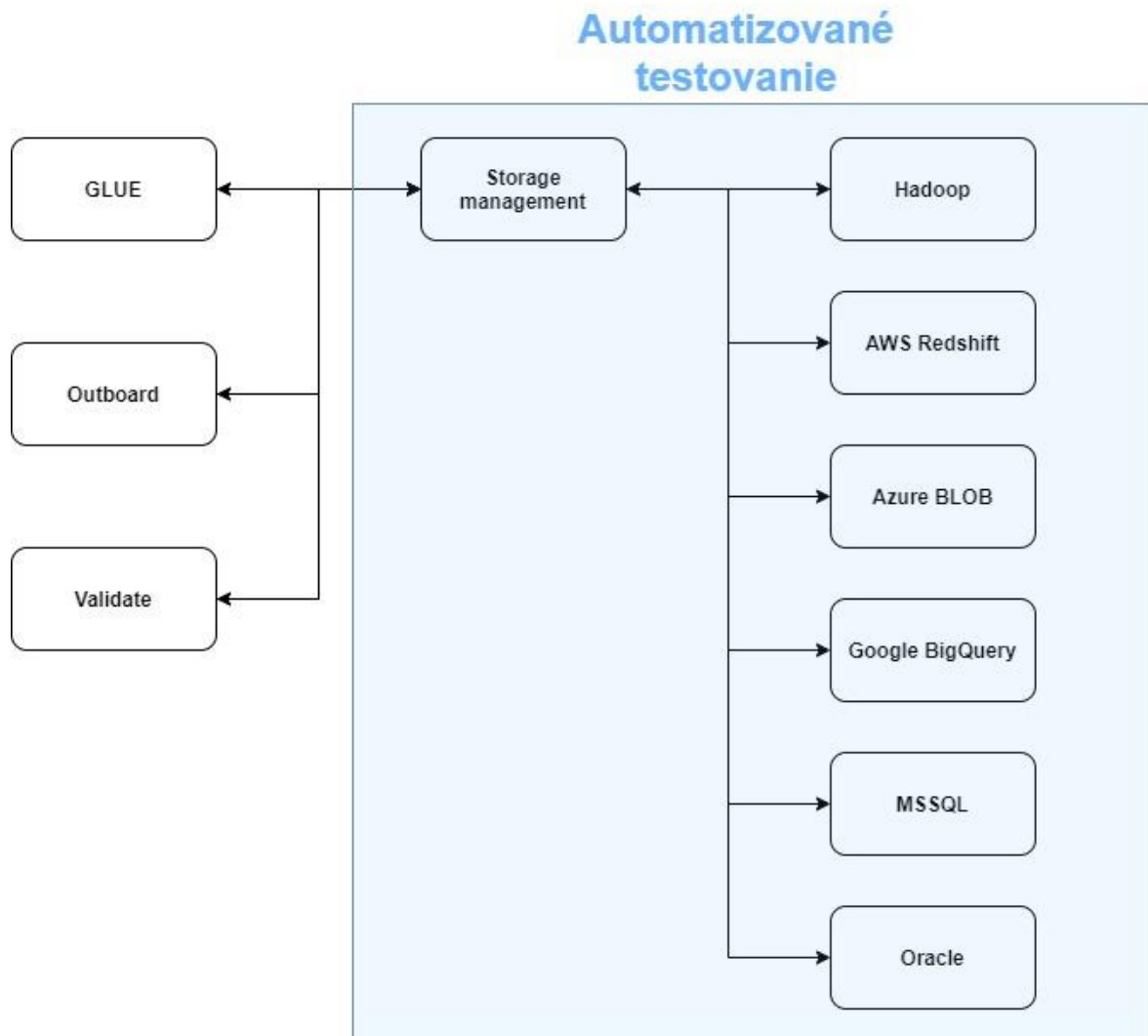
Každý z našich zákazníkov si vyberá, ktoré externé úložiská chce používať. Či už je to podľa technológie, ktorú dané externé úložisko využíva alebo je to podľa cenníku konkrétnych externých úložísk. Naša firma aktuálne podporuje operácie nad týmito externými úložiskami: Apache Hive, Azure BLOB, Azure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

### 2.1 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. 5 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.

### 2.1.1 Apache Hive

Apache Hive je distribuovaný systém skladovania údajov, odolný voči poruchám, ktorý umožňuje analýzu v rozsiahlom merítku. Dátový sklad poskytuje centrálnu 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. [13]

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). [13]

Apache Hive podporuje SQL operácie na databáze, ktoré sme vybrali ako vhodné pre scenáre automatizovaného testovacieho systému. Operácie sú uvedené v nasledujúcej tabuľke. [13]

**Tabuľka 2 - SQL Operácie nad Apache Hive [14]**

CREATE	Database, Table
DROP	Database, Table
ALTER	Database, Table
DESCRIBE	Database, Table

### **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 dvoj-bodka (:) 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 môžeme 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');
```

## 2.2 Návrh automatizovaného testovacieho systému

Pri návrhu automatizovaného testovacieho systému musíme dbať na znovu-použitelnosť jednotlivých častí systému. Keďže tento systém bude využívať viacero produktov našej firmy, prvým krokom bolo navrhnuť 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 je produkt SM. Je to balík implementácií všetkých externých úložísk a doplnkových funkcií, ktoré naša firma ponúka zákazníkom pre zapisovanie, čítanie a modifikáciu dát na externých úložiskách.

### 2.2.1 Konvencie pomenovaní spoločnosti Datavard 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 takmer všetky objekty začínajú touto predponou.

V rámci vzájomnej dohody medzi zamestnancami našej firmy máme takisto určité pravidlá pre pomenovávanie premenných. 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

Pre atribúty metód prvé písmeno signalizuje člena triedy (member). Z tohto dôvodu, všetky atribúty začínajú písmenom „m“. Druhé písmeno signalizuje dátový typ, rovnako ako pri premenných.

Parametre pre ABAP programy začínajú písmenom „p“. Takéto parametre môžu byť typu vstupného poľa, check box-u alebo radio button group. Select option v rámci programu začína písmenami „so“.

### **2.2.2 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.

Každý typ testovacieho scenára bude mať v zázname tabuľky SCEN a stĺpci CLASSNAME názov triedy, ktorá bude slúžiť na volanie konkrétnych implementácií testovacích scenárov.

#### **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

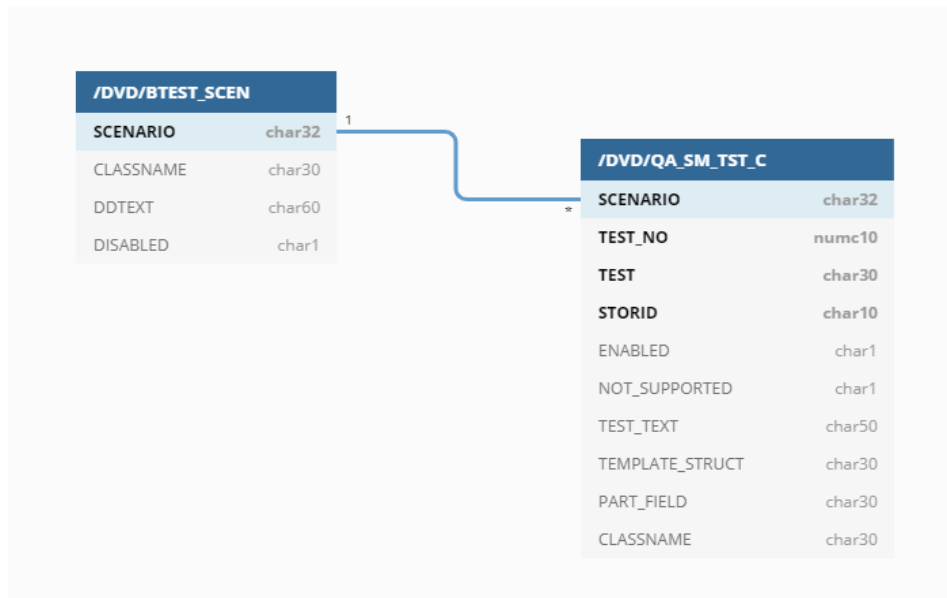
- DISABLED (CHAR1) – identifikácia platnosti typu scenára

### **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 pre vytvorenie tabuľky



Obrázok 6 - Vzťah medzi tabuľkami SCEN a TST

### Stĺpec TEMPLATE\_STRUCT (CHAR30)

Stĺpec TEMPLATE\_STRUCT v konfiguračnej tabuľke testovacích scenárov TST 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.

### Abstraktná trieda pre testovacie scenáre

Táto trieda bude abstraktnou triedou pre už konkrétne implementácie testovacích scenárov. Triedy pre konkrétne implementácie testovacích scenárov budú zdedené od tejto abstraktnej triedy, čo znamená, že budú zdieľať jej metódy a atribúty.

### Trieda pre volanie konkrétnych testovacích scenárov

Trieda načíta testovacie scenáre z konfiguračnej tabuľky TST a zavolá konkrétne implementácie testovacích scenárov.

#### 2.2.3 Rozhranie pre triedy pracujúce s testovacími scenármi

Abstraktná trieda a trieda pre volanie 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.

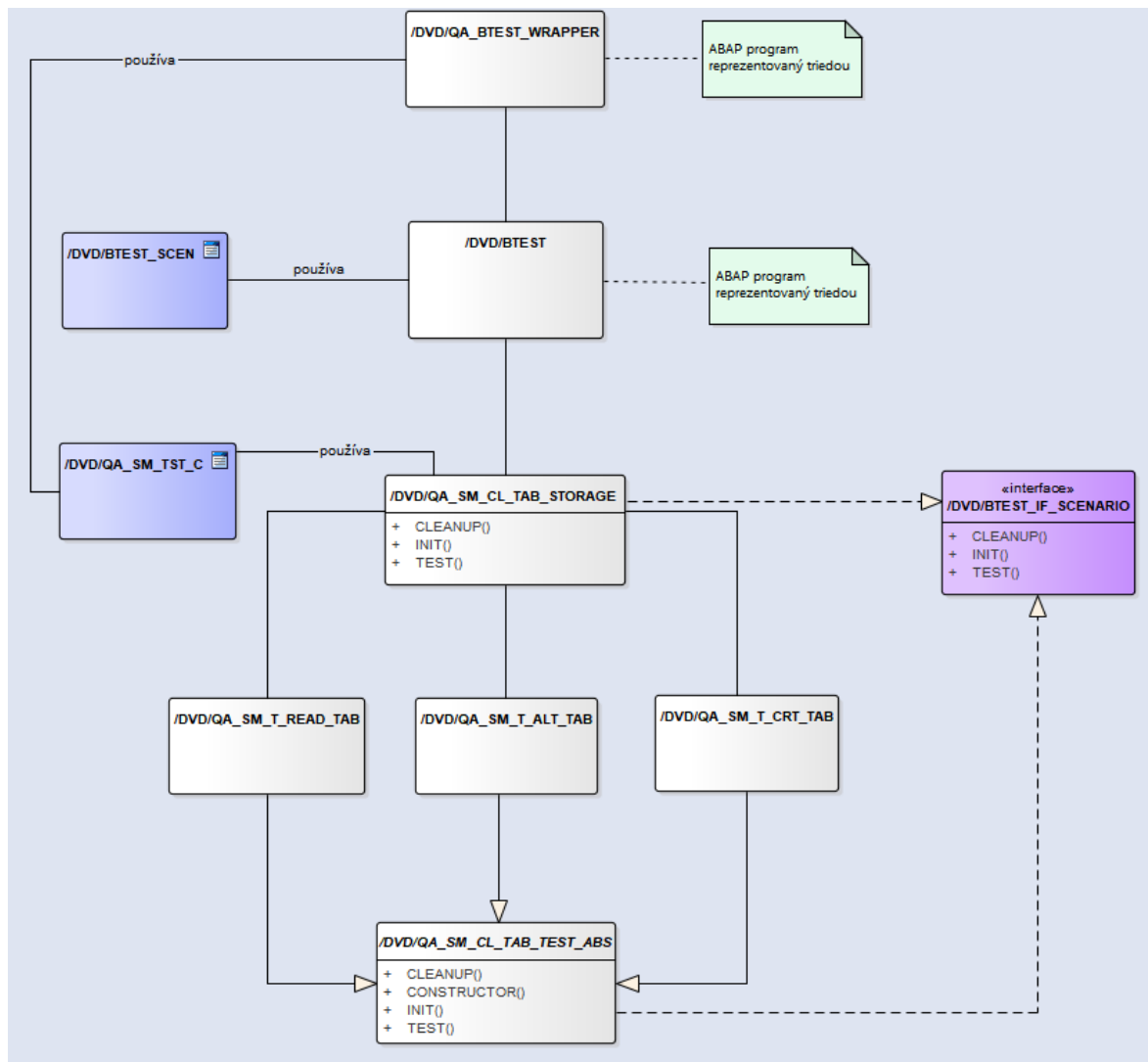
#### **2.2.4 Program pre ľahšiu manipuláciu s testovacími scenármi**

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, zaobaluje 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. Po výbere testovacích scenárov, WRAPPER spustí program BTEST s vyplnenými parametrami.

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.

#### **2.2.5 Zhrnutie návrhu**

Programom WRAPPER si vyberieme testovacie scenáre, ktoré chceme spúšťať. WRAPPER spustí program BTEST s vyplnenými parametrami. Program BTEST zavolá metódy triedy pre volanie konkrétnych testovacích scenárov. Táto trieda si vyberie záznamy z tabuľky TST a postupne vykonáva testovacie scenáre. Pre znázornenie návrhu sme vytvorili diagram tried, ktorý môžeme vidieť na obrázku č. 7.



Obrázok 7 - Návrh automatizovaného testovacieho systému

## 2.3 Implementácia automatizovaného testovacieho systému

### 2.3.1 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 SUCCESS je na zaznamenávanie úspešných testov a metóda FAILURE na zaznamenávanie neúspešných testov.

### 2.3.2 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). Navyše 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 konfigurácii 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.



### 2.3.3 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\_partition 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 odchyť 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.

### 2.3.4 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.

### 2.3.5 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 vkladat' 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.

#### **2.3.6 Implementácia triedy pre volanie testovacích scenárov SM**

Triedu pre volanie 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

případe 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 – štruktúra 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 iterácii 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 scenáru. 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 odchyť 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.

#### **2.3.7 Implementácia programu pre spúšťanie testovacích scenárov**

Program BTEST slúži na 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. P\_debug slúži na debugovanie testov tak, že po spustení programu zastaví vykonávanie a otvorí debugger. P\_notify je začiarkovacie políčko, ktoré signalizuje programu BTEST, či bude výsledky testov posielat' 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 iterácii 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.

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

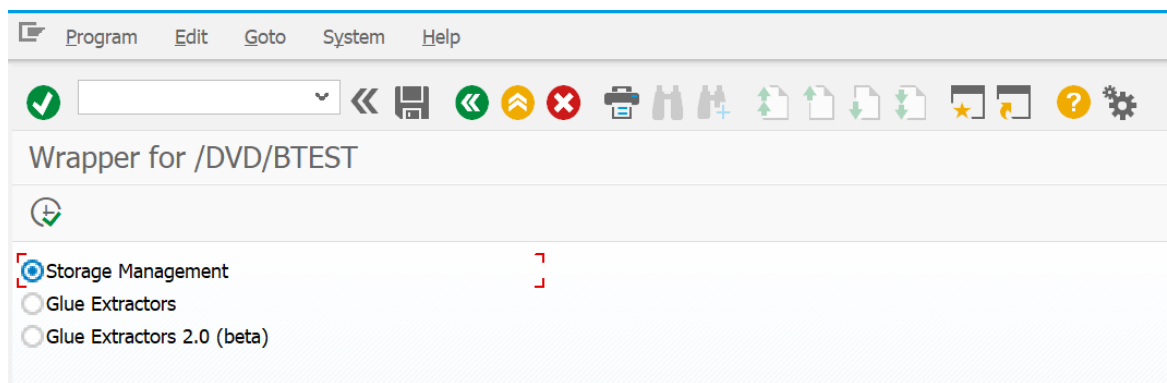
Program WRAPPER poskytuje grafické rozhranie na vyberanie testovacích scenárov. Obsahuje parametre p\_sm, p\_glex2 a p\_ext2, podľa ktorých sa vyplní premenná gv\_scen napríklad pre p\_sm bude mať gv\_scen hodnotu „SM\_TAB\_STORAGE“. Podľa výberu jedného z parametrov pomocou začiarkovacích políčk (check 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.

## **2.4 Proces spustenia automatizovaného testovacieho systému**

### **2.4.1 Jednorazové spustenie**

Spustenie automatizovaného testovacieho systému začína spustením programu WRAPPER, kde si môžeme vybrať z podporovaných produktov na testovanie prostredníctvom radio button tlačidiel. Ako môžeme vidieť na obrázku č. 8, program aktuálne podporuje výber testovacích scenárov pre produkt SM a produkt Glue.



Obrázok 8 - Program WRAPPER, výber produktu

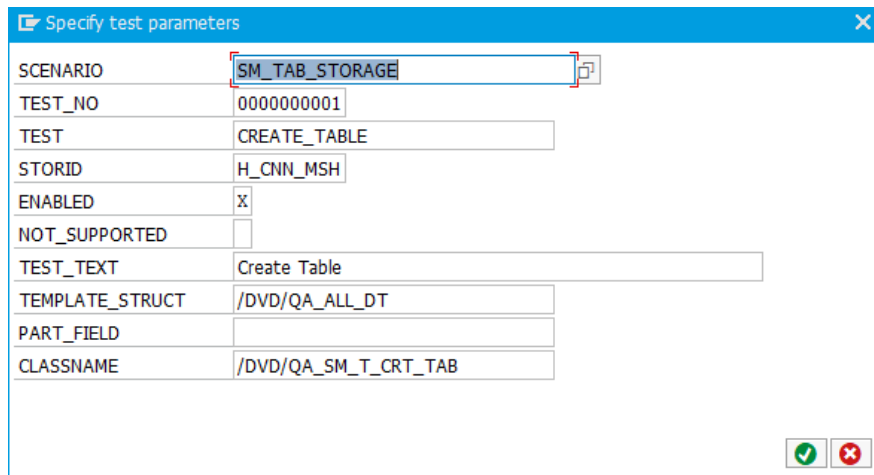
Stlačením tlačidla execute sa dostaneme k zoznamu testovacích scenárov uložených pre daný produkt, v tomto prípade sú to scenáre pre SM, čo môžeme vidieť na obrázku číslo 9.

SCENARIO	TEST_NO	TEST	STORID	ENABLED	NOT_SUPPORTED	TEST_TEXT	TEMPLATE_STRUCT	PART_FIE	CLASSNAME
SM_TAB_STORAGE	1	CREATE_TABLE	H_CNN_MSH	<input checked="" type="checkbox"/>		Create Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_CRT_TAB
SM_TAB_STORAGE	1	CREATE_TABLE	REDSHIFT1	<input checked="" type="checkbox"/>		Create Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_CRT_TAB
SM_TAB_STORAGE	1	CREATE_TABLE	ZAP_MSSQL	<input checked="" type="checkbox"/>		Create Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_CRT_TAB
SM_TAB_STORAGE	1	CREATE_TABLE	ZAP_TR_ORA	<input checked="" type="checkbox"/>		Create Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_CRT_TAB
SM_TAB_STORAGE	3	ALTER_TABLE	H_CNN_MSH	<input checked="" type="checkbox"/>		Alter Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_ALT_TAB
SM_TAB_STORAGE	3	ALTER_TABLE	REDSHIFT1	<input checked="" type="checkbox"/>		Alter Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_ALT_TAB
SM_TAB_STORAGE	3	ALTER_TABLE	ZAP_MSSQL	<input checked="" type="checkbox"/>		Alter Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_ALT_TAB
SM_TAB_STORAGE	3	ALTER_TABLE	ZAP_TR_ORA	<input checked="" type="checkbox"/>		Alter Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_ALT_TAB
SM_TAB_STORAGE	21	READ_DATA_FROM_TABLE	H_CNN_MSH	<input checked="" type="checkbox"/>		Read Data From Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_READ_TAB
SM_TAB_STORAGE	21	READ_DATA_FROM_TABLE	REDSHIFT1	<input checked="" type="checkbox"/>		Read Data From Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_READ_TAB
SM_TAB_STORAGE	21	READ_DATA_FROM_TABLE	ZAP_MSSQL	<input checked="" type="checkbox"/>		Read Data From Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_READ_TAB
SM_TAB_STORAGE	21	READ_DATA_FROM_TABLE	ZAP_TR_ORA	<input checked="" type="checkbox"/>		Read Data From Table	/DVD/QA_ALL_DT		/DVD/QA_SM_T_READ_TAB

Obrázok 9 - Program WRAPPER, zoznam testovacích scenárov

V zozname si môžeme pomocou začiarokovacích 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 začiarčne stĺpec ENABLED.

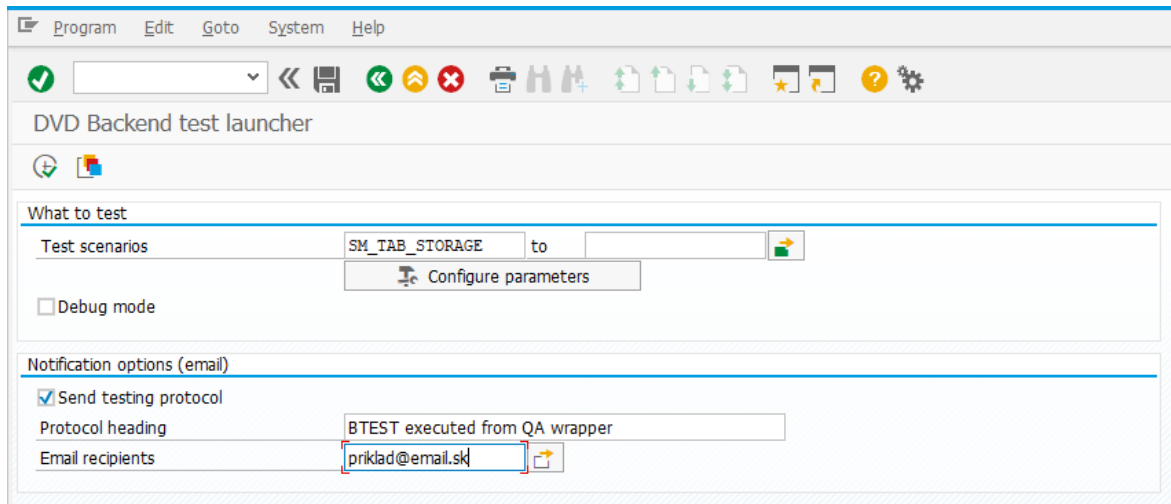
Pre pridávanie testov slúži tlačidlo Add test. Tlačidlo zobrazí parametre tabuľky TST a po ich vyplnení uloží záznam do tejto tabuľky. Grafické rozhranie pre pridávanie testov môžeme vidieť na obrázku č. 10.



SCENARIO	SM_TAB_STORAGE
TEST_NO	0000000001
TEST	CREATE_TABLE
STORID	H_CNN_MSH
ENABLED	X
NOT_SUPPORTED	
TEST_TEXT	Create Table
TEMPLATE_STRUCT	/DVD/QA_ALL_DT
PART_FIELD	
CLASSNAME	/DVD/QA_SM_T_CRT_TAB

Obrázok 10 - pridávanie testov

Následným stlačením tlačidla Execute tests sa vyplnia parametre programu BTEST a zobrazí sa jeho grafické rozhranie, čo môžeme vidieť na obrázku č. 11.



Program Edit Goto System Help

DVD Backend test launcher

What to test

Test scenarios SM\_TAB\_STORAGE to [ ]

Configure parameters

Debug mode

Notification options (email)

☒ Send testing protocol

Protocol heading BTEST executed from QA wrapper

Email recipients priklad@email.sk

Obrázok 11- Program BTEST, grafické rozhranie

V tejto fáze grafického rozhrania BTEST-u si môžeme vyplniť políčko na email, kde po vykonaní testovacích scenárov BTEST pošle výsledky. Po stlačení tlačidla Execute sa spustí vykonávanie testov. Úspešné výsledky testovacích scenárov vyzerajú, ako je ukázané na obrázku 12.

Datavard BTEST protocol		
BTEST executed from QA wrapper		
NSD @ 03.05.2020 15:56:01		
Summary		
Total tests		11
Successful tests		11
Failed tests		0
Not supported tests		0
Details		
Scenario	Test case	Exec. Time
SM TAB testing	Create Table (H_CNN_MSH)	15:50:13
	Create Table (REDSHIFT1)	15:50:13
	Create Table (ZAP_MSSQL)	15:51:17
	Create Table (ZAP_TR_ORA)	15:51:35
	Alter Table (H_CNN_MSH)	15:52:01
	Alter Table (REDSHIFT1)	15:52:01
	Alter Table (ZAP_MSSQL)	15:52:01
	Alter Table (ZAP_TR_ORA)	15:52:01
	Read data from storage (H_CNN_MSH)	15:54:22
	Read data from storage (ZAP_MSSQL)	15:55:49
	Read data from storage (ZAP_TR_ORA)	15:56:01

Obrázok 12 - Úspešné výsledky testovacích scenárov

V období, kedy sme spracovávali tieto údaje sa vyskytla chyba v implementácii pre čítanie dát z externého úložiska AWS Redshift. Ako môžeme vidieť na obrázku č. 13. test pre čítanie zlyhal. Testovací systém zaznamenal kde chyba nastala, takže pri následnej oprave tejto chyby sme vedeli kde ju hľadať.

Scenario	Test case	Exec. Time
SM TAB testing	Read Data From Table (REDSHIFT1)	16:01:25
	Error in test initialization	16:01:25
	Exception without text: /DVD/CX_SM_CONN_ERROR	16:01:25
	Failed to open JDBC session.	16:01:25
	Connection to java service failed	16:01:25
	Java service connection failed. Exec engine: REDSHIFT, Destination: JCO2_SERVER	16:01:25
	Java error message: 'ession failed: JDBC driver com.amazon.redshift.jdbc41.Driver not initialized correctly'	16:01:25

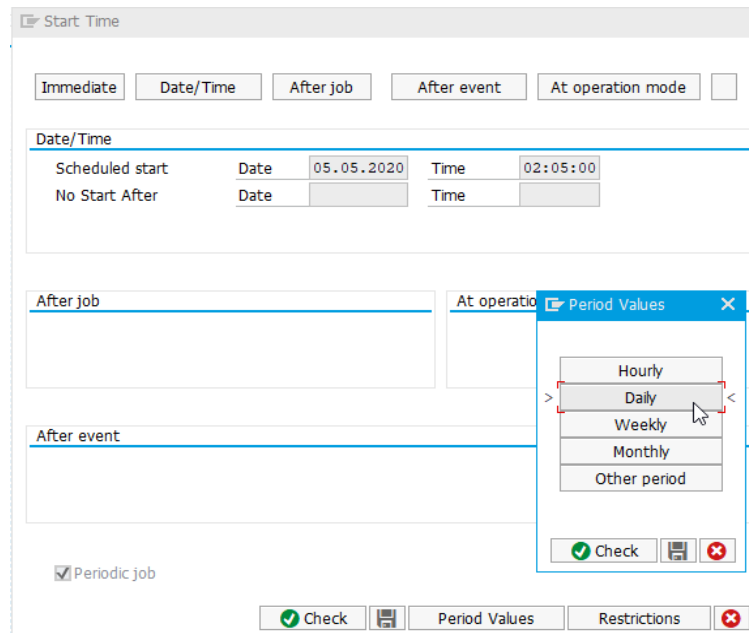
Obrázok 13 - Neúspešné výsledky testovacích scenárov

#### 2.4.2 Nastavenie pravidelného automatického spustenia testov

Aby sme nemuseli každý deň púšťať tento testovací systém manuálne, pomocou SAP funkcionality nastavíme automatické pravidelné spúšťanie. Toto spúšťanie sa bude vykonávať v noci, pretože testovanie zaťažuje SAP systém a tým obmedzuje zamestnancov našej firmy.



Pri kroku ako je ukázané na obrázku č. 10 môžeme pomocou tlačidla F9 môžeme spustiť takzvaný background job. Pomocou grafického rozhrania SAP systému si nastavíme Scheduled start. V políčku Date a Time nastavíme dátum a čas prvého spustenia. Pomocou začiarokovacieho políčka Periodic job povieme SAP-u, že tento program sa bude pravidelne opakovať. Kliknutím na tlačidlo Period Values si nastavíme pomocou stlačenia tlačidla Daily, každodenné spustenie tohto programu o čase zadanom v políčku Time. Ukážku nastavenia môžeme vidieť na obrázku č. 14.



**Obrázok 14 - Nastavenia automatického spustenia testov**

Môže sa stať, že v priebehu pracovného dňa sa robila implementácia, napríklad do nejakého externého úložiska. Po implementovaní riešenia si môže developer chcieť vyskúšať testovacie scenáre aby zistil, či bola implementácia riešenia úspešná. Keďže potrebuje testy len pre jedno externé úložisko, ostatné testy vypne a tieto zmeny sa uložia do konfiguračnej tabuľky TST. Ak developer nevráti výber testovacích scenárov do pôvodného stavu, automatické spustenie testov vykoná len tie testy, ktoré developer nechal zapnuté.

Aby sme sa takýmto prípadom vyhli, urobili sme jednoduchý program, ktorý predtým ako sa spustí automatizované testovanie v noci, upraví záznamy konfiguračnej tabuľky tak, že zapne všetky testovacie scenáre. Týmto riešením si môžeme byť istý, že sa každý deň spustia všetky testovacie scenáre, ktoré konfiguračná tabuľka obsahuje.

### **2.4.3 Ďalšie možnosti vývoja**

Scenáre automatizovaného testovacieho systému je možné priebežne rozširovať podľa externých úložísk, ktoré naša firma zákazníkom ponúka. Tým, že testovací systém je navrhnutý všeobecne pre akýkoľvek produkt spoločnosti Datavard, je možné testovacie scenáre nekonečne rozširovať o nové. Tím produktu GLUE začal vyvíjať testovacie scenáre hneď ako bol automatizovaný systém funkčný. Do budúcnosti sa plánujú vyvíjať testovacie scenáre aj pre produkty Outboard a Validate.

## Záver

Cieľom tejto bakalárskej práce bolo identifikovať testovacie scenáre zápisu, čítania a modifikácie dát pomocou zákazníckych scenárov, navrhnúť automatizovaný testovací systém v jazyku ABAP pre kontrolu komunikácie medzi systémom SAP a externými úložiskami a následne tento systém implementovať.

V teoretickej časti sme sa dozvedeli ako funguje životný cyklus informačných systémov a bližšie sme analyzovali fázu testovania. Zistili sme rozdiely medzi funkčným a nefunkčným testovaním a priblížili sme si úlohy typov funkčného a nefunkčného softvérového testovania. Dozvedeli sme sa histórií spoločnosti SAP, ako sa vyvíjali ich produkty a zistili sme čo je to ERP a ako funguje. V krátkosti sme si povedali o jazyku ABAP a objektoch, ktoré ABAP využíva.

V úvode praktickej časti sme identifikovali na základe zákazníckych scenárov, relevantné testovacie scenáre pre automatizovaný testovací systém. Apache Hive sme použili ako príklad pre ukážku SQL operácií nad databázou externého úložiska. Ďalej sme navrhovali samotný testovací systém, ukázali sme si konvencie pre pomenovania objektov firmy Datavard, navrhli sme ako bude automatizovaný systém fungovať a aké objekty bude používať. Na záver návrhu sme vytvorili UML diagram tried pre lepšie pochopenie fungovania systému. Následne sme popísali implementáciu programov, rozhraní, tried a metód používaných v testovacom systéme.

V posledných kapitolách sme sa venovali spusteniu testovacích scenárov a nastaveniu automatického každodenného spustenia scenárov, ako aj zaisteniu toho, že sa každý deň spustia všetky scenáre bez ohľadu na manipuláciu so systémom v priebehu pracovného dňa. Na záver sme v krátkosti popísali budúcnosť vývoja testovacích scenárov pre iné produkty našej firmy.

## **Zoznam príloh**

**Príloha A** USB Flash disk

## Bibliografia

- [1] D. Swersky, „RAYGUN,“ 31 Mája 2018. [Online]. Available: <https://raygun.com/blog/software-development-life-cycle/>.
- [2] V. M. R, „edureka!,“ 22 Máj 2019. [Online]. Available: <https://www.edureka.co/blog/functional-testing-vs-non-functional-testing/>.
- [3] J. Calantionio, „TEST GUILD,“ 2017. [Online]. Available: <https://testguild.com/automation-testing/>.
- [4] Krishna, „GURU99,“ 2015. [Online]. Available: <https://www.guru99.com/what-is-sap-definition-of-sap-erp-software.html>.
- [5] M. Rouse, „SearchSAP,“ [Online]. Available: [https://searchsap.techtarget.com/definition/SAP?fbclid=IwAR22o4XxOhTOdOar3gePgFADmEt3Wb\\_kbEV4HqH7u1ItcwA3OhA8kKCV2I](https://searchsap.techtarget.com/definition/SAP?fbclid=IwAR22o4XxOhTOdOar3gePgFADmEt3Wb_kbEV4HqH7u1ItcwA3OhA8kKCV2I). [Cit. 15 Apríl 2020].
- [6] SAP, „help.sap,“ SAP, [Online]. Available: [https://help.sap.com/doc/abapdocu\\_latest\\_index\\_htm/latest/en-US/index.htm](https://help.sap.com/doc/abapdocu_latest_index_htm/latest/en-US/index.htm). [Cit. 18 Apríl 2020].
- [7] TutorialsCampus, „TutorialCampus,“ 2018. [Online]. Available: <https://www.tutorialscampus.com/tutorials/sap-abap/sap-abap-report-programming.htm>. [Cit. 20 Apríl 2020].
- [8] A. K. Reddy, „Sapnuts,“ Sapnuts, 10 August 2017. [Online]. Available: <https://www.sapnuts.com/courses/core-abap/classical-reports/classical-report-events.html>. [Cit. 20 Apríl 2020].
- [9] A. K. Reddy, „Sapnuts,“ Sapnuts, 10 August 2017. [Online]. Available: <https://www.sapnuts.com/courses/core-abap/selection-screen/select-options-abap.html>. [Cit. 20 Apríl 2020].
- [10] Krishna, „GURU99,“ [Online]. Available: <https://www.guru99.com/all-about-abap-report-programming.html>. [Cit. 20 Apríl 2020].

- [11] Tutorialspoint, „tutorialspoint,“ [Online]. Available: [https://www.tutorialspoint.com/sap\\_abap/sap\\_abap\\_classes.htm](https://www.tutorialspoint.com/sap_abap/sap_abap_classes.htm). [Cit. 20 Apríl 2020].
- [12] Tutorialspoint, „tutorialspoint,“ [Online]. Available: [https://www.tutorialspoint.com/sap\\_abap/sap\\_abap\\_function\\_modules.htm](https://www.tutorialspoint.com/sap_abap/sap_abap_function_modules.htm). [Cit. 20 Apríl 2020].
- [13] AWS, „AWS Amazon,“ Amazon, [Online]. Available: <https://aws.amazon.com/big-data/what-is-hive/>. [Cit. Apríl 2020].
- [14] Apache Hive, „Apache Hive,“ [Online]. Available: <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL>. [Cit. 1 Máj 2020].
- [15] SAP, „help.SAP,“ SAP, [Online]. Available: [https://help.sap.com/doc/saphelp\\_nw73ehp1/7.31.19/en-US/fc/eb3138358411d1829f0000e829fbfe/content.htm?no\\_cache=true](https://help.sap.com/doc/saphelp_nw73ehp1/7.31.19/en-US/fc/eb3138358411d1829f0000e829fbfe/content.htm?no_cache=true). [Cit. 18 Apríl 2020].

## **Prílohy**

## **Príloha A: USB Flash disk**

Priložený USB Flash disk obsahuje:

- Text bakalárskej práce vo formáte PDF
- Enterprise Architect projekt, obsahujúci UML diagram
- Obrázky použité v bakalárskej práci