SLOVENSKÁ POĽNOHOSPODÁRSKA UNIVERZITA

V NITRE

TECHNICKÁ FAKULTA

2136291

ON-LINE TESTOVANIE V PREDMETE „PROGRAMOVANIE“

2011 **Stanislav Pohuba, Bc.**

SLOVENSKÁ POĽNOHOSPODÁRSKA UNIVERZITA

V NITRE

Dr. h. c. prof. Ing. Peter Bielik, PhD.

TECHNICKÁ FAKULTA

prof. Ing. Zdenko Tkáč, PhD.

ON-LINE TESTOVANIE V PREDMETE „PROGRAMOVANIE“

Diplomová práca

Študijný program:	Informačná a automatizačná technika v kvalite produkcie
Študijný odbor:	2386800 Kvalita produkcie
Školiace pracovisko:	Katedra elektrotechniky, automatizácie a informatiky
Školiteľ:	doc. Ing. Zuzana Palková, PhD.

Nitra, 2011 **Stanislav Pohuba, Bc.**

Čestné vyhlásenie

Podpísaný Bc. Stanislav Pohuba vyhlasujem, že som záverečnú prácu na tému *„On-line testovanie v predmete „Programovanie““* vypracoval samostatne pod odborným vedením doc. Ing. Zuzany Palkovej, PhD. a s použitím uvedenej literatúry.

Som si vedomý zákonných dôsledkov v prípade, ak uvedené údaje nie sú pravdivé.

V Nitre 17. Apríla 2011

Bc. Stanislav Pohuba

Poďakovanie

Chcel by som sa poďakovať mojej školiteľke, *doc. Ing. Zuzane Palkovej, PhD.* za jej orientačné vedenie, cenné odborné rady a pripomienky, návrhy a námety, ktoré mi poskytla pri vypracovaní diplomovej práce.

V neposlednom rade by som sa chcel poďakovať môjmu spolužiakovi a dobrému kamarátovi *Bc. Lukášovi Tomášikovi* za jeho cenné rady pri tvorbe aplikácie.

Abstrakt

V našej diplomovej práci sa zaoberáme súčasným trendom testovania softvéru, metódami a druhmi testovania, a taktiež aj spôsobmi testovania študentov na školách. Praktická časť práce sa zaoberá testovaním aplikácií vytvorených vo vývojovom prostredí Delphi. Naším cieľom je vytvoriť používateľsky príjemné prostredie testovacej aplikácie, ktorá je schopná komunikovať s externými aplikáciami. Testované aplikácie sú programy vytvárané študentmi na hodinách programovania v programovacom jazyku Delphi. Testovacia aplikácia by mala byť schopná posúdiť funkcionalitu a vyhodnocovať správnosť výsledkov poskytovaných aplikáciou študenta, po naplnení jej vstupov hodnotami. Algoritmus testovacieho programu je založený na komunikácii dvoch aplikácií prostredníctvom WinAPI funkcií implementovaných v prostredí programovacieho jazyka Delphi. Po otvorení aplikácie nadväzujeme komunikáciu prostredníctvom identifikátora okna, ktorý získame prehľadaním všetkých okien v systéme, ktoré sú práve spustené. Využívame pritom vzťahy medzi oknami a procesmi v systéme Windows. Po získaní identifikátora okna posielame aplikácii správy, v ktorých definujeme požiadavky, ktoré sa majú v danom okne vykonať. Jedná sa o zapisovanie údajov, vykonanie udalosti na tlačidlo a čítanie údajov z komponentu. Výmenou informácií medzi aplikáciami posudzujeme správnosť získaných údajov a informácie o úspešnosti z jednotlivých pokusov testovania. Tieto informácie následne uchovávame pre potreby neskoršieho použitia. Na tento účel používame databázu, ktorá uchováva a poskytuje údaje o výsledkoch testovania študentov.

Kľúčové slová: WinAPI. Okno. EXE. Handle. Unit.

Abstract

In our diploma thesis we deal with the current trend in software testing, testing methods and types, and also ways of testing students in schools. The practical part deals with the testing of applications built in Delphi. Our goal is to create a pleasant user friendly test application that is able to communicate with external applications. Tested applications are programs created by students at the lessons of programming in the Delphi programming language. The test application should be able to assess the functionality and evaluate the accuracy of results provided by the student application, after filling its input values. Algorithm of testing program is based on two applications communicating via WinAPI functions implemented in an environment of Delphi programming language. After opening the communication we relate through ID window, we will search all the windows in the system that are currently running. We use relations between the windows and processes on Windows. Once we obtain the application window, we send messages, which define the requirements to be executed in that window. It is the writing of data to the specific window, execution events and reading data from the component. Exchange of information between applications, considering the accuracy of data and information on the success of individual trials testing. This information is stored for later use needs. For this purpose we use a database that stores and provides data on student test results.

Key words: WinAPI. Window. EXE. Handle. Unit.

Obsah

Zoznam termínov a skratiek **8**

Úvod **9**

1 Aktuálny stav riešenej problematiky **12**

1.1 Čo je testovanie? 12

1.2 Fázy v procese testovania 13

1.3 Základné delenie testovania 14

1.4 Druhy testov 15

1.4.1 Unit testing – Testovanie unitov 15

1.4.1.1 Štruktúra unit testu 16

1.4.2 Integračné testy 16

1.4.3 Akceptačné testovanie 17

1.4.4 Funkcionálne testovanie 17

1.5 Metódy testovania 18

1.5.1 Metóda black box 18

1.5.1.1 Využitie black box 18

1.5.1.2 Výhody a nevýhody black box 19

1.5.2 Metóda white box 19

1.5.2.1 Využitie white box 19

1.5.2.2 Výhody a nevýhody white box 20

1.5.3 Metóda grey box 20

1.5.3.1 Využitie Grey box 20

1.5.3.2 Výhody a nevýhody grey box 21

1.6 Vyhodnotenie metód 21

1.7 Nástroje pre automatizované testovanie 22

1.7.1 Kategórie nástrojov 22

1.7.1.1 Enterprise riešenia 22

1.7.1.2 Stredná kategórie 22

1.7.1.3 Open source riešenia 23

1.8 Automatizované testovanie s Framework Delphi Dunit 23

1.8.1 Používané metódy Check 24

1.8.2 Spustenie testu 25

1.9 Outsourcing testovania softvéru 27

1.10 Testovanie v školstve 30
1.11 Súčasné trendy testovania na školách 32
1.11.1 Elektronické testovanie 32
1.11.2 Elektronické testovanie v LMS Moodle 32
1.11.3 Spôsoby testovania aplikácií v predmete programovanie 34
2 Cieľ práce 35
3 Metodika 37
4 Vlastná práca 40
4.1 Použitý software Borland Delphi 40
4.2 Prečo sme sa rozhodli pre Delphi? 41
4.3 Softvérová a hardvérová konfigurácia 42
4.4 Výber testovacej metódy 43
4.5 Popis prostredia testovacieho softvéru 44
4.6 Programová štruktúra aplikácie 49
4.7 Programové funkcie v aplikácii na prácu s DBS 50
4.8 Návrh databázového systému pre aplikáciu 52
4.8.1 Význam jednotlivých atribútov entít 53
4.9 Činnosť aplikácie 54
4.10 Hlavný program systému testovania 56
4.11 Systémové prostriedky na prácu s EXE súbormi 58
4.12 Stratégia práce pre manipuláciu s oknami 58
4.12.1 Unit WorkWithHandles 59
4.12.1.1 Zoznam použitých funkcií a procedúr 60
4.12.1.2 Popis funkčnosti podprogramov: 61
4.12.2 Unit UnitWithTasks 72
4.13 Možnosti vylepšenia aplikácie Online testovanie 74
4.14 Výsledky práce a testovanie aplikácie 76
Záver 79
Zoznam použitej literatúry 82
Prílohy 85

Zoznam termínov a skratiek

API - skratka pre *Application Programing Interface*, rozhranie pre programovanie aplikácií tvorené procedúrami, funkciami a triedami a je súčasťou operačného systému

ASCII - anglická skratka pre *American Standard Code for Information Interchange*, kódovací systém znakov anglickej abecedy, číslic, iných znakov a riadiacich kódov

DUnit - nástroj na vytváranie, spúšťanie a vykonávanie testov jednotiek programového kódu v Delphi

E-learning - efektívne využívanie Informačných Technológií v procese vzdelávania

Entita - objekt reálneho sveta, ktorý je schopný nezávislej existencie a je jednoznačne odlíšiteľný od ostatných objektov

EXE - prípona v názve počítačového súboru, označuje spustiteľný súbor

Framework - softvérový nástroj používaný na vytváranie štandardnej štruktúry aplikácie pre konkrétny operačný systém

GUI - anglicky *Graphical User Interface*, skratka pre grafické používateľské rozhranie

Handle – jedinečný *identifikátor* okien a procesov

IDE - *Integrated Development Environment*, integrované vývojové prostredie (editor, prekladač, ladiace prostriedky a podporné utility) pre programovacie jazyky

IP - logický číselný identifikátor fyzického sieťového rozhrania v sieti, ktorý komunikuje s inými uzlami prostredníctvom *protokolu IP*

LMS - systém riadenia výučby *(Learning Management System)*, aplikácia na administratívu a organizáciu výučby v rámci elektronického vzdelávania

MOODLE - *Modular Object-Oriented Dynamic Learning Environment* (modulárne objektovo orientované dynamické výukové prostredie), softvérový balík pre vytváranie kurzov založených na Internete a web stránkach

Online - stav, keď je zariadenie aktivované a pripravené na činnosť, schopné kontroly alebo komunikácie s počítačom

SQL - *Structured Query Language*, počítačový jazyk na manipuláciu (výber, vkladanie, úpravu a mazanie) a definíciu dát

Unit - nezávislá entita programového kódu v programovacom jazyku Object Pascal

Úvod

Testovanie, vo všeobecnosti, sa v súčasnosti stalo takmer neoddeliteľnou súčasťou životného cyklu vývoja produktov a služieb. Aby sme docielili zhodu s požadovanými parametrami testovaných objektov, musíme testovať dostatočne kvalitne a precízne. Môžeme povedať, že testovaním dosiahneme overenie kvalitatívnych znakov skúmaných produktov. Ak by sme preniesli proces testovania do školského prostredia, tak okrem testovania produktov, či služieb je možné testovať napríklad žiakov alebo študentov. Proces testovania je väčšinou úzko spätý s hodnotením získaných údajov. Overujúce skúšky a testy používame na získanie vedomostí a schopností študentov. Aby sme mohli testovanie označiť za objektívne, je potrebné aby sme pri testovaní študentov, či žiakov použili rovnaké pravidlá a podmienky. Cieľom našej práce je zaoberať sa testovaním v predmete Programovanie. Rozhodli sme sa, že skombinujeme testovanie študentov a testovanie aplikácií a vytvoríme testovacie prostredie „Online testovanie aplikácií“ vo vývojovom prostredí Delphi. Diplomová práca je z obsahového hľadiska rozdelená na dve časti, teoretická a praktická časť.

V teoretickej časti našej práce sa budeme venovať súčasným trendom a metódam testovania. Nakoľko existuje v dnešnej dobe obrovské množstvo testovacích metód a nástrojov, vyberieme a podáme dôležité informácie o niektorých z nich a zároveň si vysvetlíme výhody a nevýhody pri ich používaní. V krátkosti objasníme možnosti výberu testovacích nástrojov, ich dostupnosť pre verejnosť i špecializované inštitúcie zaoberajúce sa výhradne testovaním softvérových aplikácií. Vysvetlíme postupnosť krokov testovania vo vývojom prostredí Delphi pomocou automatizovaného nástroja *framework DUnit*.

Testovanie a činnosti s ním spojené môžu vykonávať ľudia i počítače, preto hovoríme o manuálnom a automatizovanom testovaní. Automatizované testovanie, ktorým sa budeme v našej praktickej časti práce zaoberať, vykonáva počítačový program, ktorý má vopred definované, akým spôsobom, a ktoré časti softvéru má kontrolovať na základe vstupnej množiny hodnôt. Vzhľadom na zložitosť a rozmanitosť softvérových aplikácií však neexistuje žiadny univerzálny nástroj testovania, preto je potrebná prítomnosť ľudskej zložky.

V praktickej časti budeme realizovať problematiku testovania aplikácií metódou čiernej skrinky a vytvoríme testovací systém „Online testovanie aplikácií“ pre účely

testovania študentov na školách v predmete Programovanie. Rozhodli sme sa testovať funkcionalitu aplikácií použitím spôsobu čiernej skrinky, to znamená, že nepoznáme zdrojový kód testovanej aplikácie, ale vieme presne, aké výstupné hodnoty má aplikácia poskytovať. Systém budeme konštruovaný tak, že nebude potrebovať k dispozícii zdrojový kód aplikácie, ale stačí mu len výsledný spustiteľný *EXE* súbor výslednej aplikácie. Aplikácia nebude určená na testovanie kvality zdrojového kódu, pretože pri prevode vstupných hodnôt na výstupné nie je zaručené, že pri behu programu sa použijú všetky vetvy a podmienky zdrojového kódu. Študentove programy sa pokúsime testovať z hľadiska splnenia požadovanej funkcionality a korektnosti poskytovaných výstupov. Algoritmus aplikácie Online testovanie chceme riešiť tak, že algoritmus programu bude založený na komunikácii dvoch aplikácií prostredníctvom *Windows API* funkcií implementovaných v prostredí Delphi. Dôležité pre nás bude nadviazanie komunikácie s oknom. Po nadviazaní komunikácie budeme môcť aplikácii posielať správy, v ktorých definujeme požiadavky, ktoré sa majú v danom okne vykonať, resp. zapisovanie, čítanie údajov a vykonanie udalosti na tlačidlo. To znamená, že dôležité bude pre nás sledovať proces spracovávania a transformácie vstupných údajov na požadované výstupné hodnoty. Výmenou informácií medzi aplikáciami posúdime správnosť získaných údajov a následne sa informácie uchovávajú pre potreby neskoršieho použitia. Ďalšou úlohou aplikácie Online testovanie, ktorú budeme realizovať je, aby po ukončení testovania bolo možné prezerať hodnotenia a výsledky úspešnosti jednotlivých študentov. Aplikáciu prepojíme s databázovým systémom, ktorý uchová všetky potrebné informácie o testoch študenta. Databáza zatiaľ nebude umiestnená na webovom serveri, ale plánujeme ju nasadiť zatiaľ len v offline móde prostredníctvom *databázového klienta*.

Túto tému sme sa rozhodli vypracovať a lokalizovať do školského prostredia z viacerých dôvodov. Do školského prostredia sme sa rozhodli integrovať testovacie metódy z dôvodu snahy zvýšiť efektívnosť, objektívnosť, jednoduchosť získavania obrazu vedomostí študentov a evidencie výsledkov študentov z jednotlivých testovacích pokusov. Výsledkom aplikácie automatizovaného testovania je jednoduchšie spracovávanie údajov, uľahčenie práce so zadávaním a vytváraním testov, výrazné skrátenie času potrebného na kontrolovanie zdrojových kódov pedagógmi a sprehľadnenie celej činnosti testovania od zadania až po hodnotenie. Táto vízia položila základ pre vytvorenie automatizovaného testovacieho softvéru určeného práve na efektívne získavanie informácií o vedomostiach študentov z predmetu

Programovanie s využitím vývojového prostredia Borland Delphi. Chceli by sme navrhnúť a vytvoriť systém, ktorý by sa líšil od klasických spôsobov testovania, ktoré sa bežne používajú na školách. Existuje množstvo nástrojov, ktoré slúžia na vytváranie, náhodné generovanie, automatickú kontrolu a evidenciu testov. Absencia testovacích softvérov na kontrolu programovaných aplikácií v danom programovacom prostredí nás podnecuje k riešeniu tejto situácie. Našim cieľom je touto prácou a používaním nášho nástroja motivovať a viesť študentov k logickému, algoritmickému mysleniu a používaniu stanovených pravidiel, na ktorých sa vopred vyučujúci so študentmi dohodne. Rozhranie aplikácie sme sa rozhodli integrovať do školského prostredia s cieľom uľahčiť pedagógom kontrolovanie študentských prác a dosiahnuť jednoduchšiu evidenciu výsledkov vypracovaných zadaní študentov.

1 Aktuálny stav riešenej problematiky

1.1 Čo je testovanie?

Dnešná doba je charakteristická rýchlym vývojom softvérových aplikácií a požiadavky v oblasti kvality sú oproti minulosti čoraz väčšie. S rýchlym rozvojom softvérového priemyslu sa rozvíja aj konkurenčný boj spoločností produkujúcich softvérové riešenia, o získanie, či udržanie si zákazníka. Schopnosť konkurovať ostatným produktom na trhu podnecuje spoločnosti k neustálemu zlepšovaniu svojich technológií a k snahe podieľať sa na zvyšovaní kvality softvérových systémov.

Jedným z dôležitých variantov, ako docieliť zvýšenie kvality softvéru, je kvalitné testovanie. Cieľom testovania je overiť a podať dôkaz o tom, že vyvíjaný softvér spĺňa požadované vlastnosti kvality a funkcionality. Každá chyba aplikácie znamená určité riziko a možné straty, preto má testovanie v súčasnosti veľký význam pri hľadaní chýb a defektov. Čím dlhšie zostanú chyby neodhalené, tým väčšie straty môžu vzniknúť. Včasné odstránenie chýb pomáha šetriť čas a náklady v prípade ich neskoršieho odstraňovania. Testovanie sa stalo takmer neoddeliteľnou etapou v životnom cykle vývoja softvéru. V procese testovania najčastejšie zisťujeme, či aplikácia vykonáva úkony, ktoré sa od nej požadujú v špecifikácii, to znamená akým spôsobom sa dokáže vysporiadať s neštandardnými stavmi, ako odpovedá aplikácia na zvýšenú záťaž, či deficit systémových prostriedkov a pod. Zároveň sa testujú reakcie systému na chyby spôsobené používateľom a stabilita aplikácie pred útokmi. Ak sa počas testovania nezistia žiadne chyby alebo nedostatky, nemusí to vždy znamenať, že v aplikácii sa reálne žiadne chyby nevyskytujú. Softvérovú aplikáciu rovnako ako vykonávanie testov uskutočňujú ľudia, ktorí sa môžu zmýliť. Existuje aj možnosť, že testovanie sa nevykonalo v potrebnej miere. Na základe počtu odhalených chýb je možné vyvodiť viacero záverov a určiť kvalitu softvéru, testovacích metód a testov: (ČERMÁK, 2010a)

1. Kvalita softvéru a vykonaných testov je vysoká. Predpokladané je odhalenie malého počtu chýb.
2. Kvalita softvéru je vysoká, ale kvalita testov nízka. Predpokladané je odhalenie ešte menšieho počtu chýb ako v prvom prípade.
3. Kvalita softvéru a testov je nízka. Predpokladané je odhalenie malého počtu chýb, i keď softvér ich môže obsahovať veľké množstvo.
4. Kvalita softvéru je nízka a kvalita testov naopak, vysoká. Predpokladané je odhalenie veľmi vysokého počtu chýb. (ČERMÁK, 2010a)

Niekedy môže byť presvedčenie o kvalite softvéru alebo testov mätúce. Nízky počet odhalených chýb nemusí byť vždy výsledkom vysokej kvality softvéru a použitých testov. Softvér väčšinou obsahuje veľké množstvo chýb, no kvôli nízkej kvalite testovania sa chýb môže detekovať len zopár. (ČERMÁK, 2010a)

Testy je možné zostaviť ešte skôr ako sa začne s programovaním softvéru, je však potrebné mať k dispozícii *SRS dokument (Software Requirements Specification)*. Pri vývoji softvéru by mala vždy existovať špecifikácia úloh, čo má daný softvér vykonávať a testy sa môžu navrhnúť podľa funkčných požiadaviek uvedených v *SRS*. Možno tak vytvoriť testovacie prípady pre manuálne aj automatizované testy.
(ČERMÁK, 2010a)

Jedna z vlastností, ktorú by mala testovaná aplikácia spĺňať z hľadiska vlastností kvality je testovateľnosť.

„Testovateľnosť (testability) opisuje ako ľahké alebo ťažké je testovať aplikáciu. Počiatočné rozhodnutia pri návrhu môžu mať veľký vplyv na počet potrebných testovacích vzoriek. Základné pravidlo hovorí, že čím je návrh zložitejší, tým náročnejšie bude ho podrobne otestovať. Jednoduchý návrh, resp. návrh využívajúci už otestované komponenty znižuje požiadavky na testovanie.“(BIELIKOVÁ, 2007, s.23)

1.2 Fázy v procese testovania

Testovanie je zložitý systém a možno ho rozdeliť do piatich častí. Prvou časťou je plánovanie testovania. V tejto fáze sa definujú ciele testovania a vytvára sa špecifikácia činností, ktoré sú prostriedkom pre splnenie hlavného cieľa. Druhú fázu tvorí analýza a návrh testovania, kde prichádza ku konkretizácii testovacích podmienok, vznikajú zvyčajne testovacie scenáre a testovací plán. Vyhodnocuje sa testovateľnosť základnej časti a testovateľnosť objektov testovania. Tretia fáza je vykonanie a implementácia testovania. Ide o fázu, v ktorej prebieha nastavenie a prevedenie testov prostredníctvom testovacích procedúr, ktoré musia byť v súlade s testovacím plánom. Porovnávajú sa získané výsledné hodnoty s hodnotami očakávanými, resp. hľadajú sa defekty. Štvrtou fázou je vyhodnotenie a reportovanie. Fáza vyhodnotenia a reportovania je významná činnosť hodnotiaca vykonané testy s predsavzatými cieľmi a plánmi testovania. Výsledkom je súhrnná správa a hodnotenie potrebnosti vykonávania ďalších testov. V poslednej fáze, aktivity uzatvorenia testu, sa kontroluje

stav, či boli uzatvorené všetky incidenty a archivujú sa všetky technické náležitosti pre prípad neskoršieho použitia. (MÜLLER, 2007)

1.3 Základné delenie testovania

Testy sa rozdeľujú podľa spôsobu prístupu k údajom dôležitým na uskutočnenie testov na *white box, black box a grey box* testovanie. Testy je možné členiť aj na základe subjektu, ktorý ich realizuje. Týmto spôsobom sa rozdeľujú testy na testy vykonávané: (ČERMÁK, 2010a)

1. *Dodávateľom*, resp. poskytovateľom vykonávané testy, pričom subjekt sa špecializuje a vykonáva prevažne *unit testy*, *integračné a systémové testy*.
2. *Odberateľom*, zákazníkom, ktorý vykonáva predovšetkým *akceptačné testy*.

(ČERMÁK, 2010a)

Ďalšia možnosť ako rozdeliť testy je na základe spôsobu ich vykonania:

1. *Automatizované testy* – Výhodou týchto testov je ich rýchlosť a možnosť otestovať veľký počet vstupných informácií. Tieto testy je možné veľmi ľahko opakovať v čase. Automatizované testy sa využívajú všade tam, kde treba vykonávať mnohonásobné opakovanie testu s inými údajmi, napríklad pri záťažových testoch. Náročnosť týchto testov spočíva v tom, že ich treba vopred programovať. (ČERMÁK, 2010a)

   Automatizovaným testovaním softvéru môžeme chápať aj testovanie procesu alebo jeho časti, ktoré sa vykonáva bez priameho kontaktu s človekom. Používa sa pritom špecializovaný softvér. Najznámejšie softvérové aplikácie z kategórie automatizovaných nástrojov sú tie, ktoré dokážu nahradiť manuálne kliknutie testera v aplikácii. (HERBST, 2010a)
2. *Manuálne testy* – Charakteristickou vlastnosťou manuálnych testov je ich pomalší priebeh. Nie je nimi možné otestovať tak veľké množstvo vstupných údajov v reálne krátkom čase ako pri automatizovanom spôsobe. Manuálne testy však majú v súčasnosti široké uplatnenie pri testovaní rozsiahlych softvérových projektov. Všetky výsledky a testy je nutné písať ručne, no na druhej strane manuálne testy sa vyznačujú absenciou nutnosti programovania testov. Existujú však prípady testov, kde by uplatnenie automatizovaného spôsobu bolo veľmi náročné a preto je nevyhnutné používať práve manuálny spôsob, napríklad test použiteľnosti. (ČERMÁK, 2010a)

Posledné je delenie testovania podľa času, kedy sa vykonáva:

1. *Statické testovanie* – Realizuje rozoberaním zdrojového kódu nespusteného softvéru a hľadajú sa v ňom chyby. Statické testovanie poskytuje možnosť nájsť syntaktické chyby a taktiež poskytuje diagnostiku zdrojového kódu. (ČERMÁK, 2010a) (TRNKA, 2008)
2. *Dynamické testovanie* – Vykonáva sa počas behu programu, kedy prebieha aj hľadanie chýb. Dynamické testovanie zväčša nachádza menej chýb ako statické testovanie. (ČERMÁK, 2010a) (TRNKA, 2008)

1.4 Druhy testov

1.4.1 Unit testing – Testovanie unitov

Prvý spôsob testovania nazvaný podľa veľkosti testovanej jednotky je *Unit testing*. Jedná sa o testovanie najmenšej testovateľnej jednotky v rámci systému. *Unit test* je množina testov, ktorá kontroluje a overuje jedinú triedu alebo podprogram izolovaný od všetkých ostatných častí programu. *Unit testing* je jedna z techník extrémneho programovania. Procedúry, pomocou ktorých overujeme, či všetky individuálne jednotky zdrojového kódu pracujú podľa stanovenej definície, sa nazývajú *unit test*. *Unit testing* nám umožňuje pre každú verejných metód tried zisťovať ich návratovú hodnotu a porovnávať ju s očakávanou. *Unit test* korektne prejde, ak sa očakávaná hodnota rovná návratovej. Na testovanie *unitov* slúžia programy umožňujúce vytváranie testovacích vstupov a čítanie výstupov. Výhodou je možnosť ich neustáleho spúšťania. (GLONČÁK, 2000)

Individuálnu jednotku si možno predstaviť ako najmenšiu časť aplikácie, ktorú môžeme testovať. V objektovo orientovanom programovaní môže túto jednotku reprezentovať napríklad trieda, za najmenšiu testovateľnú jednotkou však môžeme považovať i samostatný podprogram, funkcia, či procedúra. Cieľom *unit testing* je testovanie korektnosti jednotlivých častí programu. Jednotlivé testy spustia požadovaný *unit* s definovanými parametrami a výsledok porovnávajú s očakávanou hodnotou. Hlavnou náplňou *unit testing* je overenie, či použité metódy a podprogramy pracujú tak, ako sú na ne kladené požiadavky. (JANEK, 2007)

1.4.1.1 Štruktúra unit testu

Potreba veľmi častého opakovania testov, ich spúšťanie a úprava, viedla k automatizácii procesu testovania. Extrémne programovanie vzhľadom k tomu, že nemá priamu spojitosť so žiadnym vývojovým nástrojom, používa hlavne voľne distribuované nástroje. Ide o nástroje *xUnit*. Tento súbor testovacích nástrojov by sme mohli považovať za rodinu testovacích nástrojov slúžiacich na vykonávanie *unit testov*. Písmeno „x“ v názve konkrétneho nástroja značí programovací jazyk alebo platformu, pre ktorý je určený prístup k testovaniu. Napríklad *JUnit* pre *Javu*, *DUnit* pre Delphi a pod. Nástroje *xUnit* špecifických vývojových prostredí väčšinou ponúkajú štandardne nasledovné súčasti: (BUCHALCEVOVÁ, 2008)

- *Test fixture*: Časť, ktorá je tvorená procesmi potrebnými na uskutočnenie jednotlivých testov. V tejto časti môže byť obsiahnuté vytvorenie inštancií, objektov a pod.
- *Test case*: Je jeden prípad, ktorý definuje požiadavku na testovanú oblasť. Je to najmenšia testovaná jednotka (*unit*). Obsahuje volanie konkrétnej časti programu, pričom volíme parametre tak, aby sme mohli pri získaní návratovej hodnoty porovnávať túto hodnotu s nami predpokladanou hodnotou. Je potrebné dbať na údajový typ, použitú syntax pri zadávaní vstupných hodnôt. Taktiež je treba dávať pozor na rozsah používaných hodnôt a hraničných hodnôt.
- *Test suite*: Ak sa má skupina testovacích prípadov vykonať naraz, potom na zoskupovanie týchto prípadov slúži práve skupina prípadov *test suite*.
- *Test runner*: Prostredie testovacieho nástroja, ktoré vizuálnou formou slúži na realizáciu testov. Zároveň poskytuje výsledky testovania. (JANEK, 2007)

Okrem *unit testing* je známych ešte niekoľko druhov testov, napríklad: *Integračné testy, Systémové testy, Akceptačné testy, Funkcionálne testovanie, Štrukturálne testovanie.*

1.4.2 Integračné testy

Pri *integračnom* testovaní je dôležité, aby sme rozlišovali testovanie vnútorné a vonkajšie. Vnútorné je založene na testovaní komunikácie medzi viacerými jednotkami programu, modulmi. Vonkajšie *integračné* testovanie je založené na integrácii viacerých aplikácií do zložitejších celkov. *Integračné* testovanie je druh testovania, ktorý sa používa hlavne na overenie spolupráce skupiny modulov, resp.

aplikácií, zároveň sa zisťuje ich spolupráca medzi sebou. Ako vyplýva z názvu, pri tomto spôsobe sa testuje integrácia modulov v systéme, integráciu môžeme chápať ako spájanie jednotlivých častí. (HERBST, 2010b)

1.4.3 Akceptačné testovanie

Akceptačné testovanie je testovanie, ktoré nasleduje po integračnom testovaní, vykonáva ho sám zákazník ešte predtým, ako sa uvedie produkt do prevádzky. Cieľom tohto testovania je dosiahnutie dôvery k testovanému systému. Zákazník sám alebo ním poverená osoba testuje spôsobilosť aplikácie, to znamená, či je produkt spôsobilý prevádzky. Ak *akceptačné* testovanie prebehne pozitívne, produkt sa môže zaviesť do prevádzky. (MÜLLER, 2007)

1.4.4 Funkcionálne testovanie

Funkcionálne testovanie sa zameriava na zistenie a overenie, či daná softvérová aplikácia vyhovuje špecifikáciám z hľadiska výstupných hodnôt dosiahnutých na základe zadaných vstupných hodnôt. Neberieme do úvahy vnútornú štruktúru, ale sledujeme správanie sa systému z externého hľadiska a zisťujeme, či sa vstupy korektne spracovávajú na výstupy. Do tejto kategórie patria aj programy, ktoré imitujú používateľa komunikujúceho s časťou *GUI*. Automatizovaním *funkčných* testov môžeme dosiahnuť výrazné skvalitnenie programu, pretože na zistenie funkcionality overujú celý systém. Tento druh testovania úzko súvisí s metódou čiernej skrinky. Úplné vykonanie testu funkcionality však nie je možné kvôli početnosti možných vstupov. Vyberáme takú množinu vstupov, ktorej pravdepodobnosť príslušnosti k chybovým vstupom je vysoká. Je možné uskutočniť rozdelenie vstupov a výstupov do tried ekvivalencie podľa vstupnej a výstupnej podmienky. Popis jednotlivých tried ekvivalencie je nasledovný: (GLONČÁK, 2000) (MÜLLER, 2007) (BIELIKOVÁ, 2000)

1. Každý jeden vstup alebo výstup patrí do určitej triedy ekvivalencie.
2. Z toho žiadny vstup alebo výstup nepatrí do viacerých tried.
3. Zistenie chyby pri konkrétnom vstupe z triedy ekvivalencie nám umožňuje detekovať chybu výberom inej hodnoty z rovnakej triedy ekvivalencie. (BIELIKOVÁ, 2000)

Funkcionálne testovanie je základným strategickým pilierom riešenia praktickej časti našej práce.

1.5 Metódy testovania

1.5.1 Metóda black box

Testovanie *black box*, často označované ako *funkčné* testovanie, je spôsob, akým realizovať testovanie bez znalosti vnútornej dátovej štruktúry systému. Zdrojový kód ani dokumentácia nie je k dispozícii, preto je za potreby používať testovacie scenáre. Metódou *black box* zisťujeme správanie sa systému, to znamená, či sa systém ako celok správa tak, ako je definované v špecifikácii. Na vstup testovanej aplikácie sa privedú určené hodnoty, a na základe znalosti hodnôt, ktoré sa musia objaviť na výstupe sa kontroluje ich správnosť. Programátor nemá informácie, ktoré sa týkajú obsluhy výnimiek a chýb a časti zdrojového kódu, ktorý sa vykonáva. Pre aplikáciu sú jednoznačne definované rozsahy a typy povolených a nepovolených hodnôt pre výstup na základe určených vstupov. Pri opakovanom zadaní rôznych vstupných hodnôt na vstup aplikácie tester vie, aké výstupné hodnoty alebo správanie aplikácie môže očakávať. *Black box* metóda môže prebiehať celkom manuálne, ale i úplne automatizovane. Používajú sa pritom rôzne nástroje. *Black box* je metóda ideálna pre použitie v prípadoch, keď sú známe alebo presne definované hodnoty alebo rozsahy možných hodnôt. *Black box* si môžeme predstaviť ako čiernu skrinku s nedefinovanou vnútornou štruktúrou. (ČERMÁK, 2010b) (WILLIAMS, 2006a)

1.5.1.1 Využitie black box

Black box sa využíva hlavne na zistenie chýb už bežiaceho systému alebo aplikácie. Ku kompletnému vykonaniu testu nám postačuje samotný program alebo jeho umiestnenie. Výhodou *black box* testovania je v prvom rade jednoduchosť, nakoľko testy môže vykonávať i osoba bez znalosti programovacích jazykov. Vďaka rýchlosti je možné relatívne v krátkom čase otestovať aj väčšie systémy. Testovanie je však závislé na testovacích scenároch. Zmenou programovacieho jazyka alebo operačného systému sa spôsob a priebeh testovania nezmení. Výhodou je aj fakt, že tester nepotrebuje k dispozícii zdrojový kód. (ČERMÁK, 2010b)

1.5.1.2 Výhody a nevýhody black box

K veľkým nevýhodám *black box* patrí nízka predpovedateľnosť kvality zdrojového kódu. V prípade, že na výstupe sa neobjaví neočakávaná hodnota, nie je zaručené, že zdrojový kód aplikácie v poriadku. Z hľadiska efektívnosti môže obsahovať rôzne defekty. Okrem požadovaných funkcií sa môže objaviť i nežiaduce správanie sa aplikácie, ktoré sa prejavuje napríklad vykonávaním nešpecifikovanej operácie, ktorá sa neprejaví na výstupe. (ČERMÁK, 2010b)

1.5.2 Metóda white box

Predpokladom pre testovanie metódou *white box*, niekedy nazývanou *glass box* alebo *open box*, je znalosť vnútornej štruktúry softvéru. Vnútornou štruktúrou môžeme chápať charakter a usporiadanie vnútorných dátových štruktúr. Pri realizovaní testovania *white box* má tester k dispozícii dokumentáciu softvéru a taktiež zdrojový kód testovanej aplikácie. *White box* vďaka podrobným informáciám vnútornej štruktúry má možnosť testovať všetky existujúce cesty, ktoré môže daný program vykonať. Je možné testovať aj chybové vstupy. Pri použití metódy *black box* sa overujú iba výstupné hodnoty, no metóda *white box* nám umožňuje kontrolovať celé vnútro programu a zároveň overiť správnosť na výstupe. Z toho dôvodu je *white box* náročnejšia metóda, pretože je potrebné kontrolovať všetky vnútorné programové cesty. Pre korektné testovanie je nutné pochopiť zdrojový kód. V literatúrach je často tento spôsob označovaný ako audit zdrojového kódu. Realizácia *white box* testovania sa uskutočňuje buď ručne alebo úplne automatizovane. V praxi sa však mnohokrát stretávame s vhodnou kombináciou týchto spôsobov. (ČERMÁK, 2010c) (WILLIAMS, 2006b)

1.5.2.1 Využitie white box

Testovanie *white box* je použiteľné napríklad pre webové služby pri hľadaní chýb testermi a vývojovými pracovníkmi. *White box* test môžeme takisto využiť aj na zistenie, akým spôsobom je spravovaný prístup k častiam aplikácie a k jej údajom. *White box* metóda slúži tiež k nájdeniu nežiaduceho a potenciálne nebezpečného kódu. V praxi sa aplikácia navonok môže javiť ako bezchybná, a môže pracovať v súlade s dokumentáciou, no zdrojový kód úplne v poriadku byť nemusí. Existujú testy, ktoré sú zamerané na overovanie, či aplikácia funguje správne podľa požiadaviek v špecifikácii.

Ak sa nájdu určité nezrovnalosti a chyby, je možné, že práve tieto chyby spôsobujú neštandardné správanie sa aplikácie počas prevádzky. Neželaný kód môže vykonávať akúkoľvek činnosť a môže sa aktivovať v rôznom čase alebo ako reakcia na nejakú udalosť. V každom prípade je však dôležité, nehľadiac na príčinu výskytu nežiaduceho kódu, tento kód včas odhaliť a odstrániť. *White box* má oproti *black box* testovaniu výhodu v tom, že chyby je možné nájsť aj iným spôsobom ako náhodným. (ČERMÁK, 2010c)

1.5.2.2 Výhody a nevýhody white box

K výhodám *white box* testovania patrí odhalenie nežiaduceho kódu a odhalenie nezrovnalostí skôr, než sa zdrojový kód skompiluje. K nevýhodám *white box* testovania patria hlavne vysoké investičné náklady pri analýze zdrojového kódu, potreba dokonalej znalosti testovaného systému a programovacích jazykov. (ČERMÁK, 2010c)

1.5.3 Metóda grey box

Grey box testovanie je v literatúrach často označované aj výrazom *translucent box*. Hlavným znakom *grey box* je neúplná znalosť vnútornej dátovej štruktúry softvéru. Testovacie scenáre sa realizujú metódou *black box*. Môžeme povedať, že tento spôsob vznikol zlúčením oboch testovacích metód, *black box* a *white box*. Nemožno preto hovoriť čisto o *black box*, nakoľko čiastočne poznáme vnútorné programové členenie a často máme prístup k údajom v databáze. *White box* sa zase líši práve tým, že znalosť vnútornej štruktúry rozoberáme do hĺbky. Veľkým prínosom *grey box* testovania je fakt, že je veľmi jednoduché. Pretože tester vie, že softvér vo vnútri funguje správne, má možnosť tak lepšie vykonať testovanie zvonka. (ČERMÁK, 2010d)

1.5.3.1 Využitie Grey box

Grey box využíva obe predošlé testovacie metódy. *White box* sa používa k nájdeniu slabých stránok softvéru a oblastí bezpečnostných chýb. *Black box* metóda sa potom použije na zistenie, či nájdené miesta s bezpečnostnými chybami môžu byť miestom vykonania útoku na systém. Často sa tento typ používa na testovanie aplikácií, u ktorých máme kontrolu nad vstupmi a výstupmi a zároveň umožnený prístup k údajom v databáze. Počas testovania môžeme kontrolovať všetky tri hodnoty a zistiť, kde dochádza v aplikácii ku korekcii výsledkov, resp. výstupných hodnôt. *Grey box* je

zaužívaný aj v prípade, ak nechce poskytnúť testerovi zdrojový ani binárny kód, ale poskytujeme len architektúru celého systému. (ČERMÁK, 2010d)

1.5.3.2 Výhody a nevýhody grey box

K výhodám patrí prepojenie výhod *black box* a *white box* prístupu. Nie je potrebný zdrojový ani binárny kód, pretože testovanie sa zakladá na znalosti funkčnej špecifikácie a architektúre aplikácie. Nevýhodami tejto metódy je neúplné testovanie, ktoré spôsobuje absencia zdrojových a binárnych kódov. Preto nie je možné otestovať aplikáciu do hĺbky. Nevýhodou je takisto aj to, že nevieme zistiť, či aplikácia obsahuje nejaký škodlivý kód alebo či je kód dostatočne efektívny. (ČERMÁK, 2010d)

1.6 Vyhodnotenie metód

Nie je možné jednoznačne povedať, ktorá z metód a druhov testovania je najlepšia, či najuniverzálnejšia. Každý spôsob má svoje opodstatnenie a špeciálne prípady použitia v určitej fáze vývoja konkrétneho softvéru. Vo veľkej miere je výber testovacej metódy ovplyvnený možnosťami dodávateľa testov, zložitosťou. Pri výbere treba zvažovať a brať do úvahy fakt, kto a akým spôsobom bude danú aplikáciu používať. Všetky tri skupiny testovacích metód možno nechať realizovať domácim, ale aj externým zamestnancom a pracovníkom. Preto treba dbať na to, aby neprišlo k zneužitiu informácii alebo znalostí. Pri testovaní *white box* prichádza tester do priameho kontaktu so zdrojovým kódom a ak má dostatočné skúsenosti a znalosti, môže zdrojový kód zneužiť a využiť vo svoj prospech. Riziko pri použití *black box* sa už netýka zneužitia zdrojového kódu, nakoľko nie je k dispozícii, možno však odcudziť napríklad vizuálny návrh alebo dizajn softvéru. (ČERMÁK, 2010e)

V procese testovania je dôležité dosiahnuť splnenie niekoľkých podmienok. Z podmienok merateľnosti a opakovateľnosti vyplýva, že ak uskutočníme test nad rovnakou dátovou oblasťou viackrát s použitím rovnakých testovacích scenárov, je nutné dosiahnutie rovnakých alebo v zanedbateľnej miere odlišných výsledkov. Minimálne, zanedbateľné odchýlky môžu pri určitých typoch testovania bežne nastať. Najdôležitejší je výstup z testovania, resp. počet a typ nájdených chýb, počet a typ vyriešených a nevyriešených chýb. (ČERMÁK, 2010f)

Pri testovaní funkcionality aplikácie je veľmi dôležité dobre a efektívne zvoliť tie najdôležitejšie oblasti testovania. Neuváženým rozhodovaním pri výbere množiny

testovacích hodnôt môže táto množina vstupov nekontrolovane narásť, napríklad pri testovaní celého systému alebo podsystému. Je zrejmé, že takýmto spôsobom nebude možné s použitím akejkoľvek metódy testovanie vykonať úplne. Cieľom *funkcionálneho* testovania je vybrať takú podmnožinu testovacích prípadov, ktorá má veľkú pravdepodobnosť detekovať najväčšiu množinu chýb. Jedna možnosť je náhodný výber podmnožiny možných chybových vstupov. Z hľadiska efektívnosti je toto riešenie najhoršie, je totiž malá pravdepodobnosť, že náhodným prístupom natrafíme na optimálnu podmnožinu. (BIELIKOVÁ, 2000)

Najvhodnejšie metódy testovania vzniknú kombináciou základných metód. Zvoliť správnu kombináciu tak, aby silné a slabé stránky každej metódy boli v prijateľnom pomere, je kľúčovým faktorom pri výbere testovacích metód a nástrojov. (BIELIKOVÁ, 2000)

1.7 Nástroje pre automatizované testovanie

1.7.1 Kategórie nástrojov

V súčasnosti je na trhu početný sortiment nástrojov určených na automatizované vykonávanie testov. Líšia sa predovšetkým používateľským pohodlím, množstvom podporovaných technológií a v neposlednom rade sa líšia cenou. Z tohto hľadiska rozlišujeme tri kategórie automatizovaných nástrojov: (ČERMÁK, 2010g)

1.7.1.1 Enterprise riešenia

Pred časom prišli na trh s riešením automatizovaných testov firmy *Mercury Interactive* a *Rational Software*. V dnešnej dobe sú však už súčasťou najväčších svetových firiem *HP* a *IBM*. Služby týchto spoločností využívajú hlavne významné finančné inštitúcie. (ČERMÁK, 2010g)

1.7.1.2 Stredná kategórie

Do strednej kategórie zaraďujeme nástroje spoločností, ktoré ponúkajú menej univerzálne produkty v cenovej kategórii od stoviek až po tisíce dolárov. Napriek svojej menšej univerzálnosti však môžu splniť účel a poskytnúť potrebnú funkcionalitu. Môžeme sem zaradiť napríklad nástroj *Wapt* umožňujúci testovať záťaž webových stránok a aplikácií. Nástrojom automatizovaného testovania v strednej kategórii je napríklad aj *TestComplete* od spoločnosti *AutomatedQ*. (ČERMÁK, 2010g)

1.7.1.3 Open source riešenia

Voľnosť a neobmedzenosť vývojových prostredí viedla k vzniku niekoľkých open source riešení. Medzi najznámejšie z týchto riešení patria *frameworky: Selenium, JMeter, JTest, JUnit* a pod. (ČERMÁK, 2010g)

JUnit sa stal testovacím *frameworkom* pre jazyk *Java*. Neskôr bol *Junit* bol aplikovaný do ďalších jazykov, napríklad:

- PHP *(PHPUnit)*
- C# *(NUnit)*
- Python *(PyUnit)*
- Delphi *(DUnit)*
- Free Pascal *(FPCUnit)*
- C++ *(CPPUnit)*
- JavaScript *(JSUnit)* a iné. (ČERMÁK, 2010g)

1.8 Automatizované testovanie s Framework Delphi Dunit

Dunit je testovací *framework* pre Borland Delphi programy určený pre automatizované testovanie v tomto prostredí. Pôvodne bol inšpirovaný *frameworkom JUnit* napísanom v jazyku Java. *Dunit* je open source *framework* umožňujúci vyvíjanie, vytváranie a spúšťanie automatizovaných testov pre Delphi Win32 a C++ Builder aplikácie. *Dunit* obsahuje súbor vlastných metód pre testovanie podmienok. Je možné vytvárať vlastné podmienky a pritom využíva veľké množstvo metód. Každý test *Dunit* implementuje triedu typu *TTestCase*. Predtým ako príde k samotnému programovaniu testovacích prípadov, je potrebné najskôr správne nainštalovať *Dunit* do prostredia Delphi. (EMBARCADERO, 2010) (GOLKO, 2010)

Pre písanie testovacích prípadov je potrebné vytvoriť triedu zdedenú z *TTestCase*. Trieda *TTestCase* je deklarovaná v *unite TestFramework.* Je možné pridávať testy k triede testovacieho prípadu. *Test runner* detekuje, ktoré testy sú dostupné v triede testovacieho prípadu. Nachádzajú sa tu dve užitočné metódy: *Setup* a *TearDown*. Vykonanie každého individuálneho testovacieho prípadu v rámci triedy začína so *Setup*, následne sa vykoná procedúra, ktorá vykoná test a nakoniec program končí volaním procedúry *TearDown*. *TearDown* zabezpečí odstránenie objektov. (EMBARCADERO, 2010) (GOLKO, 2010)

1.8.1 Používané metódy Check

Celý postup zväčša obsahuje väčšie množstvo volaní metódy *Check*, ktoré sú uvedené v tabuľke č. 1. Metóda *Check* je deklarovaná nasledovne:

```
procedure Check(Podmienka: boolean; Sprava: string = ''); virtual;
```

(GOLKO, 2010)

Tab. 1 Zoznam metód Check

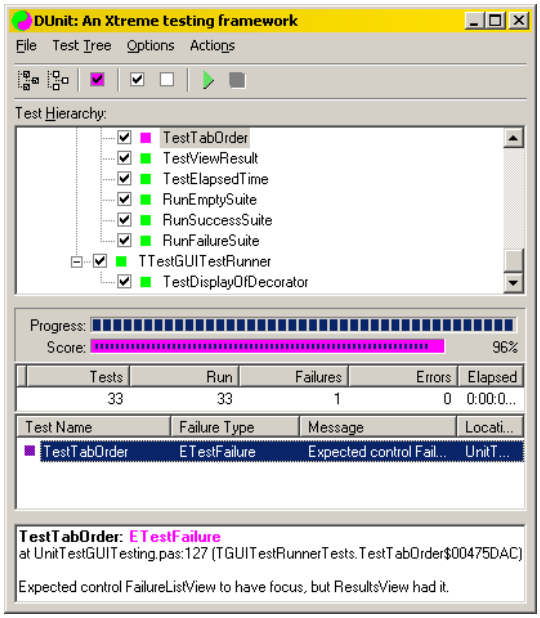
Zdroj: http://docwiki.embarcadero.com/RADStudio/2010/en/DUnit_Overview

Funkcia	Popis
Check	Skontroluje, či podmienka bola splnená.
CheckEquals	Skontroluje, či sa dve položky rovnajú.
CheckNotEquals	Skontroluje, či položky nie sú rovnaké.
CheckNotNull	Skontroluje, či položka je Not Null.
CheckNull	Skontroluje, či položka je Null.
CheckSame	Skontroluje, či dve položky majú rovnakú hodnotu.
EqualsErrorMessage	Kontrola, či chybové hlásenie zodpovedá špecifikovanému chybovému hláseniu.
Fail	Zisťuje zlyhanie.
FailEquals	Zisťuje, či poruchový stav sa rovná špecifikovanému poruchovému stavu.
FailNotEquals	Zisťuje, či poruchový stav sa nerovná špecifikovanému poruchovému stavu.
FailNotSame	Skontroluje, či dve zlyhané podmienky nie sú rovnaké.
NotEqualsErrorMessage	Skontroluje, či dve chybové správy nie sú rovnaké.
NotSameErrorMessage	Skontroluje, či chybové hlásenie nezodpovedá uvedenému chybovému hláseniu.

Testovacie sady môžu obsahovať viac testov, a tým poskytuje spôsob, ako sa zostaviť strom testov. Centrom operácií *Dunit* je *Test register*, ktorý udržuje všetky testovacie sady. Typicky sa *Test register* vytvára počas inicializácie aplikácie testu. Testovacie sady môžu byť vytvorené pomocou *TTestSuite* triedy, deklarovanej v *TestFramework*. Najvhodnejší a často používaný spôsob ako vytvoriť testovacie sady, je použiť metódu *Suite* triedy *TTestCase*, ktorý vytvorí nový objekt *TTestSuite*. (GOLKO, 2010)

1.8.2 Spustenie testu

Dunit obsahuje dve štandardné triedy *Test runner*: *TGUITestRunner* s interaktívnym rozhraním *GUI* a *TTextTestRunner* s režimom príkazového riadku. Testy by mali byť čo najúplnejšie. Je veľmi zložité pokryť všetky možné scenáre. Testy kontrolujú, či je vyvolaná výnimka, keď sa očakáva výnimka. *RunRegisterTest* je procedúra, ktorá spustí registrované *test suites*. Interaktívne rozhranie *GUI* je zobrazené na obrázku č. 1: (GOLKO, 2010)



Obr. 1 Používateľské rozhranie GUI

Zdroj: http://dunit.sourceforge.net/#Screenshot

Spustenie testu sa realizuje pomocou príkazu:

`GUITestRunner.RunRegisteredTest;`

(GOLKO, 2010)

Ukážkový časť zdrojového kódu, pred a po skončení cyklu, pre testovací prípad, ktorý slúži na kontrolu počtu položiek v *StringListe*:

```
unit Project1TestCases;
uses TestFrameWork, Classes;
type
 TTestCaseFirst = class(TTestCase)
 private
   MyList : TStringList;
 protected
```

```
   procedure SetUp; override;
   procedure TearDown; override;
 published
   procedure TestPopulateStringList;
 end;
procedure TTestCaseFirst.SetUp;
begin
 MyList := TStringList.Create;  // vytvorenie zoznamu
end;

procedure TTestCaseFirst.TearDown;
begin
 MyList.Free;                  // uvoľnenie zoznamu z pamäte
end;

procedure TTestCaseFirst.TestPopulateStringList;
var i : Integer;
begin
 Check(MyList.Count = 0);     // kontrola počtu záznamov na začiatku
 for i := 1 to 50 do
   MyList.Add('i');           // pridanie nového objektu
 Check(MyList.Count = 50);    // kontrola po naplnení
end;

initialization
 TestFramework.RegisterTest(TTestCaseFirst.Suite); // množina testov
end.
```

(WATTS, 1999)

Časť programového kódu, z ktorého sa vykoná spustenie *GUI test runner*:

```
program Project1Test;
uses
 Forms, TestFrameWork, GUITestRunner, Project1TestCases in
'Project1TestCases.pas';
begin
 Application.Initialize;
 GUITestRunner.RunRegisteredTests; //spustenie testovacieho prostredia
end.
```

(WATTS, 1999)

1.9 Outsourcing testovania softvéru

Dnešná doba je charakteristická vývojom informatiky a informačných technológií. Vedeckí a technickí pracovníci každý deň hľadajú nové spôsoby, technológie a riešenia problémov. Informačné technológie sa zavádzajú do činnosti všetkých odvetví a sektorov ľudskej spoločnosti. Vedenie v informačných technológiách sa orientuje hlavne na vytváranie riešení, ktoré sú flexibilné, efektívne a pružné z hľadiska týchto riešení. Je potrebné vytvárať systémy s vysokou hodnotou adaptácie na čakané a nečakané zmeny požiadaviek spoločnosti. Snahou spoločností je čo najlepšie využívanie nákladov potrebných na informačno-technologické služby. Na tento účel sa využíva metóda nazývaná *outsourcing*.

Firma, ktorá využíva *outsourcing*, zmluvne spolupracuje s inou spoločnosťou zaoberajúcou sa *outsourcingom*, teda podpornými aktivitami pre iné firmy. Tieto činnosti nevykonávajú preto samotné firmy, ale zmluvné spoločnosti. Spoločnosti poskytujúce *outsourcing* disponujú špecialistami v danej problematike. Firma takto prenesie veľkú zodpovednosť a strávený čas na inú spoločnosť a môže sa naplno venovať svojej činnosti. (ORAVEC, 2010)

Outsourcing je moderným trendom aj v oblasti testovania softvéru, napriek skutočnosti, že prieskumy ukazujú zatiaľ nízku úroveň využívania tejto služby. Proces testovania by mal vykonávať špecializovaný odborník. (ORAVEC, 2010)

„Prvé testovanie pri vývoji programu (softvérovej súčiastky) vykonávajú samotní programátori. Konečné testovanie softvérovej súčiastky by však nemal vykonávať jeho tvorca a navyše, toto testovanie by sa malo vykonávať systematicky, aby sa testovaním pokryla čo najväčšia časť systému. Testovanie je pomerne zložitá oblasť. Z tejto zložitosti však vyplýva, že testovaniu by sa mal venovať špeciálny odborník, nie programátor.“ (BIELIKOVÁ, 2000, s.10)

Na základe prieskumov sa však v nasledujúcom období predpokladá zvýšenie nárastu využívania *outsourcingu*. Informačno - technické vedenia mnohých spoločností sú si vedomé previazania softvérových aplikácií a služieb, z toho dôvodu sú od kvality aplikácií požadované nemalé nároky a požiadavky. Len úplným, presným a kvalitným testovaním softvérových aplikácií je možné dosiahnuť vysokú kvalitu softvéru a tým dosiahnuť vysoké hodnotenie vlastností kvality určitej aplikácie. Absencia vlastností kvality môže mať za následok množstvo problémov spôsobených pri používaní aplikácií v prevádzke. Pri testovaní softvérových aplikácií sa stretávame so zložitými a ťažko

riešiteľnými problémami a obmedzeniami. Prvým veľkým obmedzením je nedostatok kvalitných a kvalifikovaných odborníkov v oblasti testovania aplikácií, druhým problémom je malé množstvo kvalitných zdrojov určených na účely testovania, nízka rozvinutosť procesov určených na vykonávanie testovania a v neposlednom rade nedokonalosť v presnosti merania testovania a spojitosť s vynaloženými nákladmi. (ORAVEC, 2010)

V každom prípade si je potrebné uvedomiť, že voľbou správneho odborníka v oblasti *outsourcingu* testovania je možné úplne odstrániť alebo čiastočne znížiť vyššie spomínané obmedzenia a tým ušetriť nemalé množstvo nákladov. Z tohto dôvodu je primárnym aspektom výber a využívanie testovacích služieb ponúkaných nezávislými poskytovateľmi, zavedenie metodológie testovania a taktiež aj meranie výkonnosti, ktoré nám podáva skutočný obraz a výsledky o produkte, ktorý bol dodaný. Objektívny pohľad na kvalitu aplikácie nám poskytne špecializovaný odborník, resp. poskytovateľ služieb testovania. Celý cyklus *outsourcingu* testovania je založený na niekoľkých bodoch. Najskôr klient, ktorý požaduje testovanie aplikácie určí svoje požiadavky na testovanie. Ak je poskytovateľ *outsourcingu* zároveň dodávateľom softvéru znamená to, že tento poskytovateľ pozná klientovu aplikáciu a testovanie sa tým môže výrazne zjednodušiť. Význam takéhoto vzťahu je pozitívny pre obe zainteresované strany z hľadiska zjednodušenia komunikácie. Tento spôsob so sebou prináša takisto aj určité nevýhody. Poskytovateľ služby testovania môže tým už vopred definovať stratégiu testovania tak, aby počas testovania úspešne prešla testovacími kritériami. Klientova aplikácia potom dosahuje po skončení testovania z hľadiska kvality len tú úroveň, ktorú chcel poskytovateľ pri jej vývoji dosiahnuť. Preto klient pri tomto spôsobe stráca nezávislý pohľad na kvalitu aplikácie a tým sa môže stať, že testovanie poskytne nedôveryhodné informácie a výsledky v značnej miere skreslené. (ORAVEC, 2010)

Realitou je, že drvivá väčšina projektov má oneskorenie v procese vývoja. Čas, ktorý nebol venovaný aplikácii pri vývoji sa potom dobieha práve pri testovaní. Preto sa často s cieľom napraviť predošlé nedostatky testuje pod určitým tlakom, čo prináša neúplnosť pri testovaní celkovej funkcionality aplikácie. Ak má byť *outsourcing* aplikovaný regulárne, musí byť vývoj aplikácie viditeľne oddelený od testovania. Oddelením vývoja a testovania sa oddelí zodpovednosť poskytovateľa, resp. dodávateľa a testovacieho špecialistu. Poskytovateľ softvéru sa venuje len svojej náplni práce a nie je zaťažovaný náročnými úkonmi testovania. Z technologického hľadiska je niekedy

nemožné oddelenie vývoju a testovania softvéru. Jedná sa predovšetkým o vývoj technológií špecializovaných na prácu s tajnými, citlivými a osobnými údajmi.

(ORAVEC, 2010)

Klient musí v každom prípade požadovať oddelenie vývojového a testovacieho tímu a musí byť vopred určené presné rozdelenie kompetencií, ktoré zaručia vysokú kvalitu testovania. *Outsourcing* testovania odbremeňuje stranu klienta od komplikovaného a náročného testovania na špecialistov a klient sa môže venovať vlastným činnostiam. Aby bola dosiahnutá maximálna efektívnosť a precíznosť testovania, musí sa klient sústrediť na štyri oblasti: (ORAVEC, 2010)

1. Prispôsobiť metodológiu testovania nastavením presnosti a rozsahu testovania softvéru.
2. Štruktúra testovacieho plánu, stratégie *unit testov*, *systémových testov* a *integračných testov*.
3. Súbor nástrojov z pohľadu testovacích techník (napr. odhad testovania, chýb a i.)
4. Uspôsobovanie metodológie určitým odlišnostiam počas testovania, resp. prispôsobenie skúsenostiam, ktoré sa získali počas samotného testovania. (ORAVEC, 2010)

Na úspešné riadenie *outsourcingu* je nevyhnutné správne sledovať celý testovací proces od začiatku až po koniec životného cyklu vývoja aplikácie. Výsledok úspešného nasadenia *outsourcingu* v testovaní softvérovej aplikácie je zníženie nákladov IT služieb a zvýšenie efektívnosti a kvality aplikácií v procese testovania softvérových aplikácií. (ORAVEC, 2010)

1.10 Testovanie v školstve

Testovanie vo všeobecnosti malo v minulosti a má aj dnes veľké uplatnenie. Okrem testovania náročných a rozsiahlych softvérových projektov, ktoré sme spomínali v predošlých kapitolách, môžeme testovať rôzne produkty či služby. Človek neustále niečo vytvára, testuje a využíva vo svoj prospech. Samotným používaním uplatňuje proces testovania. Testovaním zisťuje vlastnosti a kvalitatívne znaky daného predmetu. V reálnom živote sa stretávame aj s procesom overovania vedomostí a skúseností ľudí. Cieľom takéhoto testovania je zistiť špecifickým spôsobom úroveň vedomostí a znalostí. S týmto spôsobom testovania sa stretávame hlavne na školách, kde študenti a žiaci musia predložiť a preukázať vyučujúcim úroveň, akou ovládajú učivo, či prebranú látku. Záujem o štúdium je v súčasnosti vysoký a takisto sa zvyšuje aj množstvo študentov na stredných, no hlavne na vysokých školách. Neraz tak stojí pedagóg v náročnej situácii, že musí stráviť veľa času opravovaním písomiek a testov. Veľa škôl preto prešlo v poslednej dobe na počítačom podporovaný spôsob vyučovania a testovania. Má totiž viacero výhod.

„... program s kontrolnou funkciou dokáže žiaka „vyskúšať“, ale aj zhodnotiť jeho odpovede – percentami úspešnosti, grafickým vyjadrením, prípadne aj známkou a pod. Pravda, toto hodnotenie ma zástancov (objektívnosť, rýchlosť), ale aj odporcov (stráca sa interakcia učiteľ – žiak, chýba individuálny prístup).“ (PETLÁK, 2004, s. 236)

Využívanie výpočtovej techniky v školskom prostredí sa v dnes stáva už samozrejmosťou. Školy majú k dispozícii rôzne systémy a softvéry, ktoré uľahčujú vyučujúcim zvládnuť množstvo informácií, s ktorými prichádzajú každodenne do kontaktu. Tieto systémy poväčšine umožňujú aj testovanie študentov, a líšia sa cenovou dostupnosťou, obsahom, zložitosťou a inými atribútmi. Ako príklad môžeme využitie počítačov v predmete informatika. V dnešnej dobe vie s počítačom pracovať už takmer každý a preto je vhodné zahrnúť jeho používanie aj v procese vzdelávania.

„Pod tradičným nasadením počítača do výučby sa spravidla rozumie:

- *Výučba pomocou počítača (CAIDI – Computer Aided Instruction)*
- *Počítačom riadená výučba*
- *Generovanie testov*
- *Nevýukové aplikácie“* (HELD, 1999, s. 67)

Spôsob takejto výučby umožňuje študentom precvičovanie učiva a kontrolu vedomostí. Súčasné školské *LMS* systémy ponúkajú okrem rôznych *e-learningových*

kurzov zväčša len generovanie klasických elektronických testov. Študent sa najskôr prihlási do kurzu s testom, systém mu vygeneruje sadu otázok, ku ktorým má niekoľko možností na výber. Po vyplnení všetkých otázok a odoslaní odpovedí, systém sám vyhodnotí odpovede a študenta oznámkuje bodovo, či percentuálne. Tento spôsob hodnotenia je veľmi objektívny, nakoľko nikto nevie, aké otázky dostane a tiež samotné hodnotenie vykonáva systém podľa implementovaného hodnotiaceho algoritmu. Výhoda takého systému spočíva v rýchlosti akou sa spracuje množstvo informácií a v odľahčení vyučujúceho od časovo náročného opravovania. Preto je z tohto hľadiska automatizovaný spôsob testovania výhodnejší oproti manuálnemu spôsobu.

S predmetom programovanie je úzko spätý aj proces testovania, a nie len testovanie samotných študentov, ale i testovanie aplikácií. Vzniká tu oproti klasickým testom problém v tom, že zatiaľ čo pri klasických testoch mala každá otázka presne definované zadanie s jednoznačnou odpoveďou, pri programovaných aplikáciách máme presne definované len zadanie. Každý študent môže naprogramovať aplikáciu iným spôsobom, pritom všetky môžu fungovať korektne a môžu poskytovať správne výstupné hodnoty. Pri opravovaní programov musí vyučujúci manuálne každý zdrojový kód projektu otvoriť a riadok po riadku ho analyzovať a kontrolovať riadok po riadku, následne projekt spustiť a na náhodných vstupných hodnotách overiť správnosť programu. Neraz táto činnosť zaberie vyučujúcemu veľa času, ktorý by mohol využiť na iné časti výchovno – vzdelávacieho procesu. Preto by bol aj v tejto oblasti vhodný automatizovaný systém kontroly a testovania programových aplikácií. Keďže naša téma sa práve týka tejto problematiky, budeme sa v nasledujúcich kapitolách venovať téme automatickej kontroly aplikácií vytváraných študentmi na školách v predmete programovanie. Využijeme pritom teoretické poznatky získané zo súčasných trendov testovania softvérových aplikácií.

1.11 Súčasné trendy testovania na školách

Najpoužívanejší spôsob testovania na školách sú v súčasnosti didaktické testy. Ako sme spomínali, v súčasnosti sa viac preferuje automatické generovanie testov, ktoré vyberá systém z vopred pripravenej *bázy* údajov. Teória testov sa doteraz používala hlavne v papierovej forme. Elektronické testy majú veľké množstvo výhod, zabezpečujú výbornú spätnú väzbu, sú rýchle a hodnotenie študentov je objektívne. Papierová forma je účinná, no prináša so sebou aj niekoľko nevýhod, (CÁPAY, 2007)

- vzniká množstvo materiálov na uskladnenie
- zvyšuje sa časová náročnosť pri vytváraní i samotnej kontrole testov (CÁPAY, 2007)

1.11.1 Elektronické testovanie

Dnešná doba je charakteristická rozšírením osobných počítačov a celosvetovej siete internet. Z toho dôvodu sa vďaka možnostiam, ktoré poskytuje rozmohla nová metóda vzdelávania *e-learning*. Snahou spoločností a organizácií je zavádzať nový typ špecifického *CMS* systému *(Content Management System),* resp. vzdelávací systém *LMS - Learning Management System*. Za najrozšírenejší *LMS* systém by sme mohli považovať *LMS Moodle*. Zo známych *LMS* systémov môžeme spomenúť zopár:
(CÁPAY, 2007)

- Open source riešenia: *Claroline, Fle3, ILIAS, Moodle*.
- Komerčné riešenia: *MS Class Server, Eden, eAmos, eDoceo* a iné.

(CÁPAY, 2007)

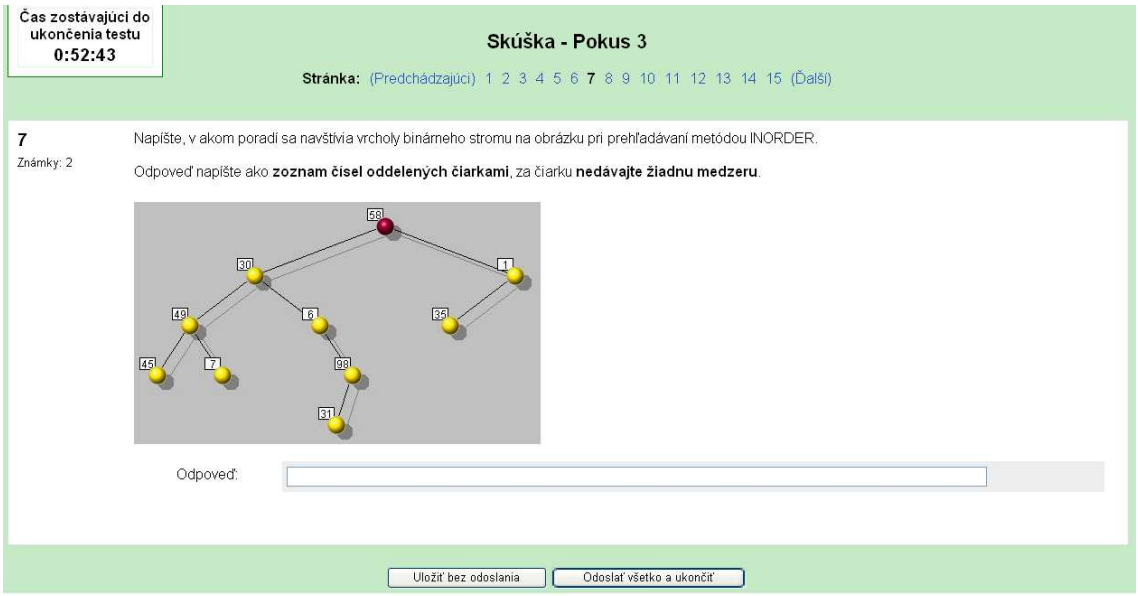
Tieto riešenia sú dostupné pre akúkoľvek organizáciu, no okrem známych systémov *LMS (CMS, LCMS)* sa stretávame aj s iným spôsobom vytvárania systémov, a to spôsob vytvorenia systému presne podľa požiadaviek univerzity, či fakulty. (CÁPAY, 2007)

1.11.2 Elektronické testovanie v LMS Moodle

LMS Moodle systém ponúka možnosti na testovanie študentov v podobe vytvárania zadaní. Tento spôsob je možné použiť na kontrolu aplikácií v predmete Programovanie spôsobom, študent odošle zadanie a pedagóg ho neskôr môže zo systému stiahnuť a analyzovaním a čítaním zdrojového kódu hodnotiť zadanie bodovo. Ďalším spôsobom

testovanie je aktivita *Test - Quiz*. *Test - Quiz* je známy tým, že umožňuje pedagógovi vytvárať sadu otázok utriedených v kategóriách. Kategórie slúžia na zaraďovanie otázok podľa zložitosti, teda tvoria určitú rovnocennú úroveň. Systém môže náhodne generovať otázky a k nim je možné náhodne generovať aj poradie odpovedí v rámci otázky. Všetky otázky sa najskôr vyhodnocujú a až po vyriešení sa odosielajú. Vstup do testu je realizovaný cez prístupové heslá alebo obmedzenie *IP* adresy. Otázky je možné vytvárať viacerými spôsobmi, napríklad otázka s výberom odpovede, otázka s krátkou slovnou odpoveďou a pod. Samozrejmosťou je vloženie obrázku ako pomôcka k otázke. (CÁPAY, 2007)

Ukážku testu v systéme *LMS Moodle* vidíme na obrázku č. 2. Ako každý systém, aj *Moodle* má niekoľko nevýhod. Zdĺhavé importovanie sady otázok do systému, pretože otázky je možné vkladať vo formulári po jednom. Systém *LMS Moodle* využíva aj Katedra informatiky, Fakulty prírodných vied, UKF (Univerzita Konštantína Filozofa) v Nitre pre predmety Programovanie, Databázové systémy a i.



Obr. 2 LMS Moodle Test – Quiz

1.11.3 Spôsoby testovania aplikácií v predmete programovanie

Aplikácie vytvorené v konkrétnom vývojovom prostredí je možné testovať len špeciálnymi softvérmi, väčšinou sú to systémy vyrábané na mieru pre určitú inštitúciu. Dá sa povedať, že aj systém *LMS Moodle* podporuje testovanie aplikácií, no nie v pravom slova zmysle, pretože v konečnom dôsledku musí pedagóg pozrieť zdrojový kód a podľa jeho korektnosti priradiť bodové hodnotenie. Na testovanie funkčnosti aplikácií v súčasnej dobe poznáme napríklad systém na kontrolu aplikácií prostredníctvom vstupno – výstupných súborov. Podobne ako náš systém Online testovanie v Delphi, aj systém vstupno – výstupných súborov musí využívať určitý spôsob na vkladanie vstupných údajov a na čítanie výstupných údajov. Na riešenie tohto spôsobu môže vzniknúť viacero možností.

1. Preddefinované formuláre, v ktorých je už vopred daný spôsob zapisovania a čítania vstupných a výstupných hodnôt aplikácie, aby sa tak skrátil čas potrebný na programovanie aplikácie. Mohlo by totiž prísť k problému, že študent správne naprogramuje úlohu, no chyba vo vstupno – výstupnej časti spôsobí nefunkčnosť celej aplikácie.
2. *Unity* na prácu so súbormi. Pracovný unit by obsahoval funkcie slúžiace na vytvorenie, naplnenie a čítanie súboru. Študent by unit použil vo svojej aplikácii a bez potrebnej znalosti práce so súbormi by mohol volať funkcie obsiahnuté v zdrojovom kóde unitu na prácu so súbormi. Úlohou unitu je napĺňať aplikáciu hodnotami cez súbor alebo zapisovanie výstupu, napríklad volaním funkcie:

```
nacitaj_vstup (var a, b, c: integer; in_subor: text);
zapis_vystup (a, b, c: integer; out_subor: text);
```

Na účel automatického napĺňania aplikácie hodnotami potrebujeme opäť systém generovania vstupných hodnôt. Ďalším problémom je spustenie výpočtu v testovanej aplikácii. Je možné riešiť tento problém tak, že najskôr sa naplní vstupný súbor hodnotami, následne sa otvorí okno aplikácie a vytvorením okna sa spustí spracovanie programovej časti. Po ukončení je okno potrebné zavrieť, a opäť opakovať cyklus testovania. Po každom zavretí prečítame výstupné hodnoty a porovnáme s očakávanými hodnotami.

2 Cieľ práce

Cieľom našej diplomovej práce je spracovať dostupné informácie a podať dôležité informácie v oblasti problematiky testovania. V teoretickej časti sa zameriame hlavne na význam pojmu testovania, uvedieme dôvody, prečo je táto fáza v životnom cykle softvérovej aplikácie tak veľmi dôležitá a rozoberieme súčasné trendy v tejto oblasti. Zároveň sa pokúsime v krátkosti oboznámiť s rôznymi typmi a spôsobmi testovania, ktoré sa bežne v praxi používajú a ku každému spôsobu budeme konštatovať jeho význam, výhody a nevýhody pri jeho používaní. V jednej časti teoretickej časti sa zameriame na automatizovaný spôsob testovania v Delphi pomocou *framework DUnit*, spomenieme jeho význam a niekoľko spôsobov využitia a v neposlednom rade sa pokúsime zhrnúť, ktorý typ testovania je najlepší, či najuniverzálnejší. Teoretická časť práce bude zameraná aj na testovanie v prostredí programovacieho jazyka Delphi a v tejto kapitole by sme chceli uviesť možnosti manuálne riadeného testovania, ktoré poskytujú a vykonávajú v súčasnosti mnohé spoločnosti a firmy zamerané na testovanie softvéru.

V praktickej časti našej práce sa pozrieme na problém testovania aplikácií vo vývojovom prostredí Delphi a pokúsime sa vytvoriť aplikáciu určenú na automatizované testovanie aplikácií vytvorených v tomto prostredí bez znalosti vnútornej štruktúry testovanej aplikácie. Budeme sa pozerať na testovanú aplikáciu ako na čiernu skrinku, pri ktorej poznáme jej správanie, to znamená že vieme predpovedať, aké hodnoty by sa mali zobraziť na výstupe, ak na vstupe bude hodnota, ktorej súvislosť s výstupom poznáme. Prácu nám uľahčí skutočnosť, že študent bude mať vopred stanovené pravidlá na programovanie aplikácie určenej na testovanie naším systémom. Tento program bude určený svojím obsahom a náplňou do školského prostredia a bude určený na testovanie vedomostí študentov z programovania v prostredí Delphi. Tým sa zúži okruh použiteľnosti pre prostredie stredných a vysokých škôl. Aby sme vyhoveli požiadavke dodržanie cieľu zadania diplomovej práce Online testovanie v predmete Programovanie, skonštruujeme aplikáciu tak, aby bola použiteľná cez rozhranie internet alebo databázového klienta v režime offline. Internetové prostredie bude prostredníkom na prenos informácií medzi našou aplikáciou a databázou umiestnenou na databázovom serveri. Pri návrhu a tvorbe programu budeme mať dispozícii na rozdiel od iných systémov určených na testovanie len jeden súbor s príponou *.EXE*.

Ako hlavný cieľ môžeme považovať zaistenie prístupu do študentovej aplikácie tak, aby sme mali kontrolu nad komponentmi, ktoré použil pri tvorbe svojej aplikácie. Kontrolou máme na mysli vytvoriť taký program, ktorý bude mať k dispozícii funkcie, ktoré dokážu zistiť triedy a počet hľadaných typov komponentov. Ak zistíme triedy a počet hlavných komponentov, ktorých absenciu alebo zmenu typu či počtu považujeme za nesplnenie požiadaviek pre testovanie, pokúsime sa nadviazať obojsmernú komunikáciu medzi komponentmi a našou aplikáciou. Obojsmerná komunikácia v našom prípade znamená, že pomocou nejakej funkcie dokážeme napríklad komponenty typu *TEdit* napĺňať nami zvolenými hodnotami, zároveň požadujeme vytvorenie udalosti kliknutia v definovanom okamihu na zvolené tlačidlo a následné čítanie vlastnosti *Caption* alebo text z komponentu a prenos tejto informácie do testovacieho programu. Cieľom je nájsť spôsob, akým je možné tento proces zrealizovať. Prostredie operačného systému Windows nám ponúka niekoľko spôsobov, veľmi záleží na kreativite, no my si zvolíme spôsob pomocou identifikátorov okien, ktoré sú vo Windows vytvorené a ku ktorým má systém pridelené jednoznačné a jedinečné číslo. Ak dokážeme túto komunikáciu uskutočniť, získané informácie vieme využiť na zistenie, či aplikácia, ktorú študent vytvoril spĺňa požiadavky, ktoré boli od študenta požadované. Získané údaje chceme spracovať a v zrozumiteľnej forme sa zapísať do vopred vytvorených tabuliek v databáze, odkiaľ ich bude možné čítať, zapisovať, editovať a pod. V neposlednom rade cieľom je dodržanie všetkých podmienok a faktorov, ktoré ovplyvňujú celkový dojem a kvalitu vytvorenej aplikácie.

3 Metodika

V našej práci sa pokúsime navrhnúť fungujúci automatizovaný testovací systém pre vnútorné vzdelávacie potreby stredných a vysokých škôl. Vývoj aplikácie budeme vykonávať na **osobnom počítači** s originálnym operačným systémom **Windows XP od spoločnosti Microsoft**, a zároveň bude obsahovať potrebné softvérové vybavenie na tvorbu aplikácie. Pri vývoji programu použijeme prostredie programovacieho jazyka **Borland Delphi Enterprise 7**. Na vytvorenie databázových schém a diagramov použijeme objektový program **DIA, číslo verzie 0.95-1**, určený na kreslenie a návrh rôznych plánov, diagramov a schém. Na kreslenie použijeme grafické prostredie voľne šíriteľného softvéru **GIMP vo verzii 2.2** s grafickým toolkitom **GTK+, verzia 2.8.18**, ktorý slúži na vytváranie grafických užívateľských rozhraní. Pre prácu s databázou použijeme softvér **XAMPP vo verzii 1.6.6** vyvíjaný firmou Apache Friends, ktorý nám ponúkne aplikačný balík pre *webserver* (*MySQL*, *phpMyAdmin*, *Apache* a i.). V prostredí Delphi nám pripojenie k databáze poskytne súbor inštalovaných voľne šíriteľných komponentov *MySQL***, MyComponents**.

Navrhovaný systém by mal svojou funkcionalitou pokrývať potreby predmetu Programovanie vo vývojovom prostredí Delphi v odbore Informatika. Predpokladom pre úspešné vytvorenie je podrobné štúdium a získanie všetkých potrebných informácií o danej problematike. Pomocou internetových odkazov a knižných publikácií zosumarizujeme doposiaľ známe metódy testovania a pokúsime sa o vytvorenie vlastného testovacieho programu.

Celú aplikáciu rozdelíme do troch hlavných častí. Každá z týchto častí bude tvorená formulárom. Formuláre budú podľa potreby navzájom spolupracovať a komunikovať pri súčasnej výmene potrebných údajov. Prvou časťou bude uvítací formulár určený na výber prihlásenia študenta, registráciu nového užívateľa alebo prihlásenie vyučujúceho. Druhú časť bude tvoriť formulár, ktorý poskytne priestor pre realizáciu študenta, resp. miesto zobrazenia vylosovaných úloh a miesto pre odosielanie vykonaných zadaní. Zároveň bude obsahovať informačné položky pre študenta, kde ho systém po odovzdaní a vyhodnotení úloh oboznámi s výsledkami testovania číselnými údajmi. Tretí formulár je určený výhradne pre používanie vyučujúcim a bude obsahovať informácie týkajúce sa výsledkov študentov z testov a zároveň tento formulár poskytne funkciu vytvorenia dočasného hesla potrebného pre vykonanie testu. Kvôli prehľadnosti a jednoduchosti zvolíme tabuľkový spôsob prezentovania údajov.

Úlohou automatizovaného systému je kontrolovať správnosť a funkčnosť naprogramovaných aplikácií. Testovací systém bude aplikácia, ktorá prostredníctvom databázového klienta bude pristupovať k databáze umiestenej na webovom serveri, spočiatku to bude náš počítač, kde bude umiestnená aj testovacia aplikácia. Počítač, resp. server použijeme na ukladanie informácií vytvorených v našej aplikácii. Databáza je miestom, v ktorom budú uložené informácie o jednotlivých študentoch, ich používateľské mená, priezviská, čísla vylosovaných otázok, počty získaných bodov z jednotlivých testov, štatistické hodnoty z vykonaných predošlých testov a pod. Ďalej by mal program obsahovať informácie o jednotlivých testovacích úlohách, počet bodov za danú úlohu a jednotlivé zadania. Aplikácia by mala byť schopná po prihlásení študenta poskytnúť po vyžiadaní študenta vylosovanie zadaní, ktoré študent vo svojom záujme vypracuje a každú z nich samostatne pripne na určené miesto vo formulári a odošle vo forme jediného súboru s koncovou príponou *.EXE* do systému, ktorý ich automaticky vyhodnotí. Snažíme sa o vytvorenie aplikácie, ktorá bude nielen programovo dobre zvládnutá, ale bude pre študenta, či vyučujúceho svojím prostredím používateľsky príjemná. Predpokladom je zladenie grafickej úpravy a intuitívnosti ovládania. Naša „*user friendly*“ aplikácia by taktiež pri normálnom zaobchádzaní nemala spôsobovať používateľovi problémy pri manipulácii alebo hlásiť neočakávané chybové hlásenia. Avšak hlavným objektom skúmania sa v našom prípade stáva samotná aplikácia študenta, konkrétne všetky súbory s príponou *.EXE* pre jednotlivé zadania, ktoré stihol študent počas testu vypracovať.

Aplikácie študenta budú uložené počas testu na disku a jediný spôsob akým sa naša aplikácia dostane k vypracovaným zadaniam je absolútna cesta k súboru. Problém je, že sa pozeráme na študentovu aplikáciu ako na čiernu skrinku, v ktorej nevidíme jej vnútornú programovú štruktúru, no vieme, aké výstupné hodnoty na základe zadaných vstupných hodnôt majú byť vo výpise výstupných hodnôt. Zvolíme systém kontroly tak, aby bolo možné nejakým spôsobom pristupovať automaticky prostredníctvom funkcií našej aplikácie do formulára študentskej aplikácie a pokúsiť sa pre začiatok pomocou špeciálneho algoritmu nájsť okná v cudzom formulári a pohybovať sa v oknách tohto formulára. To znamená, že musíme zvládnuť prepojenie dvoch aplikácií a mať dokonale zvládnuté manipulovanie s údajmi pri prepojení týchto aplikácií. Pretože predpokladáme prácu pod systémom spoločnosti Microsoft, pre vyriešenie tohto problému je potrebné štúdium Windows *API* funkcií, ktoré poskytujú funkciu na prácu s oknami v operačnom systéme Windows. Výhodou je, že programové prostredie

Delphi veľmi dobre spolupracuje s prostredím Windows prostredníctvom funkcií. Zoznam API funkcií získame z internetovej stránky spoločnosti Microsoft.

Po vyriešení daného problému sa dostaneme k informáciám inej aplikácie a tým k jej formuláru, bude potrebné ďalej zistiť počet a typy týchto dcérskych okien rodičovského formulára a správne tieto okná rozlíšiť a zatriediť do kategórii podľa triedy objektu, ktoré z nich slúžia na vstup, a ktoré na zapisovanie výstupných hodnôt. Po vyriešení základného problému musíme vyriešiť akým spôsobom a akými hodnotami budeme napĺňať nájdené okná v cudzom formulári. Ideálny sa zdá byť spôsob napĺňania náhodnými hodnotami v určitom intervale hodnôt, ktoré sú pre dané zadania charakteristické. To znamená, že napríklad pre zadanie, v ktorom sa počíta *faktoriál* zadaného celého čísla, budeme uvažovať o vstupných hodnotách napríklad od 1 po 1000. Pre vyššie hodnoty vstupných údajov by testovanie bolo časovo a z hľadiska prostriedkov zbytočné. Takto spracujeme možné vstupy pre každé zadanie. Po vložení vstupných údajov musíme kliknúť programovo na tlačidlo druhej aplikácie aby sme spustili výpočet. Ako posledný bod bude potrebný zber výstupných údajov zo študentovej aplikácie a následné porovnanie týchto skutočných výsledkov a očakávanými. Na základe porovnania sa zapíšu do databázy hodnotenia študenta.

Čo sa týka údajov uložených v databáze, pri riešení a navrhovaní systému bude naša aplikácia komunikovať s databázou spočiatku len na webovom serveri *localhost* cez *Apache*, umiestnenom na našom počítači. Na webový server bude možné ju umiestniť po ukončení celej praktickej časti práce. V neposlednom rade musíme vyriešiť spoľahlivosť a bezpečnosť aplikácie tak, aby nebolo možné bežným spôsobom sa dostať k citlivým informáciám, či podvrhnúť systému hodnotenia.

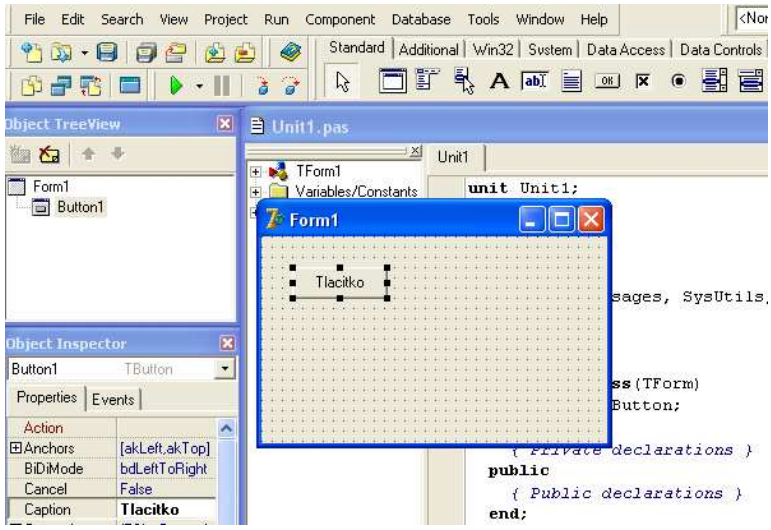
4 Vlastná práca

4.1 Použitý software Borland Delphi

Pri tvorbe praktickej časti diplomovej práce sme použili prostredie vývojového prostredia Borland Delphi. Výber tohto programovacieho jazyka pre naše účely podnietilo viacero dôvodov. Delphi je komplexný vizuálny programovací jazyk pre prostredie operačného systému Windows. Programátorské prostredie *IDE* označuje, že programátor má možnosť v jednom komplexnom balíku vyvíjať, upravovať, kompilovať, testovať, či ladiť programy. Hlavné prednosti Delphi sa ukrývajú v podstate jeho vlastnosti, „vizuálny“. Aplikácie, ktoré by sme v iných jazykoch vytvárali pomerne dlhý čas, zvládneme v prostredí Delphi v reálne krátkom čase.

(KADLEC, 2001)

Programovanie pod operačným systémom Windows nám taktiež uľahčuje riešenie viacero problémov tým, že veľa vecí za nás zvláda systém. Jednoduchosť návrhovej fázy a tvorba vizuálneho prostredia aplikácie poskytuje programátorovi viac času sústrediť sa na vlastnú *algoritmizáciu*. Vzhľad aplikácie vytvárame pomocou výberu a ukladania mnohých komponentov na formulár, ktorý vidíme na obrázku č. 3.



Obr. 3 Vývojové prostredie Delphi

Programátor následne už len programuje správanie sa týchto komponentov a grafická stavba zabezpečuje správne fungovanie aplikácie. Ako komponenty najčastejšie používame textové komponenty, tlačidlá, nápisy a pod. S každým komponentom môžeme manipulovať prostredníctvom nástroja *Object inspector*. Ku každému komponentu môžeme pristupovať z dvoch rôznych hľadísk. Z hľadiska jeho

vlastností môžeme meniť jeho veľkosť, farbu, font a mnoho ďalších vlastností. Pretože programovanie v Delphi je udalosťami riadené, je pre nás z hľadiska udalostí zaujímavá udalosť, ktorá nastane napríklad po jej stlačení, pustení tlačidla a pod. Reakciu na danú udalosť si môžeme sami naprogramovať, a tým nám je umožnená voľnosť pri programovaní a záleží len na našej fantázii, ako vynaložíme s dostupnými príkazmi.

4.2 Prečo sme sa rozhodli pre Delphi?

V dobe, keď vzniklo Delphi, si programátor mohol vybrať z množstva vývojových nástrojov. V dnešnej dobe je situácia podobná, ale s tým rozdielom, že týchto nástrojov máme na výber mnohonásobne viac. Preto, ak chceme použiť pri vytváraní serióznej aplikácie prostredie Delphi, je potrebné uviesť viacero dôvodov na jeho použitie. S ohľadom na naše potreby zhodnotíme zopár kladných stránok, ktoré nám poskytuje Delphi:

- firma Borland je významným výrobcom osvedčených kompilátorov pre jazyky Pascal a C
- umožňuje vytvárať návrh aplikácií s využitím vlastností operačného systému
- podpora Objektovo orientovaného prístupu
- podporuje používanie databáz, čo je pri súčasných požiadavkách na systémy nezanedbateľná vlastnosť
- významne sa podieľa na podpore internetu pomocou komponent
- umožňuje vytváranie vlastných komponent
- výhodou je programovanie v jazyku objektový Pascal, ktorý je považovaný v informatickej praxi za akýsi „základ“ programovacích jazykov
- Delphi je vhodným jazykom pre začiatočníkov, ktorý neskôr môžu prejsť na iné procedurálne programovacie jazyky
- Delphi obsahuje veľmi účinné ladiace prostriedky
- jazyk a syntax sú natoľko jednoduché a jednoznačné, že následný prechod na iný programovací jazyk je zväčša príjemnou záležitosťou

  (MACHOVÁ, 2004)

Tieto dôvody nás viedli v rozhodovacom procese pri výbere programovacieho jazyka. Vzhľadom k náplni našej práce a skutočnosti, že na stredných školách sa programuje zväčša v tomto jazyku, bude najlepšie ak nebudeme kombinovať viacero programovacích prostredí.

4.3 Softvérová a hardvérová konfigurácia

Po dôkladnom štúdiu potrebnej dostupnej literatúry sme vytvorili aplikáciu vo vývojovom prostredí Delphi Enterprise 7, ktorá je určené predovšetkým pre prácu pod operačným systémom Windows. Všetky vývojové, ladiace, testovacie grafické návrhy boli riešené na notebooku Sony Vaio s operačným systémom Microsoft Windows XP Professional, verzia 2002, Service Pack 3. Odporúčame preto používať nami vytvorené aplikačné prostredie taktiež len v kombinácii so systémami od spoločnosti Microsoft, verzie Windows 98 a novšie, aby neprichádzalo ku kolíziám programu, spôsobených nekompatibilitou, či nesprávnym softvérovým vybavením. V dnešnej dobe odporúčame používať systém Windows XP, Vista alebo Windows 7.

Softvérové vybavenie je v ideálnom prípade kombinovať s dostatočnými hardvérovými prostriedkami. Ako sme uviedli v metodike práce, pre vytvorenie našej aplikácie budeme potrebovať Borland Delphi 7, knižnicu komponentov *MySQL*, *XAMPP* spoločnosti *Apache Friends*, softvér *DIA*, *GIMP* a ostatné programy podľa potreby.

Príliš slabé hardvérové vybavenie môže spôsobiť problémy pri používaní aplikácie, mrznutie pri zložitých výpočtoch, či čiastočnú alebo úplnú stratu informácií. Nakoľko aplikácia využíva niekoľko dôležitých Windows *API* funkcií a je prepojená s databázovým systémom, je potrebné aby hardvérové vybavenie spĺňalo aspoň minimálne požiadavky, v opačnom prípade nie je zaručené, že z časového hľadiska budú vybavené všetky požiadavky kladené na program. Aplikačné prostredie automatizovaného testovacieho systému bolo vytvorené pri nasledovných hardvérových komponentoch. Podotýkam, že popisované hardvérové vybavenie považujem za určitý stredný až stredne nižší štandard, ktorý v dnešnej dobe je optimálnym riešením. Školské učebne sú dnes vybavené výkonnými zariadeniami, ktoré pri spracovaní požadovaných dát nebudú mať žiadne problémy.

Použitá hardvérová konfigurácia:

- Procesor CPU: Intel Mobile Core 2 Duo T5500
- Frekvencia CPU: 1,66 GHz
- Pamäť RAM: 1GB
- Grafický čip: Nvidia GeForce 7400Go s 256MB vlastnej pamäte

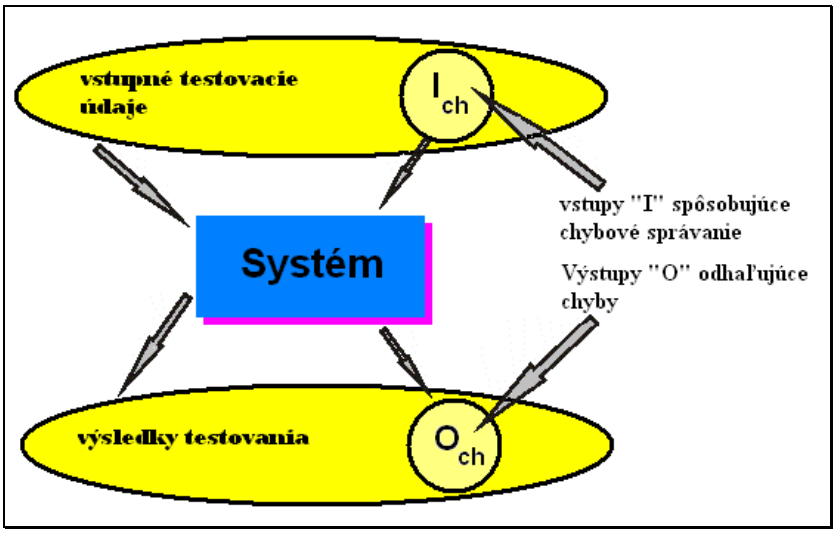
Aplikáciu je možné používať aj pri slabších hardvérových konfiguráciách, môžu sa však vyskytnúť určité časové oneskorenia pri vykonávaní algoritmov spracovávajúcich zložitejšie údaje.

4.4 Výber testovacej metódy

Po naštudovaní materiálov z problematiky testovania sme sa rozhodovali, aký druh a spôsob testovacej metódy použiť. Vieme, že budeme pracovať s *EXE* súborom a nebudeme mať prístup k zdrojovému kódu ani k vnútornej štruktúre aplikácie. Preto po zohľadnení všetkých kladných a záporných vlastností jednotlivých testovacích techník použijeme nasledovný spôsob:

- *dynamický spôsob testovania*: testovanie aplikácie bude prebiehať počas jej behu, základom je vytvorenie vhodnej množiny testovacích vstupov
- *metóda čiernej skrinky* – neberieme ohľad na zdrojový kód spustenej aplikácie
- *funkcionálne testovanie* – budeme zohľadňovať vzťah vstup – výstup aplikácie

Bude nás zaujímať tá množina vstupných hodnôt, ktorých pravdepodobnosť nájdenia chýb vo výstupných hodnotách je vysoká. Princíp testovania je zobrazený na obrázku č. 4.



Obr. 4 Princíp testovania

Pri *funkcionálnom* testovaní je veľmi dôležitý faktor výberu vstupných testovacích hodnôt. Pretože nemôžeme otestovať všetky možné vstupy, musíme dbať na správny výber každej vstupnej hodnoty. Ako príklad si uvedieme časť zdrojového kódu programu, ktorý násobí dve čísla:

```
program nasob;
res:= x*y;
  if res = 100 then
     res:=0
  else vypis(res);
```

V takomto prípade je len veľmi malá šanca, že naším vstupom budú práve hodnoty, ktorých výsledok násobenia bude číslo 100. K tejto hodnote by sme mohli prísť skôr náhodne alebo otestovaním všetkých možných vstupov. V našom prípade sa budeme snažiť otestovať čo najviac možných vstupov, aby sme predišli podobnej situácii.

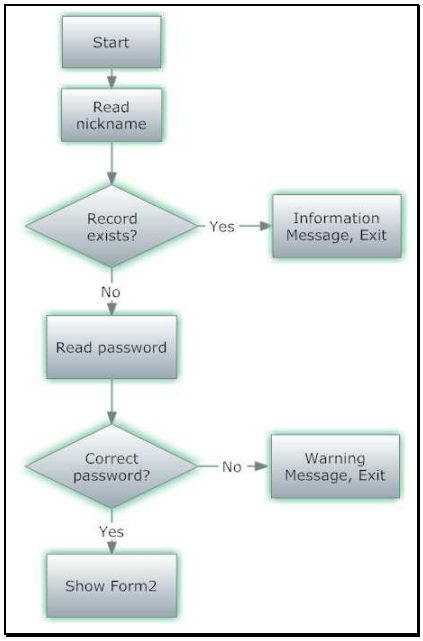
4.5 Popis prostredia testovacieho softvéru

Naša aplikácia s príjemným používateľským prostredím nie je z vizuálneho hľadiska výnimočná žiadnymi neštandardnými prvkami, nakoľko má vzhľad štandardného okna vo Windows. K používaniu programu nie je potrebná žiadna špeciálna inštalácia, je možné ju umiestniť kdekoľvek na pevný disk a otvoriť. Všetky ostatné náležitosti vykoná aplikácia automaticky za nás a následne je možná práca s ňou. Vonkajšiu štruktúru aplikácie tvorí niekoľko formulárov, z ktorých je v jednom momente možné používať len aktuálne používaný, kvôli snahe predísť zbytočným chybám pri neočakávanom chovaní sa používateľa. Po otvorení sa nám zobrazí úvodný formulár zobrazený na obrázku č. 5, ktorý ponúka užívateľovi tri možnosti výberu udalosti v hornej časti formuláru.

Obr. 5 Formulár spustenej aplikácie

Úlohou uvítacieho formuláru je predísť neoprávnenému prístupu do aplikácie a zároveň zvýšiť bezpečnosť pred spracovaním údajov. Prvý výber udalosti „*Prihlásiť*

(už som registrovaný)“ je dostupný len v prípade, že užívateľ systému je už registrovaný, to znamená, že jeho jedinečné užívateľské meno, *nickname*, ešte nefiguruje v databáze mien registrovaných užívateľov. Prihlasovací algoritmus je veľmi jednoduchý a postačuje potrebám pre daný účel používania. Po zakliknutí prvej položky v zozname a vyplnení sa vykoná nasledovná schéma príkazov, ktorá je zobrazená formou vývojového diagramu na obrázku č. 6.



Obr. 6 Vývojový diagram prihlasovania do aplikácie

Ak zadá užívateľ svoj *nickname* a odošle správu aplikácii, funkcia, ktorá skontroluje výskyt v databáze, vráti hodnotu 0 alebo 1 podľa toho, či už daný záznam existuje. Ak existuje, aplikácia podá hlásenie a zobrazí sa na formulári nové okno, kde je potrebné zadať vstupné heslo do testu ako je zobrazené na obrázku č. 7.

Obr. 7 Formulár pre overenie hesla

Ak je heslo správne, otvorí sa ďalší formulár, kde prebieha samotné testovanie a zobrazovanie zadaní a ich výsledkov testu. Kontrola na základe hesla je konštruovaná

tak, že v databáze v tabuľke heslá sú uložené dočasné heslá pre vstup do testu. Vyučujúci pred a po ukončení každého testu zmení obsah dočasného hesla, aby zabránil prípadnému prístupu študenta do aplikácie. Svojím spôsobom, každý študent má svoje prihlasovacie údaje, ktoré umožňujú práve jemu prístup k svojim údajom. Dočasné heslá sú známe aj z iných systémov. My sme sa inšpirovali známym systémom *LMS Moodle*, kde pred každým vstupom do testu vyučujúci prístupové heslo zmení.

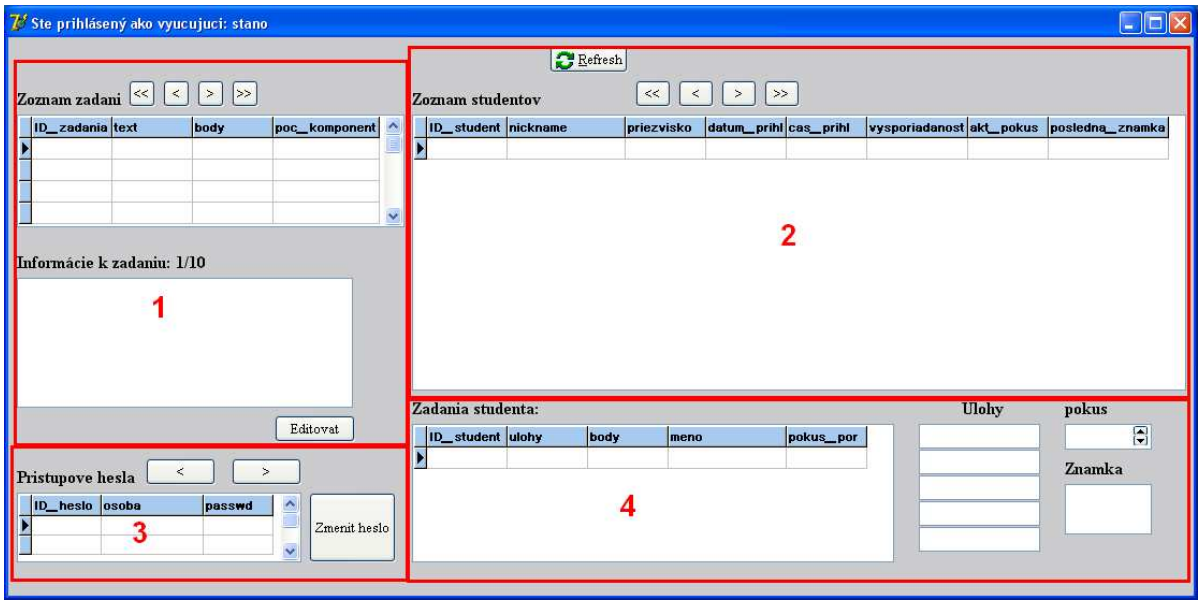
Ak užívateľ prvýkrát chce získať prístup do aplikácie, je potrebná jeho registrácia. Na tento účel slúži v poradí druhá položka v skupine *RadioGroup „Registrovať (som tu prvýkrát)*“. Používateľ zadá svoj *nickname* a *priezvisko*. Z týchto údajov musí byť len *nickname* spomedzi záznamov v databáze jedinečný. Môže sa totiž stať, že dvaja alebo viacerí užívatelia budú mať rovnaké priezvisko. Po vyplnení potrebných údajov na registráciu program skontroluje, či zadané užívateľské meno nie je používané. Ak nie je, zapíše sa do databázy ako nový člen v tabuľke.

Veľmi dôležitým prvkom v aplikácii je *Mod Profesor*. Tento mód je prostredníkom výmeny informácií medzi študentom a vyučujúcim. Po ukončení testu, či vykonaní dôležitej zmeny v systéme sa tieto údaje musia preniesť do databázy, aby sa nestratili žiadne informácie. Vyučujúci má jedno heslo, ktoré môže postupom času zmeniť, aby nezískal nikto neoprávnený prístup do jeho módu. Po zadaní správneho hesla sa otvorí formulár, kde má vyučujúci možnosť prezerať bodové i percentuálne výsledky a úspešnosť všetkých študentov, pozmeniť textové časti zadania, meniť dočasné prístupové heslá študentov, či zmeniť heslo pre prístup vyučujúceho. Prostredie formulára ako je vidieť na obrázku č. 8, je prehľadné a poskytuje základné funkcie potrebné pri testovaní študentov. Z obsahového hľadiska by sme mohli rozdeliť formulár do štyroch oblastí.

- **Oblasť 1:** Oblasť 1 je tvorená tabuľkou so zadaniami, ktoré je možné editovať a kliknutím sa presúvať na požadovanú položku v tabuľke, pričom pri výbere sa zobrazia informácie o zadaní v textovom poli pod tabuľkou.
- **Oblasť 2:** Oblasť 2 je tvorená tabuľkou, ktorá obsahuje zoznam všetkých študentov v databáze. V zozname vidíme vždy len posledné hodnotenie študenta**.**
- **Oblasť 3:** Oblasť 3 obsahuje zoznam prístupových hesiel pre prístup študentov a vyučujúceho. K dispozícii je zmena hesla tlačidlom „*Zmeniť heslo*“.

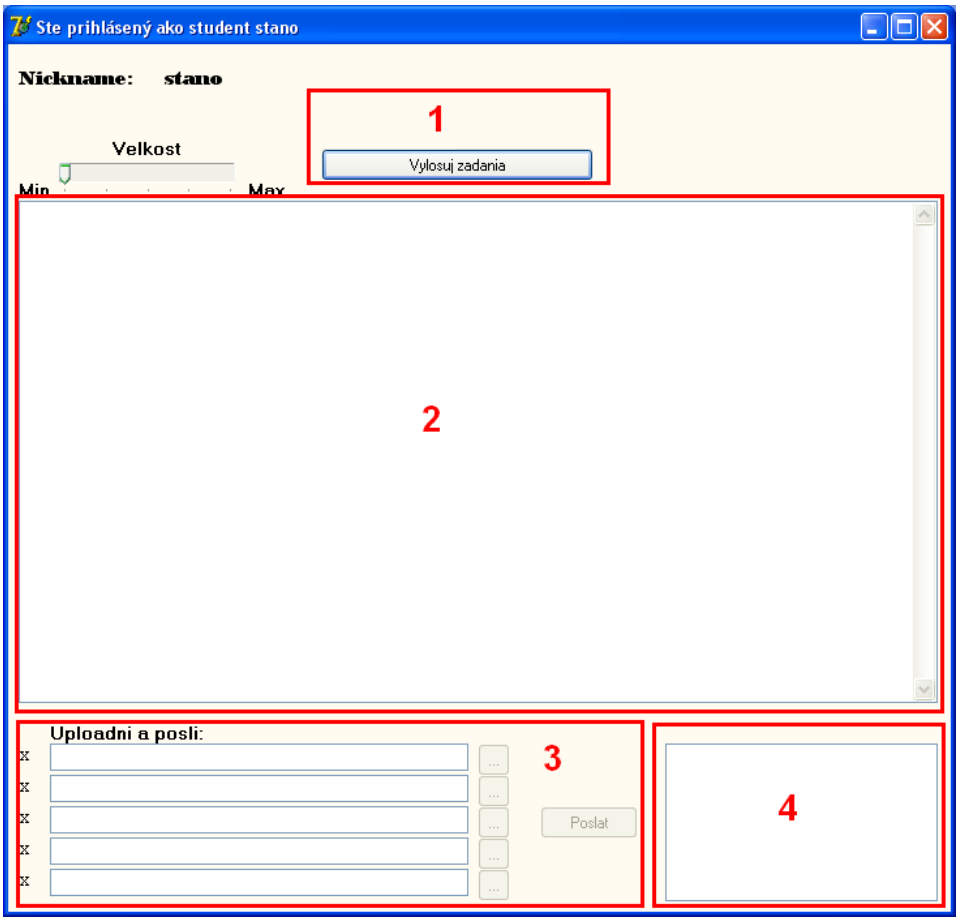
Informácia o zmene hesla sa v okamžiku aktualizuje a následné prihlásenie môže prebehnúť korektne len so zmeneným heslom.

- **Oblasť 4:** Oblasť 4 je úzko spätá s oblasťou 2. Po kliknutí na akýkoľvek záznam v tabuľke v oblasti 2 sa zobrazia všetky dostupné informácie o aktuálne vybranom študentovi v tabuľke pod ňou vzostupne podľa poradia vykonaného zadania. Tento zoznam umožňuje prezerať všetky doposiaľ riešené zadania študenta s bodovým i percentuálnym hodnotením. Napravo od tabuľky meníme poradové číslo vykonaného testu, ku ktorému sa nám prepočíta percentuálny zisk z daného pokusu.



Obr. 8 Formulár Mod Profesor

Posledným a zároveň najobsiahlejším bodom praktickej časti je formulár zobrazujúci sa po prihlásení študenta. Formulár obsahuje prvky, ktoré pôsobia pre užívateľa jednoducho a ich ovládanie sa tak stáva intuitívne. Implementuje ovládacie prvky, resp. komponenty, ktoré slúžia na vykonanie potrebného programového kódu. Po aktivácii a otvorení formulára je v hornej časti vidieť aktuálne prihláseného študenta. Okno je možné zavrieť bežným spôsobom, použitím štandardného Windows tlačidla v pravom hornom rohu. Na obrázku č. 9 vidíme formulár určený na samotný proces testovania študentov.



Obr. 9 Formulár testovania študentov

Formulár testovania ako vidíme na obrázku č. 9, je rozčlenený do štyroch oblastí:

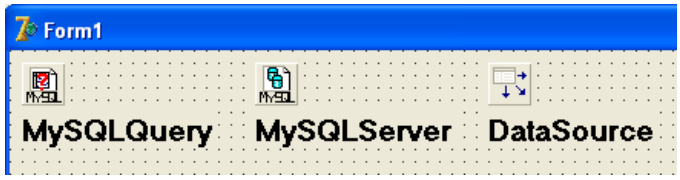
- **Oblasť 1:** Tlačidlo slúži na generovanie zadaní pre prihláseného študenta. Ak už si študent pri predchádzajúcom prihlásení vylosoval otázky a aplikáciu z nejakého dôvodu zavrel alebo prišlo k neočakávanej chybe, pre ktorú bolo nutné ukončenie aplikácie, táto skutočnosť sa zaznamená v databáze, aby bolo možné pri ďalšom prihlásení upozorniť správou o nevysporiadaní sa so systémom a vrátiť pôvodne nevypracované zadania.
- **Oblasť 2:** V tejto oblasti sa študentovi zobrazí textová podoba očíslovaných vylosovaných zadaní. Ku každej otázke vyučujúci presne nadefinuje znenie úlohy. K dispozícii je samozrejme aj možnosť zväčšiť alebo zmenšiť veľkosť písma textov zadaní pre študentov so zníženými zrakovými schopnosťami.
- **Oblasť 3:** Keď študent vypracuje pre neho určené zadania, odošle ich v jednoznačne vyznačenej zóne. V každom riadku je uvedené, ktorá úloha patrí ku ktorému riadku pomocou očíslovania zadaní. Študentovi sa vždy vylosuje 5

zadaní. Tento počet je pevne nastavený v zdrojovom kóde a je možné ho prípadne zmeniť jednoduchým zásahom v kóde. Odosielajú sa vždy len vyladené a podľa stanovených pravidiel naprogramované úlohy vo výstupnom formáte jeden súbor s príponou *.EXE*. Po vložení všetkých dostupných *EXE* súborov študent stlačí tlačidlo „*Poslať*“ a súbory sa začnú vyhodnocovať.

- **Oblasť 4:** Po vyhodnotení všetkých úloh sa výsledky zobrazia v malom okne, oblasť 4. Výsledkom je bodové hodnotenie každej úlohy, celkový počet získaných bodov a v neposlednom rade i percentuálna úspešnosť študenta. V momente zobrazenia výsledkov sú už tieto údaje zapísané v databáze a sú k dispozícii pre vyučujúceho. Študent nie je povinný vypracovať a odoslať všetky vygenerované zadania. Vždy odosiela len zadania, ktoré stihol vypracovať, nefunkčné a neodladené *EXE* súbory ostatných projektov neposiela na kontrolu.

4.6 Programová štruktúra aplikácie

Aby sme dokázali pracovať prostredníctvom aplikácie s databázou, vytvorili sme si *Win32* klienta *MySQL* v Delphi. Požiadavky užívateľov budú prichádzať použitím prostriedkov jazyka *SQL*. Využili sme pritom webový server na lokálnom počítači a databázový systém *MySQL*. Zväčša sa *MySQL* inštaluje ako súčasť balíka, preto sme ako alternatívu zvolili veľmi známy balík *XAMPP*. Stačí ho rozbaliť do cieľového adresára bez inštalácie. Aby sme urýchlili proces vytvárania databázy a tabuliek, využívali sme pritom prostredie *phpmyadmin*, ku ktorému sa dostaneme pomocou adresy *http://localhost/phpmyadmin/* v prehliadači. Ďalším potrebným prvkom v prostredí Delphi boli komponenty na pripojenie k *MySQL*, ktorých zoznam vidíme na obrázku č. 10, *MySQLQuery, MySQLServer a DataSource*. (DOCSTOC, 2010)



Obr. 10 Komponenty balíka MyComponents

Na zobrazovanie údajov v tabuľkách sme použili komponent *DBGrid*. Pomocou *DBGrid* načítame údaje z jednej tabuľky, no spomínaný komponent *MySQLQuery* umožňuje čítanie údajov z viacerých tabuliek a riešenie čítania je cez príkazy jazyka

SQL, napríklad komponent *MySQLQuery* sme použili pri zapisovaní informácií o novom užívateľovi do databázy pri registrácii.

Časť zdrojového kódu, ktorý demonštruje zápis nového užívateľa do databázy:

```
procedure zapis_meno(nickname, priezvisko: string);
var q: TMySQLQuery;
begin
q:= TMySQLQuery.Create(nil);
q.Server:= MySQLServer1;
q.SQL.Clear;
q.SQL.Add ('INSERT INTO studenti ( nickname, priezvisko, datum_prihl,
cas_prihl,    vysporiadanost,    akt_pokus,posledna_znamka)    VALUES
("'+nickname+'",     "'+priezvisko+'",     "'+datum+'",     "'+cas+'",
"'+'ano'+'", "1","0")');
q.ExecSQL;
q.free;
end;
```

Server *MySQL* je inštalovaný na lokálnom počítači, preto v našom prípade budú spracovávané požiadavky od jedného používateľa. Adresa pre *MySQLServer* má v našej aplikácii adresu *localhost*, to znamená *127.0.0.1*. Pripojenie sa však nadväzuje a uskutočňuje rovnako, akoby bol databázový systém na inom počítači. Otázka bezpečnosti v databázovom systému zatiaľ nie je vyriešená, nakoľko je z dôvodu offline používania vytvorený len účet s *login root*. Našou snahou do budúcnosti je vylepšenie aplikácie a jej aplikovanie pre skupiny užívateľov, preto z dôvodu serióznosti bude potrebné bezpečnostnú chybu odstrániť. Hodnota *LoginPrompt* je defaultne nastavená na hodnotu *false*, to znamená, že nie je nutné zadávať heslo pri prístupe k informáciám cez sieť. (DOCSTOC, 2010)

Výber webového servera v súčasnosti nie je takmer žiadny problém. K dispozícii máme dostupných množstvo *free* použiteľných serverov, môže sa ním v skutočnosti stať aj náš počítač doma, alebo v škole, ktorý v čase používania aplikácie študenta alebo vyučujúceho bude pripojený k sieti internet a konfigurácia s príslušnou databázou bude korektná.

4.7 Programové funkcie v aplikácii na prácu s DBS

Pre lepšie pochopenie štruktúry aplikácie uvádzame niekoľko často používaných podprogramov s krátkym vysvetlením ich funkcie.

```
function zaznam_existuje(uzivatel: string): boolean;
```

- Funkcia zistí prostredníctvom zadaného užívateľského mena, či existuje v databáze rovnaký záznam.

```
procedure zapis_meno(nickname, priezvisko: string);
```

- Procedúra, ktorá zapíše korektné registračné údaje o užívateľovi do databázy.

```
function overenie_hesla(ID: integer; heslo: string): boolean;
```

- Funkcia overujúca vstup užívateľa so zadaným heslom a zároveň zisťuje, či zadávateľ hesla je existujúci užívateľ.

```
function kontrola_rovnakych(a, poc: integer): boolean;
```

- Funkcia kontroluje pri generovaní zadaní, či nebol vyžrebovaný záznam nie je duplicitný. Ak bol, losuje sa zadanie ešte raz.

```
procedure zmen_na_vysporiadanost(ID_uzivatel, na_ktory_pokus: integer;
last_mark: real);
```

- Procedúra slúžiaca na zmenu stavu užívateľa po odoslaní zadaní, z nevysporiadanosti na vysporiadanosť. Vysporiadaný je študent, ktorý sa práve registroval alebo odovzdal zadania.

```
function zistenie_ID_studenta(nick: string): integer;
```

- Funkcia zistí identifikačné číslo užívateľa na základe jeho užívateľskoho mena nickname.

```
function zadanie(poradie: integer): string;
```

- Funkcia vráti a vypíše text zadania podľa poradového čísla vyžrebovanej úlohy zadania.

```
function ci_je_vysporiadany(ID: integer): boolean;
```

- Funkcia zisťujúca, či je užívateľ s daným ID číslom vysporiadaný so systémom.

```
procedure nevysporiadane_ulohy(ID: integer);
```

- Procedúra na základe ID užívateľa zistí zadania, ktoré úlohy boli užívateľovi vygenerované, ale z nejakého dôvodu neboli vypracované a odoslané na kontrolu.

```
procedure vypis_nevysporiadanych;
```

- Procedúra vypisujúca nevysporiadané úlohy zistené v procedúre nevysporiadane_ulohy.

```
function zisti_pokus(ID: integer): integer;
```

- Funkcia z parametra ID užívateľa zistí, koľko pokusov už v testovaní realizoval pod jeho identifikačným číslom.

```
function zistenie_poctu_komponentov(oznacenie: integer): integer;
```

- Funkcia z poradového čísla zadania zistí, koľko komponentov je potrebné použiť v aplikácii, ktorú študent vytvára. Dôležitá podmienka pre úspešný výsledok testovania.

```
procedure zapis_zisk_ziaka(ocislovanie_ulohy, bodovy_zisk: integer);
```

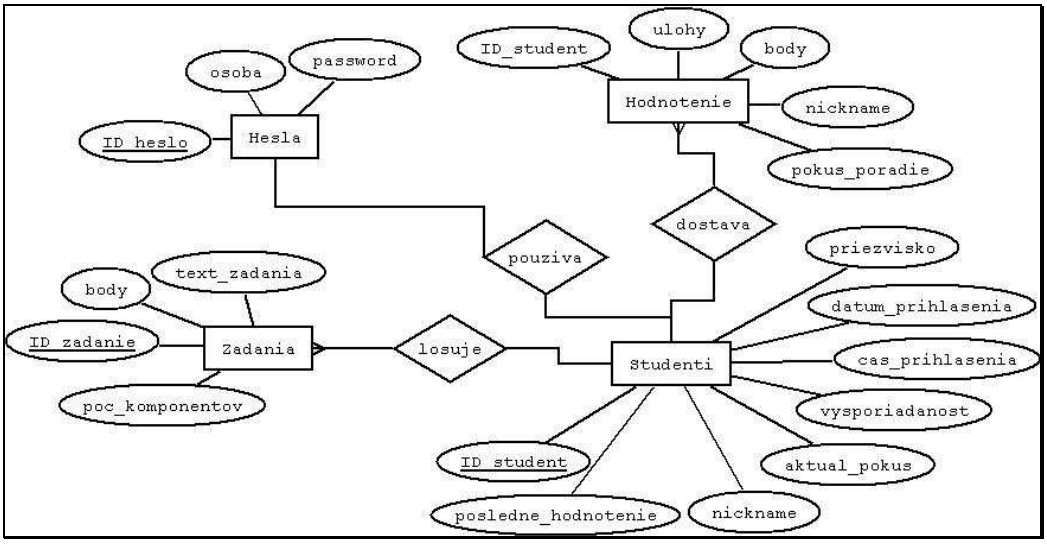
- Procedúra aktualizujúca údaje po vykonaní a vyhodnotení testu v tabuľke hodnotenie.

```
procedure zmena_hesla(osoba: string; nove_heslo: string);
```

- Procedúra zmení heslo skupine študent alebo vyučujúcemu, vykonávať môže proces zmeny hesla len vyučujúci.

4.8 Návrh databázového systému pre aplikáciu

Pre vizuálnu reprezentáciu dátových objektov sme použili *Entito – relačný diagram*, ktorý nám poskytuje globálny pohľad na štruktúru vzťahov v tabuľkách databázy. Na obrázku č. 11 sú zobrazené jednotlivé *entity databázového systému* a predstavujú tabuľky so záznamami. Ku každej entite sú priradené príslušné atribúty. Na úrovni *ER* modelu sme sa snažili hlavne o zachytenie všetkých potrebných vlastností entít a relácií, resp. vzťahov medzi nimi. Ako sme spomínali, databáza obsahuje štyri tabuľky: Tabuľka *študenti, heslá, zadania a hodnotenie*. Hlavnou tabuľkou je tabuľka študenti, ktorá s ostatnými tabuľkami je v určitom vzťahu, to znamená, že študent sa prihlasuje prostredníctvom prístupového hesla, z tabuľky zadania sa mu vygeneruje päť zadaní a do tabuľky hodnotenie sa zapisujú výsledky z vypracovaných zadaní. Na obrázku č. 11 je znázornená štruktúra, resp. *entito – relačný diagram*.



Obr. 11 ER diagram vzťahov v programe DIA

4.8.1 Význam jednotlivých atribútov entít

- **Entita Heslá obsahuje atribúty:**
  1. *ID_heslo* – je jedinečný identifikátor každého záznamu
  2. *osoba* – obsahuje textovú informáciu „študent“ alebo „profesor“ podľa toho, pre koho je konkrétne heslo určené
  3. *password* – obsahuje heslo uložené ako text typu Varchar[15], resp. dĺžka 15 znakov.
- **Entita Zadanie obsahuje atribúty:**
  1. *ID_zadanie* – jedinečný identifikátor charakterizujúci každé zadanie
  2. *text_zadania* – je typu text a obsahuje požiadavky k jednotlivým zadaniam v textovej forme.
  3. *body* **–** predstavuje celé číslo a udáva maximálny počet bodov, ktoré je možné získať zo zadania.
  4. *poc_komponentov* **–** predstavuje celé číslo vyjadrujúce počet komponentov na formulári testovanej aplikácie, po načítaní tejto hodnoty slúži pri samotnom testovaní na zistenie, či použil študent predpísaný počet komponentov. Ak nesplnil túto požiadavku, kontrola aplikácie sa skončí a súbory sú považované za netestovateľné.
- **Entita hodnotenie obsahuje atribúty:**
  1. *ID_student* **–** cudzí kľúč, číslo určené na prepájanie tabuľky s primárnym kľúčom v inej tabuľke.
  2. *ulohy* **–** číslo vyjadrujúce očíslovanie, resp. poradové číslo konkrétnej vylosovanej úlohy podľa očíslovania v tabuľke zadania.
  3. *body* **–** číslo udávajúce počet získaných bodov pri vyhodnotení odovzdaných zadaní. Pre každú vyriešenú i nevyriešenú úlohu sa priradí určitý počet bodov závislý od správnosti testovanej aplikácie. Ak úloha nebola vypracovaná a odovzdaná, systém priradí k zadaniu hodnotu, body = 0
  4. *nickname* **–** užívateľské meno študenta, slúži pri filtrovaní údajov na výpis údajov o študentovi a jeho získaných bodoch a na zjednodušenie prístupu k údajom z vyhodnotených zadaní v tabuľke.
  5. *pokus_poradie* **–** obsahuje číslo udávajúce poradové číslo pokusu riešenia zadaní užívateľa, prvou registráciou, resp. prihlásením užívateľa sa nastaví táto premenná na hodnotu 0.

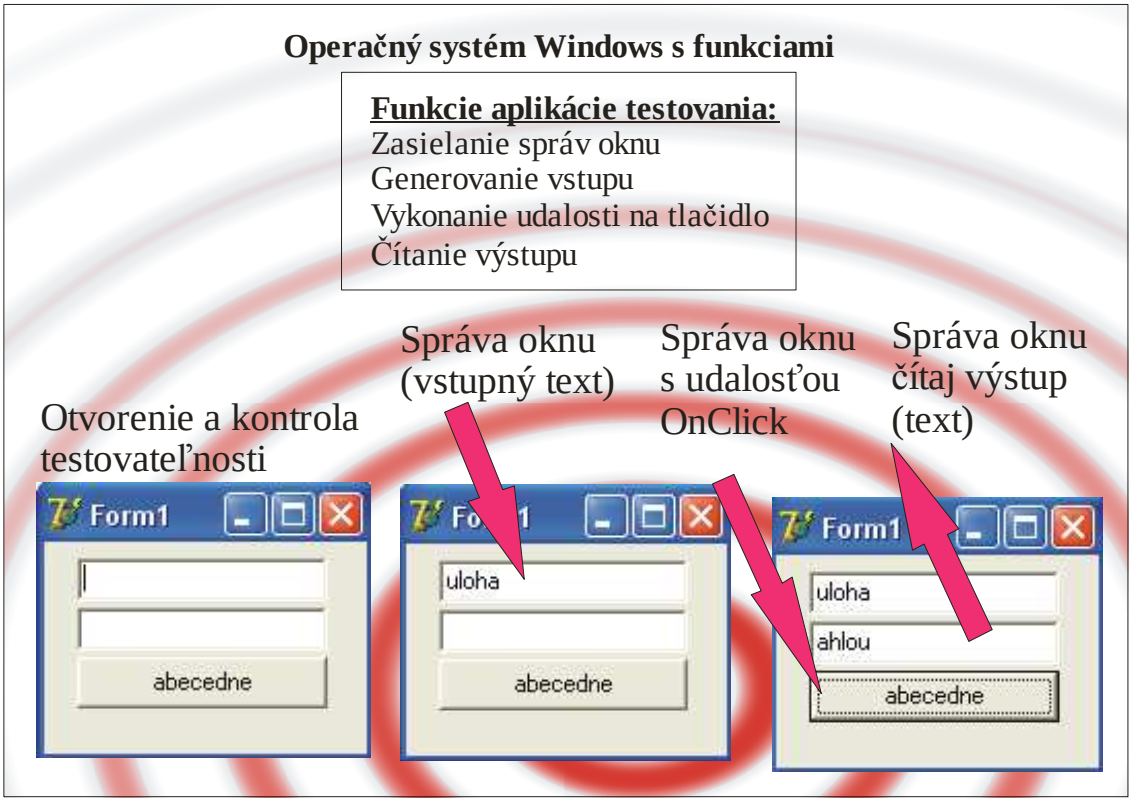
- **Entita študenti obsahuje atribúty:**
  1. *ID_student* – jedinečný identifikátor prihláseného študenta
  2. *nickname* – užívateľské meno prihláseného študenta, v tabuľke smie figurovať len jeden záznam nickname
  3. *priezvisko* – doplňujúci údaj k údaju nickname, priezvisko sa môže opakovať vo viacerých záznamoch.
  4. *datum_prihlasenia* – obsahuje dátum posledného prihlásenia študenta, slúži na lepšiu identifikáciu dátumového rozhrania, kedy bol test približne vykonaný
  5. *cas_prihlasenia* – obsahuje presný čas posledného prihlásenia užívateľa a v kombinácii s dátumom tvorí špecifický nástroj na konkretizáciu prihlásenia užívateľa
  6. *aktualny_pokus* – poradie aktuálne vykonávaného testu alebo poradie pokusu po vylosovaní nového zadania.
  7. *posledne_hodnotenie* – percentuálna hodnota bodového zisku z posledného odovzdaného zadania.
  8. *vysporiadanost* – hodnota obsahujúca text „*áno*“ alebo „*nie*“ podľa toho, či študent odoslal posledné zadanie na testovanie alebo nie. Ak boli vyžrebované zadania, ale neboli vypracované a odoslané, hodnota sa nastaví na „*nie*“, užívateľ nie je vysporiadaný so systémom.

4.9 Činnosť aplikácie

Jadrom našej aplikácie je testovací mechanizmus *EXE* súborov vytvorených vo vývojovom prostredí Delphi podľa stanovených podmienok. Databázový systém a skupinu vyššie spomínaných funkcií by sme mohli považovať za obslužné podprogramy. Ich úlohou je doplniť funkcionalitu ostatnej časti aplikácie, aby ju bolo možné používať v bežných školských podmienkach. Bez nich by uplatnenie aplikácie bolo významné prevažne pre jednotlivca. V skratke môžeme rozdeliť aplikáciu na tri programové časti. Prvá rieši vstupné informácie prichádzajúce do aplikácie, druhou hlavnou časťou je spracovanie vstupu transformovaním na výstupy, treťou je výmenník informácií prostredníctvom databáz.

Spracovanie informácií si predstavujeme nasledovne. Predpokladajme, že máme zvládnutý systém zisťovania identifikátora okna v operačnom systéme Windows, potom

by sme potrebovali pristupovať k jeho dcérskym oknám a komunikovať s nimi nasledujúcim spôsobom, ktorý je uvedený na obrázku č 12.



Obr. 12 Spôsob komunikácie s oknom

Ako vidíme na obrázku č. 12, jednou z hlavných náplní našej aplikácie je otvorenie *EXE* súboru, ak poznáme jeho absolútnu adresu. Po otvorení potrebujeme získať identifikátor okna, ktoré sme otvorili. Z otvoreného *EXE* súboru nezískavame špecifický identifikátor na základe vlastnosti *Caption* formuláru ani priamo na základe triedy formulárov. Táto možnosť je tiež použiteľná, ale prináša so sebou viacero rizík. V systéme Windows sa môže nachádzať množstvo okien s rovnakou vlastnosťou *Caption* alebo s rovnakou triedou okna, preto ak by ich bolo spustených v jednom okamihu viac, stratili by sme kontrolu nad oknom, s ktorým potrebujeme komunikovať. Preto je výhodnejšie uskutočniť najskôr otvorenie okna pomocou adresy, resp. cesty k súboru, a následne získať identifikátor z vytvoreného procesu, ktorý spomedzi všetkých okien vyhľadáme pomocou triedy okna. V Delphi je štandardná trieda okien typu *TForm*. Pracujeme tak s jedným procesom a nemusíme sa obávať možných komplikácií. Súčasťou prípravy je overenie, či študent postupoval podľa zadania a použil toľko komponentov, koľko je pre danú úlohu preddefinované. Ak sú všetky podmienky splnené, naša aplikácia generuje vstupné hodnoty.

Na obrázku č.12 máme uvedený príklad, kde sa vstupná postupnosť znakov malých písmen abecedy usporiada do abecedného poradia po kliknutí na tlačidlo. Do vstupného komponentu, resp. okna patriaceho formuláru, ktorého identifikátor si zistíme z rodičovského okna, zašleme správu na vloženie textu a vyplníme text okna našim vygenerovaným slovom. Počkáme, kým sa správa vykoná a následne posielame ďalšiu správu, v ktorej definujeme príkaz na vykonanie udalosti stlačenia tlačidla, *OnMouseClick* tak, akoby sme to vykonávali manuálne kliknutím myšou. Aplikácia vykoná algoritmus, usporiada vstupnú postupnosť znakov a výsledok zapíše do výstupného komponentu. Nakoľko poznáme identifikátor okna, vieme si výslednú hodnotu z okna *edit* prečítať ako text a preniesť ju pomocou správy do našej testovacej aplikácie a skontrolovať jej korektnosť oproti očakávanej hodnote. Ak sa výsledok zhoduje s výslednou hodnotou na základe generovaného vstupu, priradíme študentovi bod za správnu odpoveď. Tento cyklus opakujeme pre všetky odoslané aplikácie študenta určený počet krát, pritom napĺňame vstupné *edity* náhodnými hodnotami.

Z dôvodu, že sa chceme vyhnúť kolízii programu pri zasielaní správ, vykonávame pri každom zaslaní správy oknu krátke oneskorenie *delay* s trvaním približne potrebným na spracovanie a vykonanie danej správy, najčastejšie 10, 20 alebo 100 ms. Na obrázku č. 12 sme v hornej časti obrázku spomenuli operačný systém Windows, práve vďaka ktorému máme možnosť správy zasielať pomocou Windows *API* funkcií, ktorými sa budeme zaoberať v nasledujúcich kapitolách. Po skončení kontrolného cyklu musíme testovaný formulár terminovať, ukončiť, aby nespôsoboval zbytočné čerpanie systémových prostriedkov a nemuseli ho neskôr manuálne zatvoriť.

4.10 Hlavný program systému testovania

Hlavnú programová časť systému testovania tvorí tlačidlo „*Poslať*“, ktoré obsahuje zdrojový kód volajúci funkcie spracovávajúce údaje v databáze a zároveň údaje vznikajúce pri komunikácii so súborom testovanej aplikácie. Po stlačení tlačidla sú volané funkcie a procedúry umiestnené v externých programových častiach *unitoch,* z dôvodu prehľadnosti zdrojového kódu. Pre každú úlohu, ktorú systém vygeneruje študentovi, algoritmus vytvorí premennú typu *record.* Každá úloha má definovaných šesť atribútov, ktoré sú uvedené v nasledujúcom textovom bloku. Atribúty každého záznamu slúžia na zjednodušenie prístupu k vlastnostiam konkrétnej úlohy a na evidovanie informácií o úlohách.

Zoznam atribútov záznamu *TUloha* s popisom ich významu:

```
type TUloha=record
     ocislovanie:integer; //očíslovanie úlohy v databáze
     umiestnenie:string;  //umiestnenie vypracovanej úlohy na disku
     odoslal:boolean;     //informácia o odoslaní úlohy
     znamka:integer;      //známka z odovzdaných úloh
     pocet_kom:integer;   // požadovaný počet komponentov pre zadanie
     akt_pokus:integer;   // číslo pokusu testovania
end;
```

Informácie o jednotlivých úlohách sú využívané hlavne pri konečnom hodnotení a v rozhodovacom procese, či bude daná úloha hodnotená. Ak by sme pripustili na testovanie úlohu, ktorá nespĺňa požiadavky, mohol by tento fakt negatívne ovplyvniť proces testovania, v najhoršom prípade by nemusel vôbec prebehnúť alebo by mohol skončiť hlásením chyby. Preto je veľmi dôležité testovať aplikácie, ktoré aspoň z kozmetického hľadiska spĺňajú definované podmienky. Nasledujúci textový blok obsahuje časť zdrojového kódu, kde môžeme vidieť postupnosť príkazov, volania jednotlivých funkcií a vplyv niektorých atribútov:

```
var i,zisk: integer;
    p_komp: integer;
    proces: cardinal;
begin
for i:=1 to 5 do              //prejdi všetkými odoslanými aplikáciami
  begin
    if ulohy[i].odoslal then  //ak bola odoslaná i-tá úloha
      begin
        ulohy[i].pocet_komp:=zistenie_poctu_kom(ulohy[i].ocislovanie);
        proces:=Open_Application(ulohy[i].umiestnenie); //otvorí apl.
        p_komp:=Get_Child_Handles(whnd); //zisti počet dcérskych okien
        if p_komp=ulohy[i].pocet_komp then  //ak je počet správny, rob
        begin
          Sort_Components(ulohy[i].pocet_komp);  // trieď komponenty
          zisk:=fill_and_val(ulohy[i].ocislovanie,ulohy[i].pocet_komp)
          ulohy[i].znamka:=zisk;   // zisti bodový zisk
          zapis_zisk_ziaka(ulohy[i].ocislovanie,zisk); // zapíš zisk
        end
        else
          zapis_zisk_ziaka(ulohy[i].ocislovanie,0);//inak zapíš zisk 0
        TerminateProcess(proces,Exit_Code);  // zavri okno
      end else zapis_zisk_ziaka(ulohy[i].ocislovanie,0); // zisk 0
  end;
    kolky_pok:=zisti_pokus(aktual_online); //aktualizuj poradie pokusu
    zmen_na_vysporiadanost(aktual_online,kolky_pok,vysledok);
end;
```

V prvom rade vykonáme test len v prípade, ak bol odoslaný testovaný súbor, podmienka „*if ulohy[i].odoslal*“ musí nadobúdať hodnotu *true*. Ďalšou podmienkou je počet vstupných a výstupných komponentov vytvorenej aplikácie. Ak sú splnené tieto podmienky, vykoná sa komunikácia s aplikáciou, resp. *EXE* súborom, ktorej sa budeme venovať v ďalších podkapitolách.

4.11 Systémové prostriedky na prácu s EXE súbormi

Samotná kontrola aplikácie spočíva vo vymedzení a použití niekoľkých funkcií medzi otvorením a ukončením bežiacej aplikácie. Správnym nastavením a načasovaním jednotlivých úkonov je možné jednoduchým spôsobom spravovať rôzne aplikácie.

Windows API (Windows Application Programming Interface), tiež nazývané *WinAPI*, je súbor funkcií poskytovaných operačným systémom Windows od spoločnosti Microsoft, ktoré podporujú vytváranie aplikácií a prácu s nimi. Výhodou je, že tieto funkcie môžeme volať z prostredia Delphi. *WinAPI* umožňuje prístup k systému súborov Windows zariadení, procesov, vlákien, *GDI (Graphics Device Interface)* používané na výstup vizuálneho obsahu na monitor, či tlačiareň. Veľmi významnou funkciou je tvorba a správa okien vo Windows a manipulácia s myšou, vstup z klávesnice a pod. Delphi prekladá volanie funkcií *API* do syntaxe zrozumiteľnej Windows. Tento kód sa nachádza v *unitoch ShellAPI*. Ak potrebujeme používať funkcie *WinAPI*, musíme uviesť názov programových funkcií do časti *uses* v zdrojovom kóde programu.

4.12 Stratégia práce pre manipuláciu s oknami

Stratégia celého testovacieho procesu je obsiahnutá v dvoch hlavných pracovných *unitoch* vytvorených v Delphi. Sú prenositeľné do akejkoľvek aplikácie vytvorenej v tomto programovacom jazyku zapísaním názvu unitu v sekcii uses používaného programu. V *unitoch* sú naprogramované samostatné funkcie, ktoré voláme z hlavného programu. Tieto pracovné *unity* sme preto pomenovali podľa náplne práce, ktorú vykonávajú:

1. **unit WorkWithHandles**
2. **unit UnitWithTasks**

4.12.1 Unit WorkWithHandles

WorkWithHandles je unit určený na manipuláciu s Windows oknami prostredníctvom ich identifikátora, reprezentovaný údajovým typom *THandle*. Okno, s ktorým potrebujeme manipulovať, nie je zatiaľ otvorené, informácie, ktoré máme k dispozícii je absolútna adresa k danému *EXE* súboru. Pomocou existujúcej adresy sa pokúsime súbor otvoriť a vytvoriť bežiaci proces v systéme. Postupnosť príkazov zhrnieme nasledovným spôsobom v niekoľkých bodoch.

Postupnosť krokov pri manipulácii s oknom v unite WorkWithHandles:

1. Otvorenie EXE súboru s využitím absolútnej adresy k súboru.
2. Vytvoríme proces.
3. Získame hodnotu identifikátor procesu, *dwProcessID* z *ProcessInformation*.
4. Na základe tejto hodnoty zistíme manipulátor okna, v programátorskej praxi označovaný ako *handle*, vyššie spomínaného typu *THandle*. K tejto úlohe pristupujeme viac menej netradičným spôsobom. Pre okno, ktorého *handle* nám je známy, vieme zistiť ID procesu, no opačný spôsob zisťovania nie je možný. Preto jediný spôsob je prehľadávať všetky okná v systéme v systéme, ku každému oknu pamätať jeho *handle* a z tejto hodnoty následne zistiť ID procesu a hľadať okno, ktorého ID procesu zaznamenalo zhodu s našim ID procesu. Hľadáme spomedzi všetkých bežiacich procesov vo Windows ten, ktorého ID procesu je zhodné s hľadaným a zároveň trieda formuláru je *TForm1*, *TForm1* je štandardná trieda pre formuláre v Delphi.
5. Máme k dispozícii *handle* okna, s ktorým sa chystáme komunikovať, *handle* okna považujeme za rodičovské okno. Z neho pomocou Windows funkcie vyhľadáme všetky dostupné dcérske okná, *child windows*, z ktorých získame ich *handle*, ďalej vlastnosť *Caption* a *Class*. Pri každom nájdení okna vytvoríme nový objekt *HndRec* triedy *THndRec* s tromi atribútmi, ktorý bude úložným priestorom pre nájdený objekt, u ktorého si pamätáme tieto tri vlastnosti.
6. Vytvoríme si zoznam *Tlist*, do ktorého ukladáme každý nájdený záznam *HndRec*. Práca s objektmi triedy *TList* má veľa výhod, nemusíme dopredu poznať počet prvkov, objektov, ktoré bude obsahovať, preto z nášho hľadiska má výhodnejšie vlastnosti a použitie ako vytvárať dynamické pole so záznamami *HndRec*.

7. Majme teda kompletný zoznam dcérskych okien spolu s ich vlastnosťami, potom pomocou funkcie na roztriedenie daných objektov patriacich rodičovskému oknu rozdelíme objekty na tie, ktoré budú slúžiť ako vstupné okná (trieda *Tedit*), ďalej jedno výstupné okno triedy *Tedit.* Samostatný prvok bude tlačidlo, ktoré musí obsahovať každá testovaná aplikácia; tlačidlo musí byť typu *Tedit*. Ak okno nebude obsahovať tieto komponenty, považujeme aplikáciu z pohľadu testovateľnosti za netestovateľnú a vylúčime ju zo zoznamu testovaných súborov.
8. Z predchádzajúcich krokov máme k dispozícii všetky potrebné *manipulátory*. Vstupné komponenty naplníme pomocou funkcie náhodne generovanými, ale obsahovo cielenými hodnotami. Vykonáme túto činnosť zaslaním správy konkrétnemu identifikátoru okna *handle* a počkáme, kým sa každé okno naplní hodnotou.
9. Keď sú vstupné okná naplnené želanými hodnotami, zašleme správu i tlačidlu *button*, ktoré má tiež svoj identifikátor *handle*. Tlačidlo po stlačení vykoná algoritmus, ktorý je obsiahnutý v zdrojovom kóde aplikácie. Musíme počkať nejaký čas, kým sa proces neskončí. O spôsobe kopírovania a vkladania textu do okien si povieme bližšie v nasledujúcich kapitolách.
10. Ak výpočet skončil, prepojíme sa pomocou *handle* do výstupného okna *TEdit*, a skopírujeme text, ktorý sa v ňom nachádza. Text neprekonvertujeme na potrebný údajový typ. Správnosť výstupnej hodnoty kontrolujeme s nami očakávanou hodnotou.
11. Nakoniec je potrebné uvoľniť pamäť pre ďalšiu iteráciu testovania a vytvorený proces ukončiť a s ním aj všetky súvisiace procesy a okná. K ukončeniu procesu využijeme atribút *hProcess* zo štruktúry *ProcessInformation*. Písmeno „*h*“ pred názvom premennej alebo atribútu triedy považujeme za označenie identifikátora okna – *handle.*

4.12.1.1 Zoznam použitých funkcií a procedúr

Funkcie obsiahnuté v *unit WorkWithHandles* tvoria jadro zdrojového kódu určeného na prácu s oknami vo Windows. Funkcie sú uvedené v poradí, v akom ich pri komunikácii s oknami používame. Funkcie navzájom spolu komunikujú prostredníctvom parametrov, a zväčša výstup jednej funkcie je vstupom pre inú funkciu alebo procedúru.

Zoznam aplikácii aj s ich parametrami:

function Open_Application (adress: string): cardinal;
function Get_Window_From_Process (p_tid: cardinal; p_cl: string): THandle;
function Get_Child_Handles (p_parenthnd: THandle): integer;
function List_Child_Handles (p_strings: TStrings): integer;
procedure Sort_Components (p_comp: integer);
procedure Write_Text_To_Handle (p_hnd: Thandle; p_text: string);
procedure Send_Click_To_Handle (p_hnd: THandle);
function Read_Text_From_Handle (p_hnd: Thandle): string;

4.12.1.2 Popis funkčnosti podprogramov:

a) Funkcia Open_Application

Podstatou funkcie Open_Application je otvoriť požadovaný súbor, resp. vytvoriť v pamäti proces, a určiť *handle* jeho okna a tiež *handle* príslušného procesu. Majme deklaráciu globálnych premenných *PInfo a SUInfo*:

```
var PInfo: TProcessInformation; SUInfo: TStartUpInfo;
```

potom telo funkcie *Open_Application* sa vykoná nasledovne:

```
function Open_Application (adress: string): cardinal;
```

- *ZeroMemory* zmaže pamäť pre vyhradenú pre PInfo a SUInfo
- Nastavíme potrebné atribúty *SUInfo*
  - `Cb:=sizeof(SUInfo);`
  - `wShowWindow:=SW_SHOW;`
  - `dwFlags:=STARTF_USESHOWWINDOW;`
- Ak sa proces vytvorí volaním funkcie *CreateProcess* prostredníctvom adresy potom:
- **Opakuj postupnosť:**
- Zisti handle okna z ID procesu tak, že voláme funkciu *Get_Window_From_Process* s parametrami *PInfo.dwProcessID* a triedou hľadaného okna *'TForm1'*
- **Pokiaľ sa nenájde handle okna alebo kým neuplynie čas 3 sekundy.**
- Funkcia vráti *handle* okna, inak vypíše správu „Nepodarilo sa spustiť proces!“

- Návratová hodnota funkcie *Open_Application* má hodnotu *PInfo.hproces***,** resp. *handle* procesu.

Použité programové prostriedky vo funkcii Open_Application:

`ZeroMemory (Destination, Length)` – Potrebujeme používať štruktúru *ProcessInformation a StartUpInfo*, preto je potrebné vymazať blok pamäte pre danú štruktúru, funkcia *ZeroMemory* má dva parametre:

- *Destination* – ukazovateľ na počiatočnú adresu bloku pamäte, ktorý bude vymazaný
- *Length* – veľkosť bloku pamäte v byte, ktorá bude vymazaná

`SUInfo` – štruktúru *StartUpInfo* sme použili v spojení s funkciou *CreateProcess*, prostredníctvom nastavenia atribútov *cb, wShowWindow a dwFlags* sme určili hlavné vlastnosti okna, vzhľad okna, v prípade ak je vytvorené nové okno pre proces.

`CreateProcess` **–** Vytvorí nám nový proces a jeho primárne vlákno. Nový proces prebieha v zabezpečenom kontexte volajúceho procesu. Funkcia má 10 parametrov, druhý z nich je absolútna adresa k súboru. Týmto parametrom je *lpCommandline*. Ostatné parametre obsahovo nie je potrebné vysvetľovať kvôli náplni našej práce.

`PInfo.dwProcessID` – ID proces typu *doubleword*. Táto hodnota môže byť použitá pre identifikáciu procesu v systéme procesov vo Windows. Hodnota *dwProcessID* je platná len do okamihu, kým sa všetky *handle* daného procesu uzavrú a alebo kým sa objekt neuvoľní z pamäte. My používame tento atribút na hľadanie nášho procesu v zozname všetkých bežiacich procesov, ktoré sú v čase hľadania spustené. Aby sme zabezpečili presnosť pri hľadaní, pomocným parametrom vo funkcii *Get_Window_From_Process* je trieda formuláru, ktorého vlastnosti potrebujeme získať.

`PInfo.hproces` – návratová hodnota funkcie, typ *cardinal*, jedná sa o *handle* novovytvoreného procesu. V programe využívame tento atribút na *terináciu* procesu, preto si *handle* procesu ukladáme pre neskoršie použitie. (MICROSOFT, 2011c)

b) Funkcia Get_Window_From_Process

Z predchádzajúceho podprogramu voláme funkciu *Get_Window_From_Process* s parametrami *p_tid* (ID procesu) a *p_cl* (trieda okna) na zistenie *handle* okna z ID procesu. Postup je nasledovný. Majme funkciu a deklaráciu lokálnych premenných:

```
function Get_Window_From_Process(p_tid:cardinal;p_cl:string): THandle;
var thnd: THandle;  tID: cardinal; tcl: array [0..MAX_CHARS]  of char;
```

potom priebeh funkcie môžeme popísať nasledovne:

- Na začiatku nastavíme návratovú hodnotu funkcie na nulu, v prípade predpokladu, že nenájdeme žiadny požadovaný identifikátor *handle*.
- Budeme prehľadávať zásobník okien v systéme. Nájdeme si prvé okno v rámci celého systému, ktoré by potenciálne mohlo byť oknom, ktoré hľadáme.
- **Opakuj postupnosť:**
    - Ak nájdeme okno, získame ID procesu tohto okna
    - Porovnáme ID procesu, ktorého okno hľadáme, s ID procesu, ktoré sme práve našli v predchádzajúcom kroku.
    - Ak sa ID procesu hľadaného a procesu nájdeného rovnajú, potom z na základe *handle* okna získame triedu okna.
    - Ak sa aj trieda nájdeného okna zhoduje s triedou okna, ktorú hľadáme, resp. *TForm1*, funkcia vráti hodnotu *handle* okna, ktoré považuje za výsledok hľadania.
    - Hľadanie bezprostredne ukončíme v prípade nájdenia okna s požadovanými vlastnosťami, v opačnom prípade nájdeme ďalšie okno v rámci systému a pokračuje v cykle od začiatku.
- **Kým *handle* novo nájdeného okna bude nulový.**
- Po skončení cyklu sme si istí, že v systéme už nie je viac okien, ktoré by boli predmetom nášho zisťovania, preto uzavrieme *handle*, ktorý sme našli.

Použité programové prostriedky vo funkcii Get_Window_From_Process:

`FindWindow (ClassName, WindowName: String);` – *WinAPI* funkcia, ktorá vráti identifikátor okna *handle*, ktorého názov triedy a názov okna zodpovedá zadaným reťazcom v parametroch funkcie. Treba si dávať pozor pri zadávaní parametrov, pretože táto funkcia je *„case sensitive“*. Funkciu *FindWindow* sme využili pri hľadaní prvého okna v rámci systému.

`GetWindowThreadProces (thnd: Thandle; tID: cardinal);` – funkcia načíta identifikátor vlákna, ktoré vytvorilo špecifikované okno a vráti identifikátor procesu *tID*, ktorý vytvoril okno. V programe používame funkciu na zistenie identifikátora procesu, pre nájdené okno. Premenná *thnd* je *handle* okna, ktoré je vytvorené procesom s identifikátorom *tID*.

`GetClassName (thnd: Thandle; tcl: String; Max_chars: integer);` - načíta názov triedy, ktoré patrí k špecifikovanému oknu. Význam parametrov funkcie:

- *Thnnd* - handle okna, pre ktoré hľadáme triedu formuláru
- *Tcl* – názov triedy, ktoré z funkcie načítavame do premennej *Max_Chars*
- *Max_chars* – dĺžka názvu triedy ukladanej do *buffer-a* v znakoch

`GetWindow (thnd: Thandle; uCmd: integer);` - vráti *handle* okna, ktoré je v špecifikovanom vzťahu menovanom v parametri *uCmd*, k špecifikovanému oknu zadanému v parametri *thnd* ako *Thandle*. Parameter *uCmd* môžeme určiť siedmimi typmi vzťahov, napr.: *GW_CHILD, GW_HWNDFIRST, GW_HWNDLAST, GW_HWNDNEXT* a i. Našou úlohou bolo využiť funkciu pre hľadanie ďalšieho okna, ktoré je vo vzťahu k rodičovskému oknu. Výhodná možnosť je zápis parametra *GW_NEXT* pre posun na ďalšie okno. Celočíselná hodnota parametra *GW_NEXT* je 5.

`CloseHandle (thnd: THandle);` - zatvorí *handle* daného objektu. Objekt je špecifikovaný hodnotou *thnd*. (MICROSOFT, 2011c)

c) Funkcia Get_Child_Handles

Táto funkcia nám slúži na vyhľadanie všetkých dostupných dcérskych okien, resp. súrodencov, prislúchajúcich k rodičovskému oknu definovanom v parametri funkcie premennou *p_parented*. Aby sme kompletne zvládli komunikáciu s každým komponentom v rodičovskom okne, musíme poznať identifikátory *handle* týchto komponentov. Funkcie *WinAPI* nám umožnujú zistiť počet, typ a iné vlastnosti týchto komponentov na základe identifikátora hodnoty parametra *p_parented*. Majme deklaráciu lokálnych a globálnych premenných:

```
function Get_Child_Handles(p_parenthnd: THandle): integer;
var chhnd: THandle;                          // handle dcérskeho okna
    cap, cl: array [0..MAX_CHARS] of char; // text a trieda okna
    I: integer;                              // riadiaca premenná cyklu
    HndRec: THndRec;                         // objekt triedy THndRec
    Handles: Tlist;                          // zoznam objektov
```

a definíciu triedy *THndRec*, ktorá bude obsahovať objekty patriace rodičovskému oknu,

```
type Thndrec=class                           // definícia triedy THndRec
  hnd: THandle;                              // atribút handle okna
  cap, cl: string;                           // atribúty text a trieda
end;
```

potom priebeh funkcie môžeme popísať nasledovne:

- Ak nie je vytvorený zoznam objektov typu *TList*, vytvoríme ho a budeme doň vkladať objekty typu *THndRec*. Ak je zoznam vytvorený, zmažeme ho a vytvoríme nový.
- Nájdeme prvého potomka rodičovského okna, *chhnd* typu *THandle*
- **Opakuj postupnosť, pokiaľ identifikátor nájdeného okna bude nenulový, rob:**
    - Vytvor nový objekt triedy *THndRec*, resp. záznam *HndRec*
    - Získaj text z okna aktuálneho potomka
    - Získaj názov triedy z okna aktuálneho potomka
    - Zapíš do atribútov premennej *HndRec* tieto získané informácie (*cap, cl, chhnd*)
    - Pridaj objekt, resp. aktuálneho potomka, do zoznamu *handles*
    - Presuň sa na ďalšie okno patriace rodičovskému oknu, nájdi ďalšieho súrodenca potomka.
- Návratová hodnota funkcie je počet nájdených potomkov.

Použité programové prostriedky vo funkcii Get_Child_Handles

`GetWindow (thnd: Thandle; uCmd: integer);` - vráti *handle* dcérskeho okna, ktoré je v špecifikovanom vzťahu menovanom v parametri *uCmd*, k špecifikovanému oknu zadanému v parametri *thnd* ako *Thandle*. Parameter *thnd* je *handle* rodičovského okna. V prípade zisťovania *handle* dcérskeho okna sme ako vzťah *uCmd* využili parameter *GW_CHILD*. V prípade posúvania sa na ďalšie okno v poradí je to vzťah *GH_HWNDNEXT*.

`GetWindowText (hWnd: THandle; lpString: string; Max_count: integer);` - funkcia skopíruje vlastnosť *caption* do *buffera* zo špecifikovaného *handle* okna, v našom prípade *handle* okna potomka. Dokážeme napríklad skopírovať *Caption* pre triedy *TButton* alebo *TEdit*. Vznikne tu však jeden problém, ktorý si rozoberieme v sekcii čítania výsledného text po udalosti *OnMouseClick*.

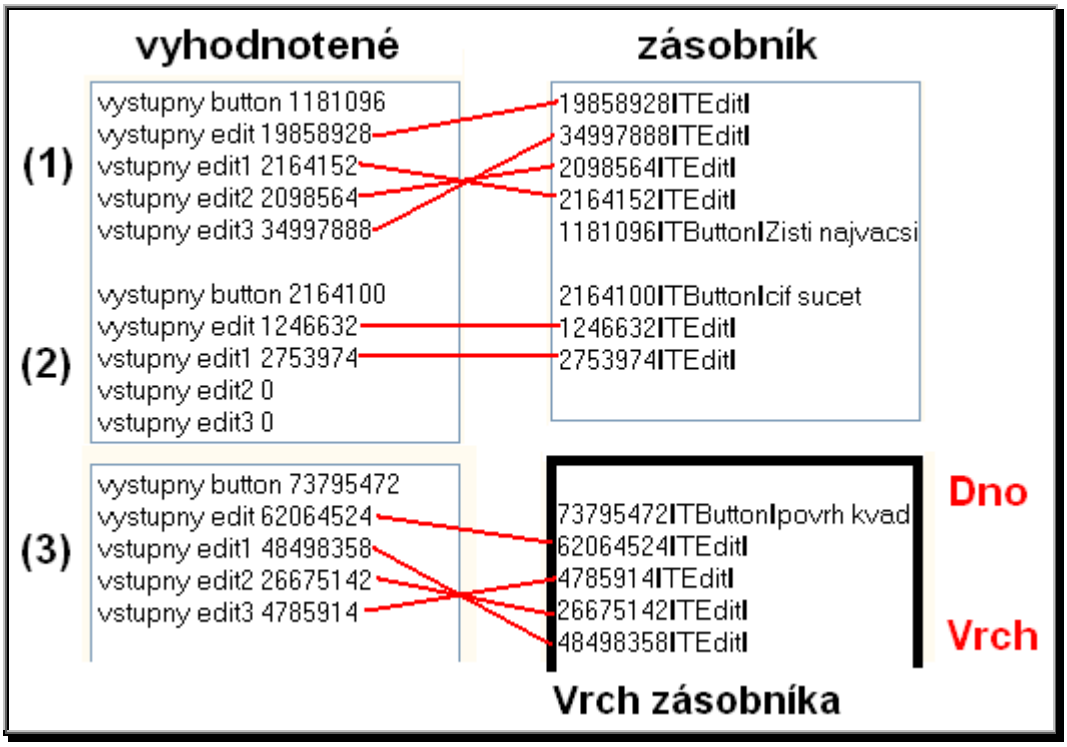
`GetClassName (thnd: Thandle; tcl: String; Max_chars: integer);` - načíta názov triedy, ktoré patrí k špecifikovanému oknu, konkrétne dcérskemu oknu vo funkcii *Get_Child_Handles*. (MICROSOFT, 2011c)

d) Funkcia List_Child_Handles

Funkcia *List_Child_Handles* nemá zvláštny význam pre proces testovania, zaradili sme ju však do kategórie použitých funkcií, nakoľko nám veľmi pomohla pri zostavovaní techniky zaraďovania dcérskych okien do kategórie vstupných a výstupných komponentov. Náplňou funkcie je vypisovať obsah zoznamu *handles* v aplikácii v roztriedenom poradí. Funkcia *Get_Child_Handles* ukladala v cykle jednotlivé *handle* a vlastnosti okien do zoznamu *handles*, avšak problémom bolo zistiť, v akom poradí nachádza algoritmus dcérske okná a ako sú v zásobníkovej pamäti uložené. Podľa podmienok testovacieho systému má študent pri vytváraní aplikácie dodržať pravidlo poradia pri ukladaní komponentov na formulár. Ak má napríklad aplikácia obsahovať dva vstupy, jedno tlačidlo a jeden výstup, potom musí postupovať pri ukladaní komponentov na formulár nasledovným spôsobom:

a) vložíme na formulár prvý vstupný komponent triedy *TEdit*
b) následne vložíme druhý až posledný vstupný komponent triedy *TEdit*
c) poradie vloženia výstupného komponentu *TEdit* je v poradí posledné z rady komponentov triedy *TEdit*
d) poradie vloženia tlačidla triedy *TButton* nemusíme dodržať

Tak ako algoritmus nachádza okná, resp. komponenty, tak sú vkladané do zásobníka. Aj keď tlačidlo položíme na formulár ako posledné alebo predposledné, stáva sa, že ho nájdeme v zásobníku na úplne inej pozícii. Problém to však nepredstavuje, pretože si vieme podľa triedy okna selektívne vybrať daný komponent. Pre konkrétnu aplikáciu v zozname všetkých aplikácií môže výpis *List_Child_Handles* vyzerať nasledovne, ako vidíme na obrázku č. 13.



Obr. 13 Zoznam dcérskych okien po roztriedení

Ako vidíme, prvok, ktorý do zásobníka vchádza ako prvý, vychádza ako posledný, a naopak. Na obrázku č. 13 vidíme, že komponent triedy *TButton* sa nachádza v zásobníku vždy na inej pozícii. Odfiltrujeme ho pomocou triedy okna. Ostatné komponenty triedy *TEdit* sme zaradili do vstupno - výstupných tried nasledovne. Spomedzi komponentov *TEdit* sa výstupný nachádza vždy ako posledný smerom ku dnu zásobníka. Musí byť však aj v aplikácii položený na formulár ako posledný alebo neskôr ako všetky ostatné vstupné komponenty. Od vrchu zásobníka vždy vyberieme toľko komponentov, koľko dcérskych okien sme našli, a odpočítame tlačidlo a výstupný *edit*, resp. rozdiel konečného počtu a čísla dva. Podľa experimentu zistíme, že pre triedu *TEdit* sú z vrchnej časti zásobníka uložené všetky komponenty *TEdit* v poradí, akom boli vložené na formulár, ich počet *1,2...n*. Triedenie komponentov sme realizovali v procedúre *Sort_Components*. Zápis procedúry je nasledovný:

```
Sort_Components (p_comp: integer);
```

Parameter procedúry slúži na kontrolu správneho počtu vložených komponentov na formulár. Keď sú všetky podmienky splnené, je možné pristúpiť k zasielaniu správ medzi testovacou a testovanou aplikáciou.

e) Procedúra Write_Text_To_Handle

Procedúra má na starosti vloženie generovaného vstupného textu do špecifikovaného okna aplikácie. Nakoľko máme zoznam všetkých okien aj s ich identifikátormi *handle*, mohli by sme teoreticky použiť na posielanie textu známu Windows *API* funkciu *SetWindowText (hWnd: THandle; Text: String).* Žiaľ, v zozname dostupných *API* funkcií sme nenašli inú funkciu používajúcu na prenos informácie pri zapisovaní textu identifikátor *handle*. Po realizácii funkcie *SetWindowText* však narazíme na problém. Správu, ktorú posielame aplikácii sa síce vykoná, no fyzicky sa text v okne, do ktorého správu posielame, nezobrazí. Aj po poslaní správy prikazujúcej *Refresh* alebo prekreslenie okna, sa text v okne nezobrazí. Po štúdiu dostupných informácií zo stránky spoločnosti Microsoft sme sa dozvedeli informáciu k používaniu *SetWindowText:* (MICROSOFT, 2011c)

„Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text control in another application.“ (MICROSOFT, 2011a)

Z toho vyplýva že môžeme zmeniť text okna, pokiaľ ide o aplikáciu, z ktorej voláme funkciu na zmenu textu, no v inej aplikácii to už nie je možné. S problémom zapisovania textu sme si poradili iným spôsobom. Využili sme pritom vlastnosť schránky vo Windows, vloženie obsahu schránky na určené miesto, známu pod názvom *Clipboard*. V programovej časti *uses* sme pridali možnosť využívať schránku, zápisom *„Clipbrd“*, a deklarovali sme premennú na prácu so schránkou: `clipb: TClipBoard;`

Kritická sekcia

Pri zapisovaní, čítaní a vykonávaní udalosti na tlačidlo sme museli vyriešiť problém kritickej sekcie. So schránkou vo Windows môžu pracovať aj iné procesy, samozrejme i užívateľ našej aplikácie. Iný proces alebo užívateľ, ktorí by chceli pristupovať k rovnakým dátam, by mohli svojím prístupom obmedziť funkčnosť celej aplikácie. Taktiež správa inej aplikácie, či procesu určená našej aplikácii by mohla predbehnúť poradie správ, ktoré sme si preddefinovali v programe. Cieľom bolo, aby vlastníkom kritickej sekcia bola naša aplikácia a po ukončení práce kritickú sekciu opustila a uvoľnila miesto kritickej sekcie pre iný proces. Aplikácia bude spracovávať len správy, ktoré posielame prostredníctvom programu, žiadny proces nesmie narušiť

poradie nami posielaných správ alebo vymazať, či zmeniť obsah schránky na neželanú hodnotu.

Realizácia zapisovania textu pomocou schránky:

- Ak nie je vytvorená schránka *ClipBoard*, vytvoríme ju
- Nastavíme našu aplikáciu ako vlastníka kritickej sekcie
- Vstúpime do kritickej sekcie
- Zmažeme obsah *ClipBoard-u*
- Do schránky vložíme potrebný text
- Posielame správu označujúcu text v zadanom okne na základe *handle* tohto okna (označíme znaky od pozície 0 pod maximálnu možnú hodnotu)
- Vkladáme text zo schránky do okna definovanom parametrom typu *Thandle*
- Po ukončení operácie vkladania vyčistíme *ClipBoard*
- Opustíme kritickú sekciu

Použité programové prostriedky vo funkcii Write_Text_To_Handle

Majme definovanú kritickú sekciu premennou, `CS: TRTLCriticalSection;`

`GetCurrentThreadID` – funkcia bez parametrov, funkcia vráti hodnotu identifikátora vlákna. Toto ID bude vlastníkom kritickej sekcie.

`CS.OwningThread` – nastavíme vlastníka kritickej sekcie (*GetCurrentThreadID*)

`EnterCriticalSection (lpCriticalSection: String);` – funkcia zabezpečí vstup do kritickej sekcie. Parameter *lpCriticalSection* je ukazovateľ na objekt kritickej sekcie, teda CS.

`SendMessageCallBack` **–** funkcia pošle špecifickú správu oknu. Ako náhle sa správa vykoná, systém volá špecifickú *CallBack* funkciu, požiadavka na potvrdenie o vykonaní správy, tzv. spätné volanie. Pri volaní funkcie *SendMessage* bez spätného volania sa nám stávalo, že sa správa nestihla vykonať v potrebnom čase a začala sa vykonávať ďalšia správa. To spôsobilo, že okno nebolo vyplnené textom a následne sme počítali so situáciou, že okno už požadované hodnoty obsahuje.

`EM_SETSEL` **–** správa vo funkcii slúži na výber, alebo označenie radu znakov v špecifikovanom okne. Parametrami sú východisková pozícia výberu znakov a konečná pozícia výberu. Tento proces zabezpečí, že označený text bude možné

skopírovať a pri ďalšom vkladaní textu sa predchádzajúci obsah prepíše novým obsahom schránky.

`WM_PASTE` – aplikácia odošle správu *WM_PASTE* a skopíruje, vloží obsah, resp. text v schránke do špecifikovaného okna, definovaného parametrom typu *THandle*, vo funkcii *SendMessageCallBack*. Vkladaný obsah musí obsahovať dáta v *CF_TEXT* formáte.

`LeaveCriticalSection (lpCriticalSection: String);` – príkaz pre vlastníka kritickej sekcie na opustenie kritickej sekcie. (MICROSOFT, 2011c)

f) Funkcia Send_Click_To_Handle

Úlohou funkcie *Send_Click_To_Handle* je realizovať kliknutie na tlačidlo bez použitia periférneho zariadenia, myš. Za ukončenie udalosti kliknutia považujeme okamih, keď užívateľ, v našom prípade program po kliknutí na tlačidlo vykoná pustenie tlačidla. Rovnako ako vo funkcii *Write_Text_To_Handle*, je potrebné, aby naša aplikácia bola v momente vykonávania udalosti kliknutia vlastníkom kritickej sekcie, a aby sa zabránilo súbežnému prístupu dvoch procesov k rovnakým údajom. Po ukončení kliknutia kritickú sekciu treba opustiť.

Programová realizácia kliknutia na tlačidlo vo funkcii Send_Click_To_Handle:

- Nastavíme našu aplikáciu ako vlastníka kritickej sekcie
- Vstúpime do kritickej sekcie
- Posielame správu na stlačenie ľavého tlačidla
- Posielame správu na pustenie ľavého tlačidla
- Opustíme kritickú sekciu

Použité programové prostriedky vo funkcii Send_Click_To_Handle

Nasledujúce prostriedky na prácu s kritickou sekciou sú zhodné s funkciami predchádzajúcej funkcie *Write_Text_To_Handle*:

- **GetCurrentThreadID**
- **CS.OwningThread**
- **EnterCriticalSection**
- **LeaveCriticalSection**

`SendMessageCallBack` – Táto funkcia funguje rovnako ako v predchádzajúcom prípade, s tým rozdielom, že obmieňame typ správy, ktorá sa má vykonať. Správa sa posiela oknu s identifikátorom v premennej *p_hnd: THandle*. Na vykonanie kliknutia posielame nasledovné typy správ:

- `WM_LBUTTONDOWN` – Vo funkcii *SendMessageCallBack* posielame špecifikovanému oknu správu s príkazom na vykonanie udalosti kliknutia ľavým tlačidlom myši. Po vykonaní tejto správy zostáva tlačidlo stlačené.
- `WM_LBUTTONUP` – Predchádzajúca správa ponechala tlačidlo stlačené. Vo funkcii *SendMessageCallBack* posielame špecifikovanému oknu správu s príkazom na vykonanie udalosti pustenia ľavého tlačidla myši. (MICROSOFT, 2011c)

g) Funkcia Read_Text_From_Handle

Úplne posledným krokom komunikácie s aplikáciou je načítanie výsledkov po kliknutí tlačidlom. Výslednú hodnotu načítame z výstupného komponentu triedy *TEdit*, ktorého identifikátor už poznáme. Rovnako ako pri kliknutí na tlačidlo, či zapisovaní hodnoty do komponentu *edit*, aj v tomto prípade sme museli vyriešiť bezpečnosť komunikácie týkajúcu sa súbežného prístupu procesov k dátam, resp. vyriešiť oblasť kritickej sekcie. Pri zapisovaní vznikol problém s použitím *WinAPI* funkcie *SetWindowText*. Pri čítaní sa vyskytol problém pri získavaní textu z okna s funkciou *GetWindowText*. Túto skutočnosť sme potvrdili informáciou zo stránky spoločnosti Microsoft.

„However, GetWindowText cannot retrieve the text of a control in another application.“ (MICROSOFT, 2011b)

To znamená, že *GetWindowText* nedokáže získať text z okna v inej aplikácii. Jediný spôsob ako získať text okna je využiť schránku systému vo Windows, *ClipBoard*, ako v prípade zapisovania textu. Vyriešenie kritickej sekcie nám zabezpečilo, že so schránkou môže pracovať len naša aplikácia. Popis algoritmu pri čítaní textu:

- Ak schránka nebola inicializovaná, vytvoríme odkaz na *clipboard*
- Nastavíme našu aplikáciu ako vlastníka kritickej sekcie
- Vstúpime do kritickej sekcie
- Vymažeme obsah schránky
- Posielanie správy oknu na označenie textu, ktorý chceme kopírovať
- Posielanie správy oknu na kopírovanie označeného textu

- Funkcia vráti získaný text z okna
- Uvoľníme *clipboard*
- Opustíme priestor kritickej sekcie

Použité programové prostriedky vo funkcii Read_Text_From_Handle

- **GetCurrentThreadID**
- **CS.OwningThread**
- **EnterCriticalSection**
- **LeaveCriticalSection**

`SendMessage` **–** Posiela požadovanú správu špecifikovanému oknu. Vybrali sme nasledovné správy definované v parametroch funkcie:

- `EM_SETSEL` – Správa oknu, že chceme označiť text v okne.
- `WM_COPY` – Aplikácia posiela správu *WM_COPY* oknu na skopírovanie aktuálneho výberu do schránky vo formáte *CF_TEXT*. (MICROSOFT, 2011c)

4.12.2 Unit UnitWithTasks

Unit UnitWithTasks je programová časť aplikácie testovania slúžiaca z hlavnej časti na generovanie oboru vstupných hodnôt. V *unite* je obsiahnutý zdrojový kód pre desať základných úloh. Pre každú úlohu je generovanie hodnôt realizované náhodným spôsobom s ohľadom na druh riešenej úlohy. Môžeme povedať, že hlavnou náplňou *UnitWithTasks* je tvorba *Test Cases*, testovacích prípadov.

Dôležité je vytvoriť množinu údajov, ktoré budeme používať ako vstupy pre jednoduchú aplikáciu. Za korektne vytvorenú aplikáciu budeme považovať takú, ktorá údaje v jednotlivých testovacích prípadoch úspešne a korektne spracuje. Majme prípad, že chceme vytvoriť testovacie prípady, ktorých úlohou by bude generovať vstupné údaje pre aplikáciu, ktorá rozhodne na základe troch zadaných dĺžok, o aký trojuholník sa jedná, potom výstupom programu môže byť správa v tvare, trojuholník je rovnoramenný, rovnostranný, všeobecný alebo prípad, keď sa trojuholník nedá zostrojiť. Je vhodné, aby testovacie prípady obsahovali takú množinu hodnôt, ktoré odhalia čo možno najväčšie množstvo chýb a ktoré budú pokrývať čo najviac kritických oblastí. Ak by sme do aplikácie posielali veľkosti strán trojuholníka [-5, 2, 4], výsledkom by mala byť správa „*Trojuholník sa nedá zostrojiť*“. V opačnom prípade by

sme mohli prehlásiť, že časť zdrojového kódu nie je v poriadku alebo nie sú dostatočne ošetrené podmienky. Generovanie čo najväčšieho množstva takýchto vstupov nie je vôbec jednoduché, pokiaľ by sa jednalo o zložitejšiu aplikáciu.

Vieme, že dĺžka strany trojuholníka nemôže byť záporné číslo a rovnako vieme, že nemôžeme v testovacích prípadoch zahrnúť všetky číselné hodnoty, nakoľko by sme veľmi rýchlo narazili na hardvérové limity. Testovacie prípady by mali byť v ideálnom prípade presne pre konkrétny druh aplikácie, teda musí obsahovať hodnoty „šité na mieru“. Vidíme, že aj pre tak jednoduchú aplikáciu ako sme uviedli, je potrebné do hĺbky rozobrať množinu možných vstupov. Je preto potrebné si uvedomiť, že pri rozsiahlych projektoch je testovanie oveľa zložitejšou záležitosťou a preto na testovanie projektov takýchto rozmerov je nutný celý tím odborníkov.

My sa však venujeme obsahovo jednoduchším aplikáciám. Ukážme si na príklade generovanie vstupu zadania ôsmej úlohy, kde sa požaduje abecedné zoradenie vstupného slova vytvoreného z rôznych malých znakov abecedy. V nasledujúcej tabuľke vidíme časť zdrojového kódu slúžiaceho na generovanie náhodného slova vytvoreného z malých písmen abecedy:

```
str_length:=1+random(10);
abc_str:='';
  for j:=1 to str_length do
  begin
    abc_char:=chr(97+random(26));
    abc_str:=abs_str+abc_char;
  end;
```

Pre každú úlohu sme nastavili počet iterácií generovania hodnôt na pevný počet opakovaní, desať. Keďže úlohou bolo vytvárať slová z malých písmen abecedy, musíme aj vstupné hodnoty upraviť. Slovo, ktoré vytvárame pozostáva zo znakov *ASCII* tabuľky od pozície 97, po pozíciu 122. Po vytvorení slova usporiadame reťazec triediacim algoritmom abecedne a vytvoríme si výstupný reťazec, ktorý budeme požadovať na výstupe testovanej aplikácie. Z *unitu UnitWithTasks* voláme programové funkcie unitu *WorkWithHandles*, ktoré spracúvajú generované vstupy. Spracované hodnoty z funkcií po ukončení komunikácie s testovaným súborom sa nám priebežne vracajú späť do *UnitWithTasks*. Volanie funkcií *WorkWithHandles* vykonávame nasledovne:

```
Write_Text_To_Handle(vstupny_edit[1],slovo);
Send_Click_To_Handle(button);
if (Read_Text_From_Handle(vystupny_edit) = slovo) then
  inc(hodnotenie);
```

Po vyhodnotení už nasleduje len zápis potrebných údajov do databázy a uzatvorenie okna a ukončenie všetkých príslušných procesov.

4.13 Možnosti vylepšenia aplikácie Online testovanie

Aplikácia automatizovaného testovania by mala spĺňať základné požiadavky na testovanie aplikácií. Preto je našou snahou a víziou do budúcnosti zlepšiť aplikáciu po každej stránke, resp. doplniť programovú funkcionalitu, odolnosť voči chybám a editáciu dôležitých údajov. Z názvu práce vyplynul cieľ práce, testovať aplikácie vytvorené v programovacom prostredí Delphi.

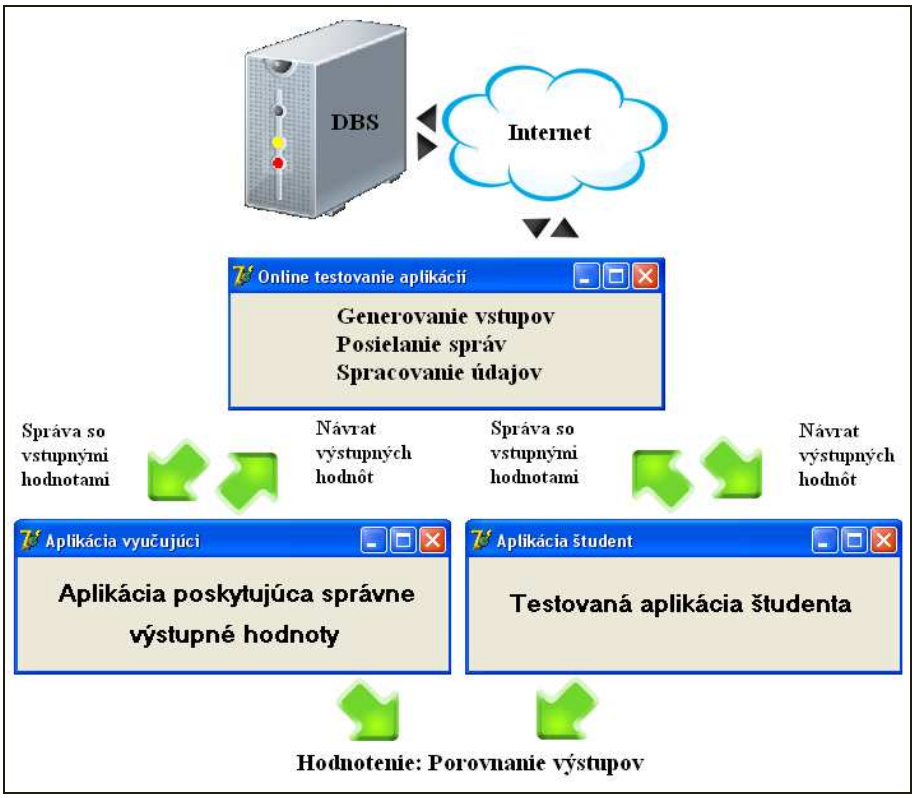
1. Ako sme spomenuli v predchádzajúcich kapitolách, formuláre v Delphi majú triedu *TForm*. Chceli by sme rozšíriť testovanie na úroveň všeobecne použiteľného softvéru testovania pre aplikácie vytvorené v ľubovoľnom programovacom prostredí, napríklad testovanie v jazykoch *C, C*$^{++}$*, Java, Visual Basic* a iné. Pomocou nášho programu dokážeme otvoriť akýkoľvek spustiteľný súbor, no k nadviazaniu komunikácie je potrebných viac špecifických údajov. Aby bola komunikácia úspešná, potrebujeme poznať názvy tried komponentov *TEdit, TButton a TForm*. Viedli by sme databázu všetkých existujúcich tried formulárov a komponentov, ktoré používa daný programovací jazyk.

2. Druhou možnosťou zlepšenia by mohlo byť uvoľnenie pravidiel pri striktnom používaní komponentov určených na zadávanie vstupných a výstupných údajov, a pri používaní tlačidla na vykonanie programu. Študent používajúci aplikáciu by mohol mať pocit väčšej voľnosti, že nie je potláčaná jeho nápaditosť a kreativita. Na čítanie vstupných hodnôt v Delphi by mohla pribudnúť možnosť využívať komponenty *Memo, LabeledEdit, SpinEdit* a ďalšie. Na zobrazenie výstupných hodnôt sme vylúčili komponent *Label*, nakoľko nevieme zistiť jeho identifikátor. Výstup by sme mohli zobrazovať taktiež do *Mema*, či *Listboxu*. Udalosť na tlačidlo by mohla byť vykonaná rôznymi typmi tlačidiel, napríklad *SpeedButton, BitBtn* a pod.

3. Tretím, zatiaľ neodskúšaným návrhom je testovanie grafických aplikácií, práca s *Image*, zisťovanie informácií o jednotlivých obrazových bodoch, zmena farby, určenie určitých špecifických informácií o obraze a i.

4. Štvrtá možnosť vylepšenia, ktorej absencia v súčasnom stave veľmi obmedzuje funkcionalitu našej aplikácie, je nutnosť pracovať so zdrojovým kódom aplikácie v *unite UnitWithTasks*, pri editácii zadaní pre študentov. Autor aplikácie alebo

vyučujúci je nútený otvoriť zdrojový kód programu a doprogramovať generovanie vstupov, spracovanie týchto hodnôt a kontrolu výstupných hodnôt. Proces editácie nie je veľmi náročný, no môže nastať malý omyl pri programovaní nového zadania a testovanie sa môže stať nepresným, pomalým, no aj nefunkčným. Preto by sme zvolili nový spôsob kontroly aplikácií bez nutnosti akéhokoľvek zásahu do programovej časti softvéru so zachovaním jadra hlavných *unitov*. Názorná činnosť upravenej testovacej aplikácie založenej na externom spôsobe editovania úloh vidíme na obrázku č. 14.

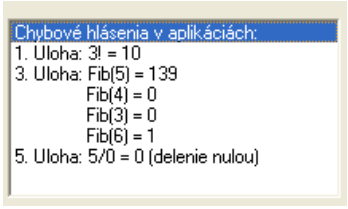


Obr. 14 Princíp testovania upravenej aplikácie

Aplikácia Online testovanie by mohla používať databázu na získanie oboru hodnôt, ktoré má pre danú úlohu testovať a počet komponentov, ktoré je potrebné použiť. Pritom by vôbec nebolo potrebné vedieť, aký typ úlohy riešime. Na každom počítači vybavenom aplikáciou by vyučujúci umiestnil, napríklad do skrytého priečinka, správne naprogramované zadania úloh so špecifickým názvom. K týmto súborom by sa naša aplikácia pred kontrolou študentských súborov pripojila, zapamätala by si poskytované výstupné hodnoty konkrétnej aplikácie pri daných vstupných hodnotách. Tie isté vstupné hodnoty by sme vložili do aplikácie študenta a získali by sme výstupy. Na základe zapamätaných hodnôt z predchádzajúceho kroku vieme tieto výstupy porovnať a konštatovať chybovosť študenta oproti požiadavkám. Nevýhodou tejto

metódy je časová náročnosť, minimálne zdvojnásobenie času potrebného na testovanie, no z celkového hľadiska ide o elegantnejší a používateľsky príjemnejší spôsob spravovania testovaných programov.

5. Poslednou víziou vylepšenia aplikácie Online testovanie by sme chceli poskytnúť možnosť ukladania chybových stavov pri kontrole aplikácie. Súčasný stav aplikácie testovania neumožňuje zistiť, ktorá množina vstupných hodnôt neposkytla správne výstupy. Počas testovania aplikácie by bolo možné ukladať všetky množiny vstupných hodnôt, ktoré spôsobili chybové hlásenia, zamrznutie, či pád aplikácie. Chybové vstupné hodnoty a k nim priradené chybové výstupy by sa ukladali do poľa a boli by k dispozícii pre pedagóga po skončení testovania. Výhodou tohto vylepšenia je podanie dôkazu o tom, že bodové hodnotenie, ktoré dosiahol študent počas testovania je objektívne. Okno poskytujúce informácie o chybách by vyzeralo podobne ako je zobrazené na obrázku č. 15:



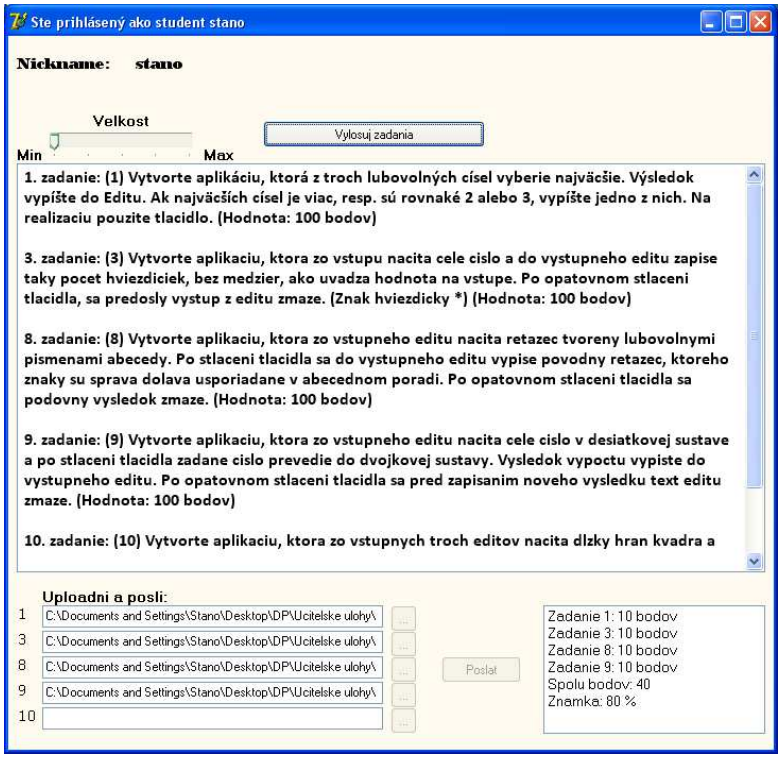
Obr. 15 Zoznam chybových stavov

4.14 Výsledky práce a testovanie aplikácie

Výsledkom našej práce je program, ktorý umožňuje testovať študentove aplikácie v predmete „Programovanie“ s využitím vývojového prostredia Delphi. Testovacia aplikácia podľa nás spĺňa požadovanú funkcionalitu, to znamená obsahuje jednoduché užívateľské prostredie, je umožnené uchovávanie údajov z jednotlivých pokusov testovania, vyučujúci môže editovať zadania, avšak zatiaľ len textovú podobu zadaní. Na úplnú editáciu úloh je potrebný zásah do zdrojového kódu, nakoľko nie je jednoduché vytvárať testovacie prípady pre nové úlohy bez prístupu k zdrojovému kódu. Aplikácia automaticky vyhodnocuje zadania, ktoré boli odoslané študentom, pričom sa zároveň vizuálne zobrazujú výsledky študentovi a po automatickom zapísaní do databázy je možné ich prezeranie pedagógom. Pre slabšie vidiacich je v textovom poli možnosť zväčšiť text zadaní.

Predpokladajme, že užívateľ, ktorý otvoril aplikáciu je už registrovaný, potom po spustení programu a prihlásení sa mu zobrazí formulár, v ktorom sa mu po

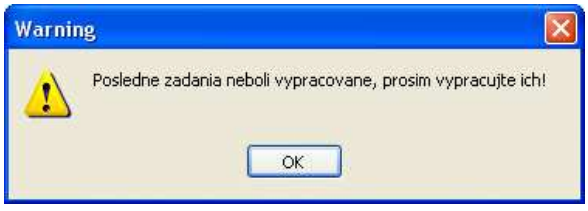
vylosovaní zadaní zobrazia zadania otázok. Ak zadania vypracuje a odošle na kontrolu do systému, formulár môže obsahovať nasledovné údaje, ktoré vidíme na obrázku č. 16:



Obr. 16 Ukážka funkčnosti aplikácie

Z obrázku vieme konštatovať, že študent dostal po vygenerovaní zadania s očíslovaním 1, 3, 8, 9 a 10. Otázky sa zoradili v vzostupnom poradí. Študent odoslal z nejakého dôvodu na kontrolu len zadania 1, 3, 8 a 9. Po stlačení tlačidla „*Poslať*“ sa vyhodnotili úlohy nasledovne: Prvé štyri zadania splnil študent na plný počet bodov, z toho vyplýva, že všetky testovacie prípady dosiahli korektné výsledky počas testu. Posledná úloha nebola odoslaná, preto systém vyhodnotil túto úlohu počtom bodov 0. To znamená, výsledný počet bodov je 40 bodov z maximálneho počtu 50 bodov, percentuálne vyjadrenie úspešnosti je 80%.

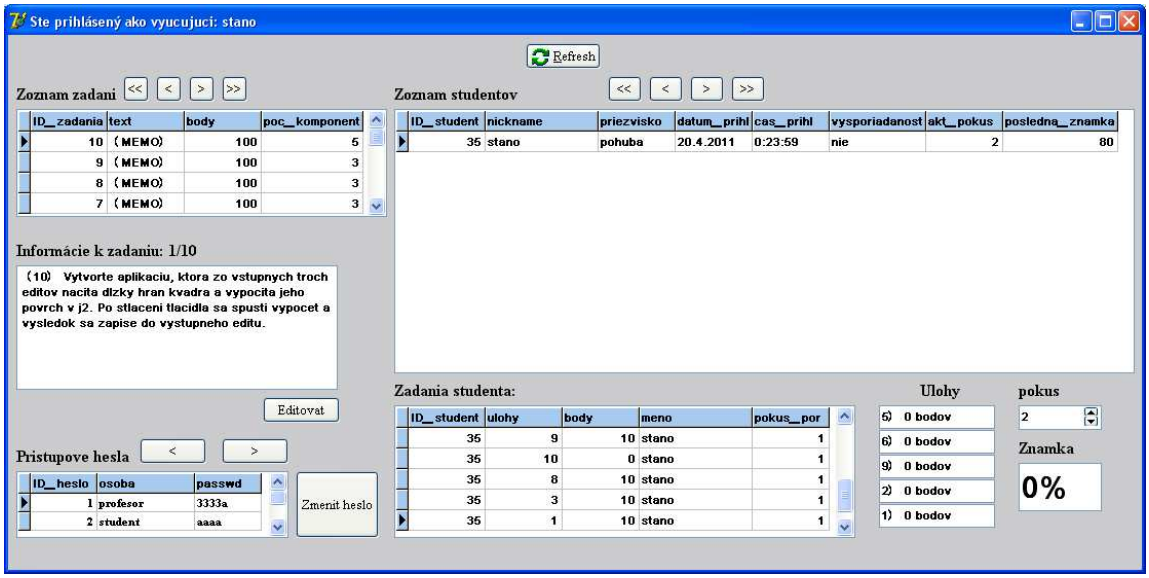
Ak by chcel študent podvádzať alebo by nechtiac zavrel by okno po vylosovaní, pri ďalšom prihlásení mu systém ohlási nasledovné upozornenie, ako vidíme na obrázku č. 17:



Obr. 17 Správa s upozornením

Systém si pamätá aj po vypnutí vylosované zadania každého študenta, a pri prihlásení mu ich zobrazí do textového poľa.

Po zapísaní údajov do databázy, môže pedagóg skontrolovať výsledky študentov, ktoré sa zobrazia v nasledujúcom formulári zobrazenom na obrázku č. 18:



Obr. 18 Formulár s hodnotením riešených zadaní

V ľavej časti vidíme spomínanú časť formulára, kde je možná zmena textovej podoby zadaní a zmena prístupového hesla do aplikácie. V pravej časti máme zoznam študentov, ktorý použili aplikáciu a testovali svoje vedomosti. V tabuľke je jediný užívateľ s *nickname* „*stano*“, ktorý už vypracoval zadania dvakrát, z toho poslednýkrát mu systém vyhodnotil zadania bodovým hodnotením 0 bodov pre každé vylosované zadanie. V tabuľke formulára na obrázku č. 18 vidíme, že pri poslednom losovaní otázok je hodnota v stĺpci „*vysporiadanosť*“ nastavená na hodnotu „*nie*“. Dôsledkom tohto stavu je práve fakt, že študent zadania, ktoré dostal, nevypracoval a okno s testovacími úlohami zatvoril. Na lepšiu kontrolu máme možnosť pozrieť čas a dátum posledného prihlásenia konkrétneho študenta. Aktuálny pokus testovania študenta „*stano*“ je 2. Pred časom vykonal test, ktorého bodové hodnotenie by sa nám zobrazilo po kliknutí na riadok v tabuľke. V spodnej tabuľke napravo na obrázku č. 18 máme možnosť vidieť všetky zadania jednotlivo, pričom ku každému je vyhodnotené bodové hodnotenie, očíslovanie úlohy a pokus.

Napravo od tabuľky vo formulári sú zobrazené hodnoty bodov z aktuálne označeného pokusu. Percentuálna hodnota úspešnosti sa pri akejkoľvek zmene prepočíta na aktualizované hodnoty.

Záver

Vo všeobecnosti sa často kladie malý dôraz na dôležitosť testovania v spojitosti s tvorbou programových aplikácií. Testovanie je proces slúžiaci hlavne k identifikácii skrytých nedostatkov a chýb, ktoré práve vďaka testovaniu môžeme včas odstrániť. V našej diplomovej práci sme sa zaoberali problematikou testovania softvérových aplikácií z viacerých hľadísk, nakoľko testovanie sa stáva v súčasnej dobre neoddeliteľnou súčasťou vývoja nie len zložitých softvérových projektov, ale aj jednoduchších systémov. Len dokonalým testovaním je možné dosiahnuť zhodu s požiadavkami a kvalitu aplikácie.

Teoretická časť práce bola venovaná súčasným trendom a metódam testovania. Z dôvodu veľkého množstva testovacích metód a nástrojov sme si vybrali len niektoré z nich a tie sme porovnali z viacerých uhlov pohľadu, vysvetlili sme výhody a nevýhody pri ich používaní. Taktiež sme v krátkosti objasnili možnosti výberu testovacích nástrojov a ich cenovú dostupnosť pre verejnosť i špecializované inštitúcie zaoberajúce sa výhradne testovaním softvérových aplikácií. Vo všeobecnosti sme rozobrali dôležité metódy a spôsoby testovania aplikácií, bez ohľadu na vývojové prostredie, v ktorom boli aplikácie vytvorené a následne testované. V neskorších podkapitolách sme zaostrili stredobod nášho záujmu na testovanie vo vývojovom prostredí Delphi. Spomenuli a na názornej ukážke sme si vysvetlili spôsob a postupnosť krokov testovania v tomto vývojom prostredí pomocou automatizovaného nástroja framework *DUnit*.

Proces testovania sme sa pokúsili implementovať do školského prostredia v predmete Programovanie. Najdôležitejšou časťou teoretickej časti práce bolo vytvorenie návrhu procesu automatizovaného testovania študentov na stredných, či vysokých školách. Do školského prostredia sme sa rozhodli integrovať testovacie metódy z dôvodu snahy zvýšiť efektívnosť, objektívnosť, jednoduchosť získavania obrazu vedomostí študentov a evidencie výsledkov študentov z jednotlivých testovacích pokusov. Výsledkom počítačového spracovania vedomostí je uľahčenie práce so zadávaním a vytváraním testov, výrazné skrátenie času potrebného na kontrolovanie zdrojových kódov pedagógmi a sprehľadnenie celej činnosti testovania od zadania až po hodnotenie. Táto vízia položila základ pre vytvorenie automatizovaného testovacieho softvéru určeného práve na efektívne získavanie informácií o vedomostiach študentov z predmetu programovanie s využitím vývojového prostredia Borland Delphi. Chceli

sme navrhnúť a vytvoriť systém, ktorý by sa líšil od klasických spôsobov testovania, ktoré sa bežne používajú na školách. Náš plán sa nám podarilo realizovať. Existuje množstvo nástrojov, ktoré slúžia na vytváranie, náhodné generovanie, automatickú kontrolu a evidenciu testov. Absencia testovacích softvérov na kontrolu programovaných aplikácií v danom programovacom prostredí nás viedla k riešeniu tejto situácie. Našim cieľom bolo používaním takéhoto nástroja motivovať a viesť študentov k logickému, algoritmickému mysleniu a používaniu stanovených pravidiel, na ktorých sa vopred vyučujúci so študentmi dohodne.

Problematiku testovania aplikácií sme realizovali v praktickej časti a vytvorili sme testovací online systém na testovanie aplikácií pre účely testovania študentov na školách v predmete Programovanie. Náš testovací systém sme konštruovali tak, aby testované aplikácie vnímal ako „čiernu skrinku“, to znamená, že pri uskutočňovaní testovania náš systém nepotrebuje mať k dispozícii zdrojový kód aplikácie, ale stačí mu len výsledný spustiteľný *EXE* súbor výslednej aplikácie.

Vytvorením softvéru určeného na testovanie aplikácií sme potvrdili fakt, že testovanie aplikácií je možné uplatniť aj v školstve. V súčasnosti známe metódy testovania sú určené zväčša pre veľké softvérové projekty, pri ktorých býva obvykle k dispozícii aj zdrojový kód a testujú sa buď všetky programové vetvy alebo len najdôležitejšie programové časti. Ak je zdrojový kód veľmi obsiahly, mohlo by takéto testovanie zabrať veľmi veľa času, preto sme zvolili spôsob, pri ktorom nás nezaujíma žiadna programová časť ani štruktúra zdrojového kódu, ale zaujímame sa len o výsledok spracovania vstupných hodnôt, resp. o výstupné hodnoty z aplikácie. Naša vytvorená aplikácia teda nie je určená na testovanie kvality zdrojového kódu, pretože pri prevode vstupných hodnôt na výstupné nie je zaručené, že pri behu programu sa použijú všetky vetvy a podmienky zdrojového kódu. Študentove programy testujeme z hľadiska splnenia požadovanej funkcionality a korektnosti poskytovaných výstupov.

Myslíme si, že našu softvérovú aplikáciu by sme mohli zaradiť k efektívnym nástrojom automatizovaného testovania. Veľkým problémom automatizovaných systémov je spôsob, akým prebieha komunikácia s testovanou aplikáciou. Exituje veľa systémov pracujúcich prostredníctvom vstupno – výstupných súborov. Náš program má tú výhodu, že k nadviazaniu komunikácie mu stačí len *EXE* súbor, ktorého absolútna adresa je nám známa.

Podľa prvotných plánov a cieľov sme nakoniec jadro algoritmu aplikácie Online testovanie vyriešili tak, že algoritmus testovacieho programu je založený na komunikácii dvoch aplikácií prostredníctvom Windows *API* funkcií implementovaných v prostredí programovacieho jazyka Delphi. Po otvorení aplikácie sme nadviazali komunikáciu prostredníctvom identifikátora okna, ktorý sme získali prehľadaním všetkých spustených okien v systéme pomocou identifikátora procesu. Po tom ako sme získali identifikátor okna, môžeme aplikácii posielať správy, v ktorých sme definovali požiadavky, ktoré sa majú v danom okne vykonať. Jedná sa o zapisovanie, čítanie údajov a vykonanie udalosti na tlačidlo. Výmenou informácií medzi aplikáciami sa posudzuje správnosť získaných údajov a následne sa informácie uchovávajú pre potreby neskoršieho použitia. Vyučujúci má možnosť jednoducho prezerať výsledky všetkých študentov aj s hodnotením. Dosiahli sme vyšší komfort v oblasti evidencie známok študentov a jednoduchší prístup.

Snahou do budúcnosti je rozšírenie funkcionality programu, zvýšenie jeho odolnosti voči chybám, a v neposlednom rade sa pokúsiť o jeho implementáciu v školstve. Zistili sme, že testovací softvér výrazne skracuje dobu potrebnú na opravovanie študentských prác. Drobnými úpravami by bolo možné testovací softvér implementovať aj v praxi na testovanie študentov. Aplikácia je prepojená s databázovým systémom, ktorý uchováva všetky potrebné informácie o jednotlivých testovacích pokusoch študenta. Databáza zatiaľ nie je umiestnená na webovom serveri, ale pracuje zatiaľ len v offline móde prostredníctvom databázového klienta. Názov Online testovanie aplikácií môžeme však spokojne ponechať, pretože tento klient sa správa akoby databáza bola umiestnená na inom počítači v sieti internet. Ak by sa niekedy systém implementoval do prevádzky, je možné jednoduchými zmenami uskutočniť komunikáciu aplikácie a databázového systému cez internetovú sieť.

Zoznam použitej literatúry

BIELIKOVÁ, M. 2000. *Softvérové inžinierstvo: Princípy a manažment*. Slovenská technická univerzita v Bratislave : Vydavateľstvo STU v Bratislave, 2000. 220 s. [cit. 2010-05-27]. ISBN 80-227-1322-8

BIELIKOVÁ, M. – NÁVRAT, P. a kol. 2007. *Štúdie vybraných tém softvérového inžinierstva.* 3. Slovenská technická univerzita v Bratislave : Vydavateľstvo STU v Bratislave, 2007. 228 s. [cit. 2010-09-15]. ISBN 978-80-227-2701-3.

BUCHALCEVOVÁ, A. - KUČERA, J. 2008. Hodnocení metodik vývoje informačních systémů z pohledu testování. In: *Systémová integrace*. [online]. 2008, roč. 15, č. 2, s. 42–54. [cit. 2010-08-09]. Dostupné na internete: http://nb.vse.cz/~buchalc/clanky/testovani.pdf. ISSN 1210-9479

CANTÚL, M. 2003. *Myslíme v jazyku Delphi 7.* Grada, 2003. 580 s. ISBN 8024706946

CÁPAY, M. 2007. *ANALÝZA ELEKTRONICKÉHO TESTOVANIA V CMS MOODLE*. [online]. UKF v Nitre. 2007. [cit. 2010-12-01]. Dostupné na internete: http://www.ki.fpv.ukf.sk/projekty/kega_3_4029_06/iski2007/papers/Capay.pdf

ČERMÁK, M. 2010a. *Testování SW*. [online]. 2010. [cit. 2011-01-20]. Dostupné na internete: http://www.cleverandsmart.cz/testovani-sw/

ČERMÁK, M. 2010b. *Black box test*. [online]. 2010. [cit. 2011-01-20]. Dostupné na internete: http://www.cleverandsmart.cz/black-box-test/

ČERMÁK, M. 2010c. *White box test*. [online]. 2010. [cit. 2011-01-20]. Dostupné na internete: http://www.cleverandsmart.cz/white-box-test/

ČERMÁK, M. 2010d. *Grey box test*. [online]. 2010. [cit. 2011-01-20]. Dostupné na internete: http://www.cleverandsmart.cz/grey-box-test/

ČERMÁK, M. 2010e. *Způsoby testování*. [online]. 2010. [cit. 2011-01-20]. Dostupné na internete: http://www.cleverandsmart.cz/zpusoby-testovani/

ČERMÁK, M. 2010f. *Test status report*. [online]. 2010. [cit. 2011-01-20]. Dostupné na internete: http://www.cleverandsmart.cz/test-status-report/

ČERMÁK, M. 2010g. *Automatizované testy aplikácí typu klient-server*. [online]. 2010. [cit. 2011-01-12]. Dostupné na internete: http://www.cleverandsmart.cz/automatizovane-testy-aplikaci-typu-klient-server/

DOCSTOC, 2010. *Vytvorenie Win32 klienta MySQL v Delphi*. [online]. 2010. [cit. 2010-09-25]. Dostupné na internete: http://www.docstoc.com/docs/23984286/Vytvorenie-Win32-klienta-MySQL-v-Delphi

EMBARCADERO. 2010. *DUnit Overview.* [online]. aktualizované 1.5.2010. [cit. 2010-09-05].Dostupné na internete: http://docwiki.embarcadero.com/RADStudio/2010/en/DUnit_Overview

GLONČÁK, J. a kol. 2000. *Etika softvérového inžiniera a softvérové procesy.* [online]. Bratislava : SOWA, 2000. [cit. 2010-03-06]. Dostupné na internete: http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/kniha/2000/sowa.pdf.

GOLKO, K. 2010. *Automated testing with DUnit.* [online]. 2010. [cit 2011-02-20] Dostupné na: http://www.howtodothings.com/computers/a928-automated-testing-with-dunit.html

HELD, Ľ. – ŽOLDOŠOVÁ, K. 1999. *Vyučovacie metódy a techniky*. Trnava : Trnavská univerzita v Trnave, november 1999. 73 s. [cit. 2010-09-18]. ISBN 80-88774-51-9

HERBST, D. 2010a. *Základy automatizace testování*. [online]. 2010. [cit. 2011-01-12]. Dostupné na internete: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=40:zaklady-automatizace-testovani&catid=3:zaklady&Itemid=11

HERBST, D. 2010b. *Druhy testování v procesu vývoje SW*. [online]. 2010. [cit. 2011-01-12]. Dostupné na internete: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11

JANEK, M. 2007. *Nástroj pro podporu testy řízeného vývoje*: bakalárska práca. Praha: České vysoké učení technické v Prahe, 2007. 38 s.

KADLEC, V. 2001. *Umíme to s Delphi – 1. Díl.* [online]. 2001. [cit. 2010-09-05]. Dostupné na internete: http://www.zive.cz/clanky/umime-to-s-delphi--1-dil/sc-3-a-30959/default.aspx

MACHOVÁ, J. 2004. *Začíname s Delphi*. Prešov : Metodicko-pedagogické centrum v Prešove, 2004. 49 s. [cit. 2011-01-08]. ISBN 80-8045-353-5

MICROSOFT, 2011a. *SetWindowText Function.* [online]. [cit. 2011-02-10]. Dostupné na internete: http://msdn.microsoft.com/en-us/library/ms633546%28v=vs.85%29.aspx

MICROSOFT, 2011b. *GetWindowText Function.* [online]. [cit. 2011-02-10]. Dostupné na internete: http://msdn.microsoft.com/en-us/library/ms633520%28v=vs.85%29.aspx

MICROSOFT, 2011c. [online]. [cit. 2010-09-23]. Dostupné na internete: http://msdn.microsoft.com

MOLINARO, A. 2009. *SQL: Kuchařka programátora.* Computer Press, 2009. 576 s. ISBN 9788025126172

MÜLLER, T. a kol. 2007. *Certifikovaný tester*: Učebné osnovy pre základný stupeň. [online]. International Software Testing Qualifications Board, 15.3.2008. [cit. 2010-11-05]. Dostupné na internete: http://www.castb.org/tiki-download_file.php?fileId=164

ORAVEC, V. 2010. IW: Outsourcing testovania softvéru. In: *INFOWARE*. [online]. 10/2010, s.28-30. [cit. 2011-01-22]. Dostupné na internete: http://www.itnews.sk/tituly/infoware/free-clanky/2010-11-19/c136875-iw-outsourcing-testovania-softveru

PATTON, R. 2002. *Testování softwaru*. Computer Press, 2002. 328 s. ISBN 8072266365

PETLÁK, E. 2004. *Všeobecná didaktika.* Bratislava : Vydavateľstvo IRIS, 2004. 311 s. [cit. 2010-11-13]. ISBN 80-89018-64-5

PIRKL, J. 2002. *Komponenty v Delphi*. Computer Press, 2002. 456 s. ISBN 8072267469

SHELDON, R. 2005. *SQL: Začínáme programovat*. Grada, 2005. 500 s. ISBN 8024709996

TRNKA, A. 2008. Výber modelu testovania aplikácií dátových skladov. In: *Journal of Information Technologies*. [online]. 2008, č. 1 [cit. 2010-05-10]. Dostupné na internete: http://ki.fpv.ucm.sk/casopis/clanky/01-trnka.pdf. ISSN 1337-7469

WATTS, W. 1999. *DUnit Xtreme testing for Delphi.* [online]. 1999. [cit. 2010-11-20]. Dostupné na internete: http://dunit.sourceforge.net/README.html

WILLIAMS, L. 2006a. *Testing Overview and Black-Box Testing Techniques.* [online]. 2006. [cit. 2010-10-18]. Dostupné na internete: http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf

WILLIAMS, L. 2006b. *White-Box Testing.* [online]. 2006. [cit. 2010-10-18]. Dostupné na internete: http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf

Prílohy

Príloha 1: Elektronický nosič CD

Priložené CD obsahuje:

- Diplomová práca vo formáte PDF
- Spustiteľný súbor EXE aplikácie vytvorenej v Delphi
- Vyexportovaná databáza *localhost.sql*
- Priečinok so spustiteľnými EXE súbormi vypracovaných zadaní