SLOVENSKÁ TECHNICKÁ UNIVERZITA

Fakulta chemickej a potravinárskej technológie

Radlinského 9, 812 37 Bratislava

Diplomová práca

Toolbox pre testovanie programov v prostredí MATLAB

Vypracoval: **Bc. Elena SÚKENNÍKOVÁ**

Vedúci diplomovej práce: **Ing. Michal KVASNICA**

Bratislava

2007

S T U
F C H P T

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta chemickej a potravinárskej technológie

Ústav: **Ústav informatizácie, automatizácie a matematiky**

Oddelenie: **Oddelenie informatizácie a riadenia procesov**

Číslo: **8/ÚIAM/2007**

Vec: **Zadanie diplomovej práce**

Meno a priezvisko študenta: **Bc. Elena Súkenníková**

Meno a priezvisko vedúceho diplomovej práce: **Ing. Michal Kvasnica**

Meno a priezvisko konzultanta diplomovej práce: **Ing. Ľuboš Čirka, PhD.**

Názov diplomovej práce:

Toolbox pre testovanie programov v prostredí MATLAB

Termín odovzdania diplomovej práce: **19. mája 2007**

Diplomová práca sa odovzdáva v 3 exemplároch vedúcemu ústavu – oddelenia.

Bratislava, 12. februára 2007

Ing. Michal Kvasnica
vedúci diplomovej práce

doc. Dr. Ing. Miroslav Fikar
vedúci oddelenia

Slovenská technická univerzita v Bratislave Fakulta chemickej a potravinárskej technológie dekanát - 5 -

doc. Dr. Ing. Miroslav Fikar
riaditeľ ústavu

prof. Ing. Dušan Bakoš, DrSc.
dekan

ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že diplomovú prácu som vypracovala samostatne, na základe zdrojov uvedených v literatúra, pod vedením vedúceho diplomovej práce a vedomostí získaných počas štúdia.

V Bratislave 18. 05. 2007 .....

podpis

POĎAKOVANIE

Touto cestou sa chcem poďakovať vedúcemu diplomovej práce
Ing. Michalovi Kvasnicovi, za odbornú pomoc, cenné rady a trpezlivosť, ktorú mi venoval počas vypracovania tejto práce.

Abstrakt

Diplomová práca sa zaoberá návrhom a implementáciou frameworku na testovanie softvéru v prostredí MATLAB. Daný framework sa používa pri vývoji programov pomocou metódy Test – Driven Development. Uplatnenie má pri práci viacerých vývojárov na spoločnom projekte. Umožňuje im vytvárať, spravovať a spúšťať balík testov, navrhnutý na overovanie funkčnosti, správnej činnosti a výkonnosti testovaného softvéru. Taktiež im umožňuje rýchlo určiť, či ich práca sa negatívne neovplyvňuje.

V práci je podrobne popísaný scenár požitia testovacieho frameworku, vytvárania testových súborov, ako aj rôzne možnosti spustenia a filtrácie získaných výsledkov.

Abstract

This diploma thesis deals with the design and the implementation of framework for testing software in MATLAB environment. Given framework is used at program development throught the method Test – Driven Development. The testing framework applied at work of more developers working on common project. The testing framework allows to create, administrate and run the package of tests, which was design for verification of utility, correct activity and perfomance of tested software. It also allow fast allocate, whether their work is not negatively affecting.

There is narrowly described scenario of usage of testing framework, generating test files as well as various possibilities of running and filtrating obtained results in this work.

Obsah

Zoznam obrázkov

Úvod

V súčasnosti sa nachádzame v období expanzie informačných technológií. Ich uplatnenie v spoločnosti zo dňa na deň narastá. Počítače sa stávajú čoraz viac súčasťou bežného života. Riadia väčšinu procesov okolo nás.

Rozvoj technológií je úzko spätý s tvorbou nových aplikácií vytváraných vývojármi. Vývojári pri vytváraní programov postupujú na základe určitých metód. Jednou z metód vývoja programov je Test - Driven Development. A práve cieľom tejto diplomovej práci je návrh a implementácie testovacieho frameworku, používaného pri vývoji programov pomocou tejto metódy. Konkrétne nám testovací framework umožňuje vytvárať, spravovať a spúšťať balík testov, ktorý bol navrhnutý na overovanie funkčnosti, správnej činnosti a výkonnosti testovaného softvéru.

V kapitole 1 uvádzam prehľad už známych testovacích frameworkov používaných nielen v prostredí MATLBu, ale aj iných programovacích jazykoch.

V kapitole 2 teoreticky popisujem základné pojmy, ako softvérové testovanie, unit testing a metódu Test – Driven Development. Taktiež popisujem inštaláciu testovacieho frameworku.

V kapitole 3 sa zaoberám vlastnou tvorbu. Podrobne popisujem scenár požitia testovacieho frameworku, vytvárania testových súborov, ako aj rôzne možnosti spustenia a filtrácie získaných výsledkov.

1 Existujúce aplikácie testovacích frameworkov

Na predstave extrémneho programovania a Test – Driven Development bolo v posledných rokoch vypracovaných niekoľko testovacích frameworkov u väčšiny programovacích jazykov. Podstata všetkých týchto frameworkov siaha až do SUnit-u, testovacieho frameworku jazyka Smoltalk vytvoreného v roku 1994. SUnit je matkou všetkých testovacích frameworkov, pričom jeho základný kameň vytvára metodika Test – Driven Development (TDD). Následne sa táto koncepcia preniesla v roku 1998 do programovacieho jazyka Java a vytvoril sa JUnit. Dnes už existuje celá rada týchto frameworkov, napr. CppUnitpre C++, PyUnit pre Python alebo NUnit pre .NET framework [1].

V poslednom roku bolo vyvinutých niekoľko testovacích frameworkov pre programovací jazyk MATLAB. Sú to:

- MATUnit od Thimothy Wall;
- MUnit od Brada Phelana;
- mlUnit od Thomasa Dohmkea;
- munit od Davida Leglanda;
- Unit Testing Tools od Kit Ng [1].

Všetky uvedené testovacie frameworky sú založené na metóde TDD.

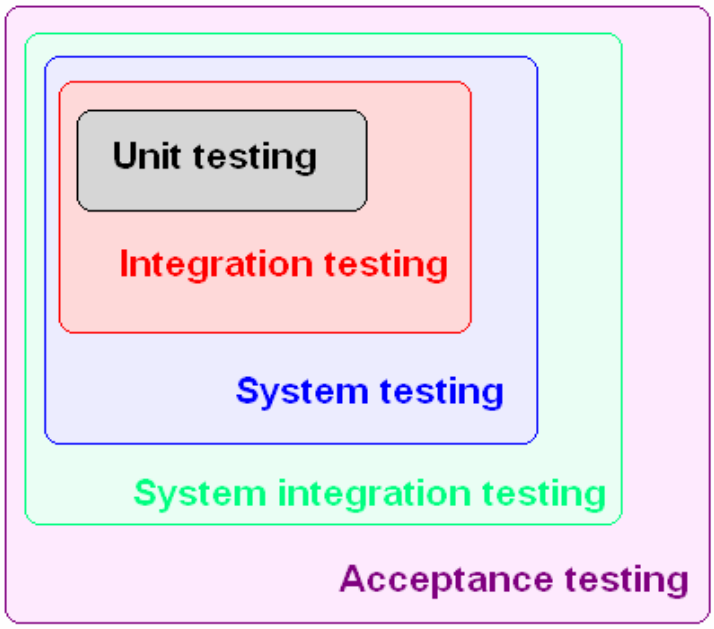
2 Úvod do problematiky

1.1 Základné pojmy

2.1.1 Softvérové testovanie

Softvérové testovanie je proces používaný na pomoc pri určovaní správnosti, úplnosti, bezpečnosti a kvality vývoja počítačového softvéru. Softvérové testovanie sa uskutočňuje na 5 úrovniach:

1. testovanie jednotky (unit testing) – testujeme minimálnu softvérovú jednotku;
2. testovanie úplnosti (integration testing) – odhalia sa defekty medzi článkami a interakcie medzi integrovanými jednotkami;
3. testovanie systému (system testing) – v ktorom softvér je integrovaný do celkového produktu a testovaním sa zisťuje, či všetky požiadavky sú splnené;
4. systémový test úplnosti (system integration testing);
5. prijímacia skúška (Acceptance testing) [2].



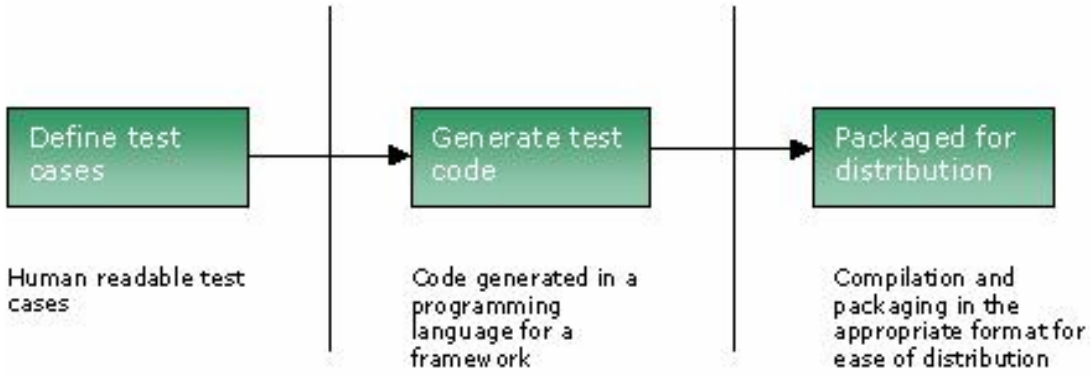
Obr. 2.1: Vzťah medzi jednotlivými úrovňami softvérového testovania.

2.1.2 Test-Driven Development

Test - Driven Development (TDD) je vytváranie softvérových postupov, pri ktorých sa najskôr napíšu samostatné testy a následne kód, ktorý spĺňa naše požiadavky. Test - Driven Development nám dáva rýchlu spätnú informáciu o tom, či daný kód bude spĺňať požiadavky určené užívateľom. Zo začiatku TDD bol zahŕňaný do extrémneho programovania, ale v súčasnosti je hlavný záujem o jeho vlastné práva [3].

TDD nie je iba metódou testovania, ale hlavne metódou návrhu softvéru [3].

Ideou TDD je definovanie štruktúry a jednoduchého jazyka testu, ktorý je transformovaný na zdrojový kód a následne kompilovaný a balený na distribúciu. Test je jeden z najdôležitejších častí TDD [4].



Obr. 2.2: Grafické znázornenie TDD [4].

Cyklus TDD [3]:

- Vytvorenie testu. Každá nová požiadavka začína napísaním testu. Pred napísaním každého testu musí mať vývojový projektant (developer) presne a zrozumiteľne zadefinované požiadavky.

- Spustenie všetkých testov a následná kontrola vytvoreného testu. Toto potvrdzuje správnosť simulačného programu, ak si nový test nevyžaduje žiadnu úpravu a ani pridanie nového kódu. Nový test taktiež môže byť chybný s očakávaných dôvodov. V tomto kroku sa test testuje sám, a to odmietavo.

- Úprava kódu. V ďalšom kroku napíšeme dáky kód, ktorý umožní, aby test prešiel. Nepíšeme žiaden nový kód, dochádza len k minimálnym úpravam kódu.

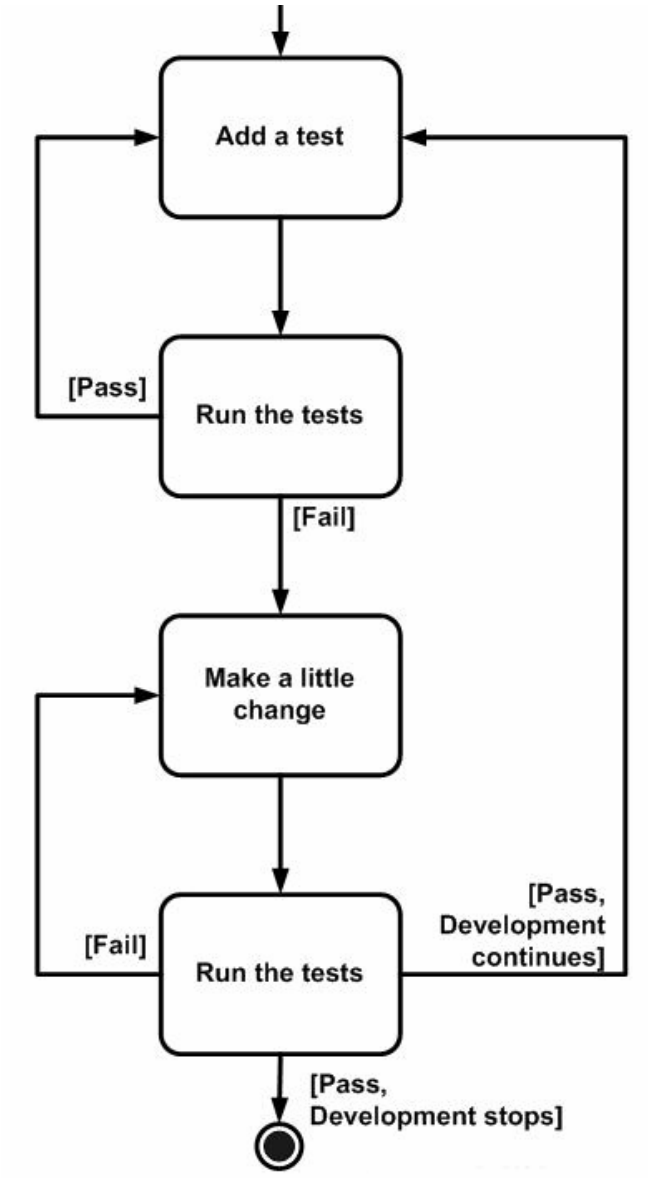
- Riadenie automatizovaných testov. Ak všetky skúšobné prípady prebehnú, programátor si môže byť istý, že kód spĺňa minimálne všetky testované požiadavky.

- Refactor code. Nový kód môže byť upravený ako je potrebné. Opakovaním testovania si vývojový projektant môže byť istý, že refactoring nepoškodí žiadnu existujúcu funkčnosť. Odstránenie opakovania je dôležitým aspektom každého softvérového návrhu.

Refactoring je proces zmeny softvérového systému takým spôsobom, že to nezmení externé chovanie kódu, ale zlepší jeho vnútornú štruktúru [5]. Refactoring je druh reorganizácie, je to usporiadanie kódu pre väčšiu prehľadnosť.

TDD má dve základné pravidlá [6]:

- Nikdy sa nepíšu jednotlivé riadky programu pokiaľ máme neúspešný test.
- Odstrániť opakovania.



Obr. 2.3: Cyklus Test - Driven Development [7].

2.1.3 Unit Testing

Unit testing je dôsledné testovanie funkčnej časti (jednotky) programu. Je vhodný pri tvorbe zložitejších programov a pri písaní funkcií, ktoré sa ťažko testujú v skutočnej prevádzke [8]. Unit testing je prvou úrovňou pri softvérovom testovaní.

Pod pojem unit testing zahrňujeme nástroje, metodiku a činnosť, ktorých cieľom je overenie správnej funkčnosti zdrojového kódu.

Proces pozostáva s troch fáz, ktoré sú ďalej rozdelené celkom do 8 základných činností [9]:

1. Uskutočnenie testovacieho plánu (zámeru):

- Plánovanie hlavných postupov, nápadov a plánov.
- Určenie vlastností, ktoré majú byť testované.
- Vytriedenie hlavných plánov.

2. Získanie skúšobnej sady:

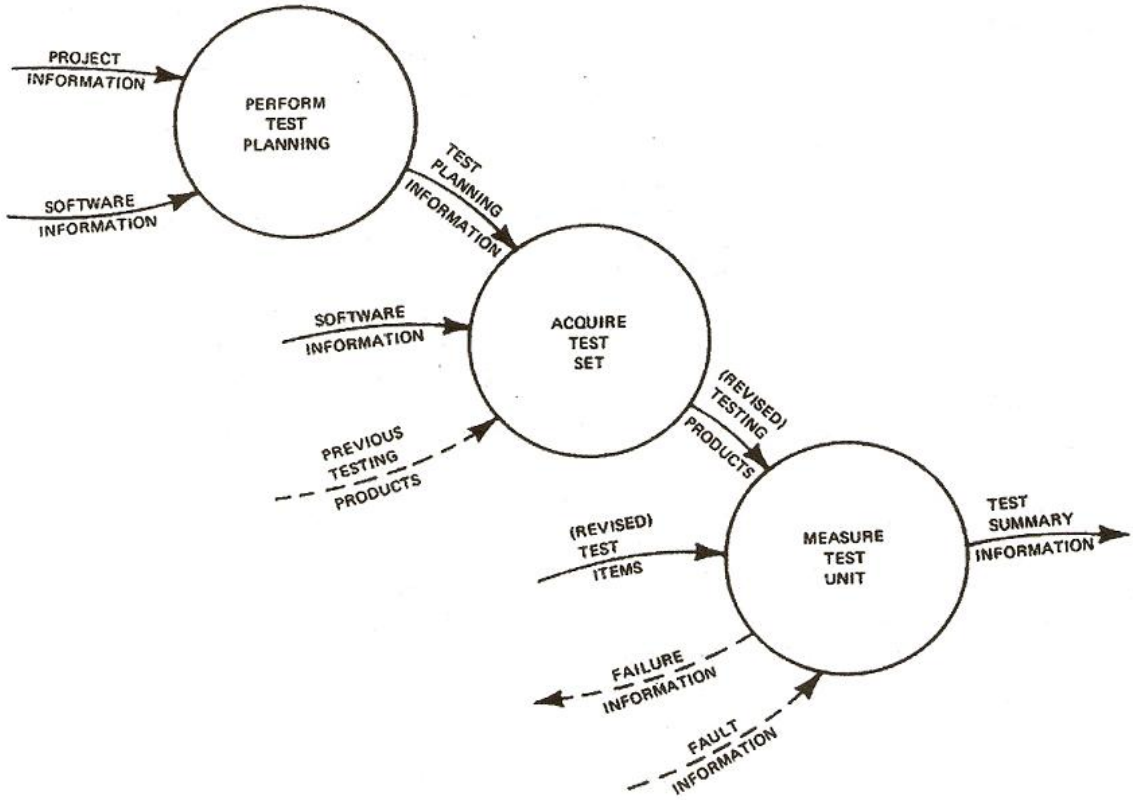
- Návrh sady testov.
- Prevedenie vytriedených plánov a návrhov.

3. Posúdenie testov:

- Výkonnosť testovania.
- Kontrola výsledkov.
- Zhodnotenie testovaného programu.

Najdôležitejšie prúdy informácií sú znázornené na obr. 2.4.

Unit test je test pre konkrétnu, zvolenú jednotku. Je určený pre vývojového projektanta, nie pre konečného užívateľa.



Obr. 2.4: Najdôležitejšie informačné prúdy medzi základnými fázami [9].

1.2 Úvod do testovacieho toolboxu

b2 je testovací framework vytvorený pre pracovné prostredie MATLAB na základe metódy Test –Driven Development.

2.2.1 Inštalácia

Testovací framework je adresár obsahujúci podadresáre a súbory. Je súčasťou priloženého CD pod názvom *b2.zip*. Inštalácia spočíva v nasledujúcich krokoch:

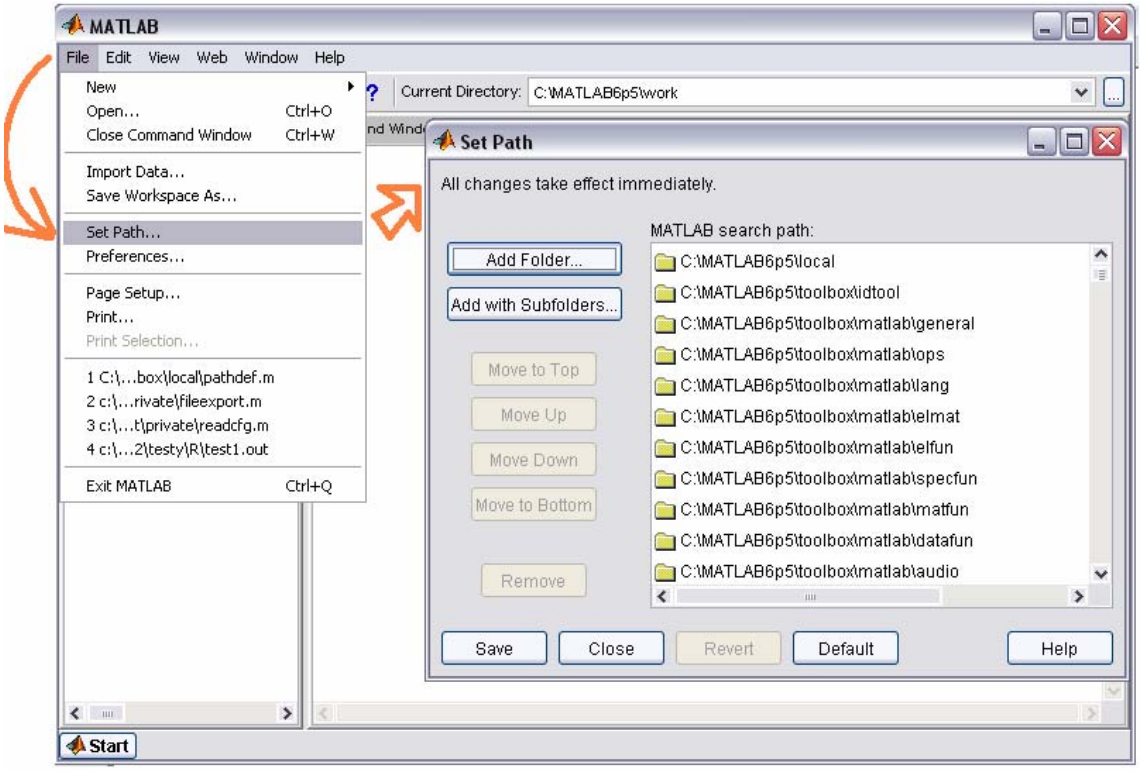
- Dekompresia súboru *b2.zip* do adresára *c:\MATLAB6p5\matlab.*
- Nastavenie vyhľadávacej cesty.
- Konfigurácia.

Nastavenie vyhľadávacej cesty

Cestu možno nastaviť viacerými spôsobmi.

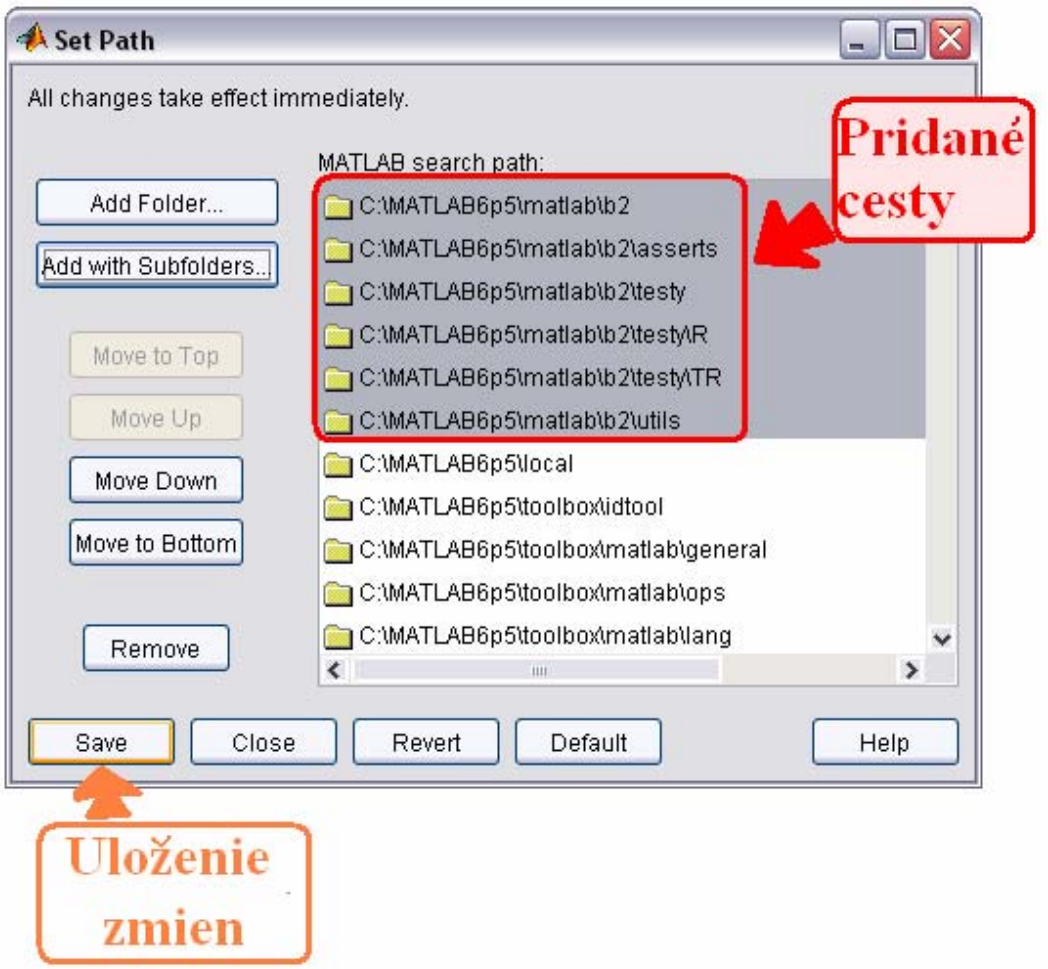
Prvý spočíva v zadefinovaní cesty v m-file *pathdef.m*, do ktorej sa musia explicitne zadať názvy všetkých adresárov a podadresárov.

Druhý spôsob spočíva vo vybratí vyhľadávacej cesty z ponuky *File →Set Path....*



Obr. 2.5: Okno pre nastavenie vyhľadávacej cesty.

Otvorí sa nám okno *Set Path* (obr. 2.5), kde kliknutím na voľbu *Add with Subfolders...* a vyberaním cesty k danému adresáru (*c:\MATLAB6p5\matlab\b2*) nastavíme vyhľadávaciu cestu. Ak sme nastavenie vykonali podľa predchádzajúcich pokynov, zobrazia sa nám v dialógovom okne pridané cesty adresárov a podadresárov testovacieho toolboxu (obr. 2.6). Na úplnosť je potrebné ešte uložiť vykonané zmeny, a to kliknutím na položku *Save.*



Obr. 2.6: Okno po nastavení vyhľadávacej cesty.

Posledný tretý spôsob spočíva zadaní príkazu addpath v príkazovom okne. Syntax príkazu je nasledujúca : „*addpath(genpath('c:\MATLAB6p5\matlab\b2'))*“. Tento spôsob na rozdiel od predchádzajúcich dvoch je len dočasný, po ukončení programe MATLAB sa nám cesta automaticky zruší.

Konfigurácia

Pred samotným spusteným testovacieho frameworku je potrebné v m-file *mbg_config.m* zmeniť premennú *MBG_CONFIG.dirs.results.* Táto premenná predstavuje cestu adresára, do ktorého sa budú ukladať exportované výsledky. V mojom prípade *MBG_CONFIG.dirs.results = 'c:\MatlabFiles\mbg'.*

2.2.2 Odinštalovanie

Odinštalovanie sa uskutočňuje opačným postupom ako inštalácia. Zahŕňa:

- odstránenie vyhľadávacej cesty,
- vymazanie adresáru *b2* z podadresára *c:\MATLAB6p5\matlab.*

Odstránenie vyhľadávacej cesty sa uskutočňuje troma spôsobmi. Prvý spôsob spočíva vo vymazaní vyhľadávacej cesty z m-file *pathdef.m*.

Druhý spočíva z vybrania si voľby *Set Path...* z ponuky *File*. Po otvorení okna *Set Path* je treba označiť cestu a kliknúť na tlačidlo *Remove*. Toto opakujeme pre všetky podadresáre adresáru b2.

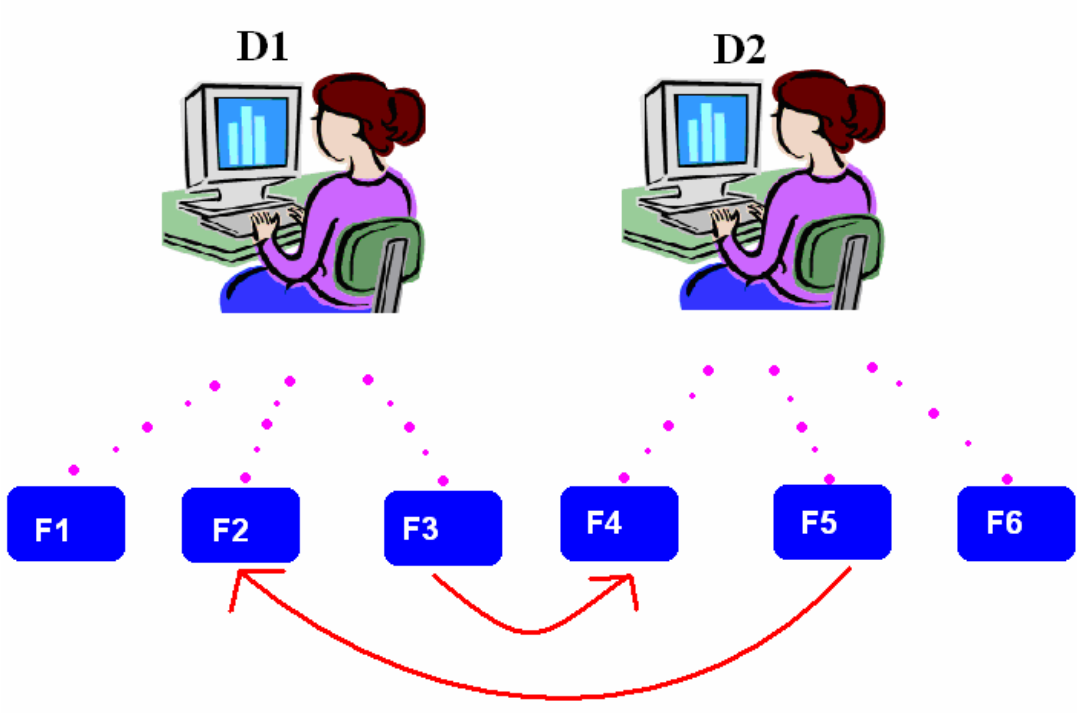
Tretí spôsob spočíva zadaním príkazu *rmpath* v príkazovom okne, pričom odstraňujeme jednotlivé vyhľadávacie cesty samostatne.

3 Testovací framework

3.1 Využitie

Pri písaní programu sa nemôže napísať kód, ktorý by nebol otestovaný, a preto sa najskôr píšu testy. Ich otestovaním vieme presne určiť, či daný kód spĺňa všetky požiadavky, ktoré sú kladené na daný test.

Toolbox pre testovanie programov možno využiť pri spoločnej práci viacerých vývojových projektantov (vývojárov) na jednom programe. Ak sa ich práca navzájom prelína, môžu ľahko zistiť, či ich zmeny pri práci nebudú mať negatívny dopad na prácu ostatných ľudí pracujúcich pri tom istom počítači.



Obr. 3.1: Príklad využitia testovacieho frameworku v praxi.

Ako možno vidieť na obr. 3.1. na vytváraní programu pozostávajúceho zo šiestich funkcií F1 až F6 sa podieľajú dvaja vývojári D1 a D2. Pracujú pri spoločnom počítači a pri práci sa navzájom striedajú. Každý z nich vytvára presne určené funkcie podľa vopred zadaných požiadavok a kritérií, pričom medzi niektorými sú vzájomné prepojenia. Červenými šípkami sú znázornené vzájomné prepojenia medzi jednotlivými funkciami. Pri práci musia obaja dohliadať, či ich vykonaná práca nepoškodí alebo inak negatívne neovplyvní prácu kolegu, a tým neskazia ich spoločnú prácu. Napr. pri úpravách vo funkcii F2 musí developer D1 prihliadať na to,či danou zmenou nespôsobí nefunkčnosť funkcie F5, ktorá vo svojom skripte túto funkciu využíva.

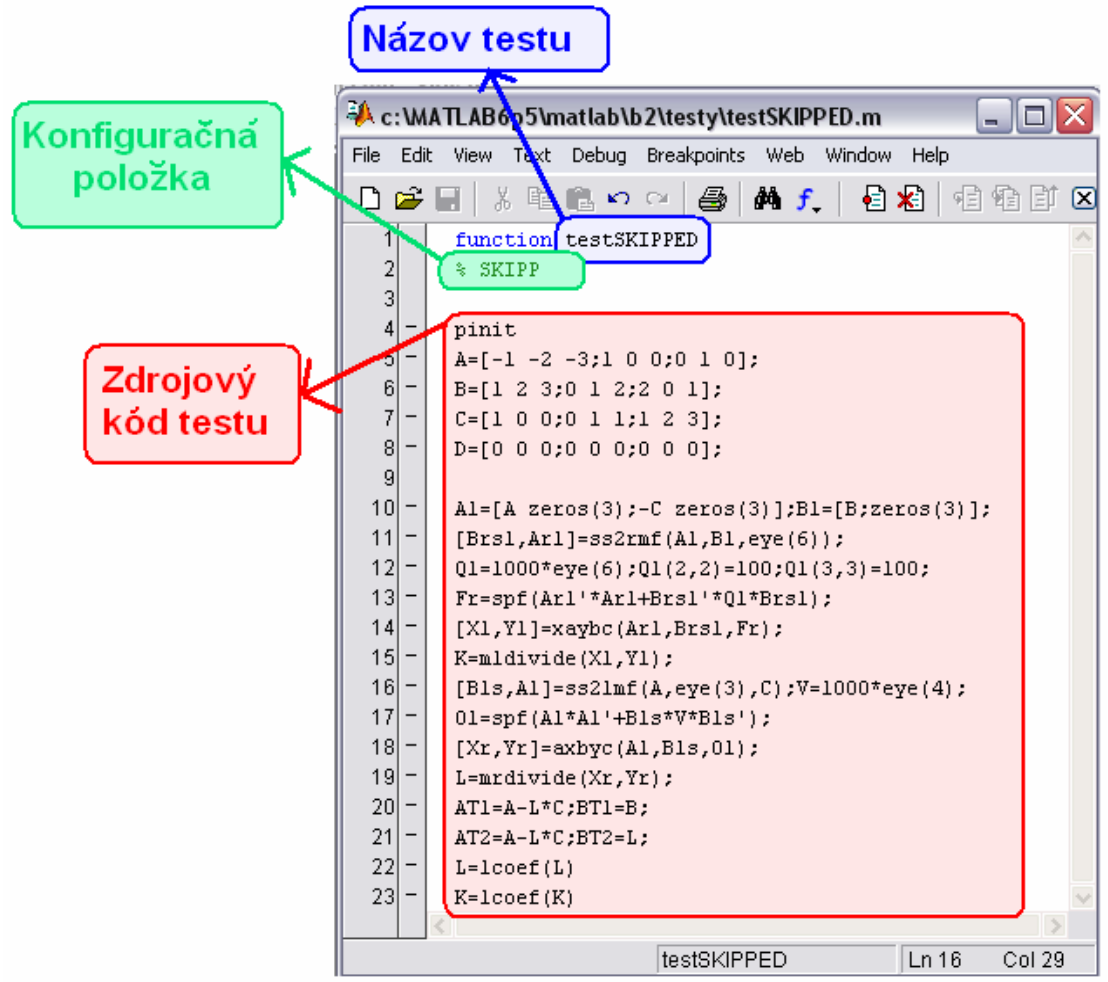
3.2 Vytváranie testov

Pri vytváraní programu sa nemôže napísať kód, ktorý nie je vopred otestovaný, preto treba najskôr napísať test až následne kód. Testy by mali obsahovať taký kód, ktorý by umožňoval otestovanie čo najväčšej časti z kódu vytváranej funkcie. Napr. pri vytváraní kódu sme najskôr vytvorili test, ktorý obsahuje v svojom kóde tri samostatné cykly. Po prebehnutí testovania test vyprodukoval chybové hlásenie v druhom cykle. Toto testovanie nám ale nezistilo, či je chyba aj v tretom cykle, pretože pri chybovom hlásení v druhom cykle, sa nám testovanie ukončilo. Vhodnejšie je vytvoriť viac testov, ktoré by testovali jednotlivé cykly samostatne. Testovanie týchto testov je navzájom nezávislé, a aj keď sa vyskytne chybové hlásenie v niektorom z nich, testovanie pokračuje ďalej v testovaní nasledujúcich testov. Tento spôsob testovania označujeme ako jednotkové testovanie. Týmto spôsobom dokážeme otestovať celý kód na rozdiel od prvého spôsobu testovania. To je hlavný dôvod požitia jednotkového testovania v praxi.

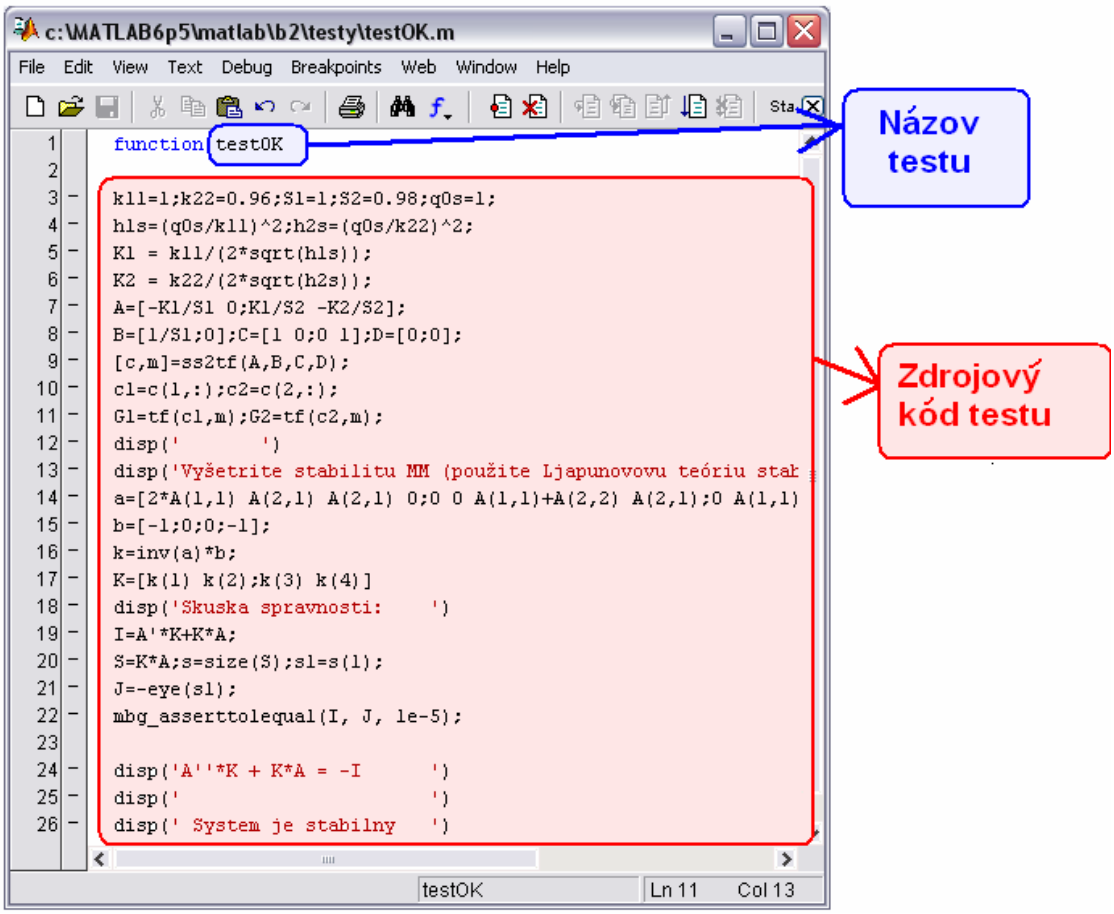
Jednotlivé testy zapisujeme do skriptov. Skripty testov obsahujú:

- názov testu;
- konfiguračnú položku;
- zdrojový kód.

Na obr. 3.2. a obr. 3.3. sú znázornené základné kostry skriptu testu s konfiguračnými položkami a testu bez konfiguračných položiek. V prvom riadku skriptu sa vedľa položky *function* nachádza názov testu. Pod týmto istým názvom s príponou .m je test uložení. V druhom riadku skriptu pod názvom testu sa nachádzajú v komentári konfiguračné položky. A následne po konfiguračných položkách do štvrtého riadku píšeme vlastný kód testu. Konfiguračné položky nie sú povinné. V tom prípade sa pod názvom funkcie vynecháva prázdny riadok a vlastný kód funkcie sa píše do tretieho riadku skriptu.



Obr. 3.2: Základná kostra skriptu testu s konfiguračnými položkami.



Obr. 3.3: Základná kostra skriptu testu bez konfiguračných položiek.

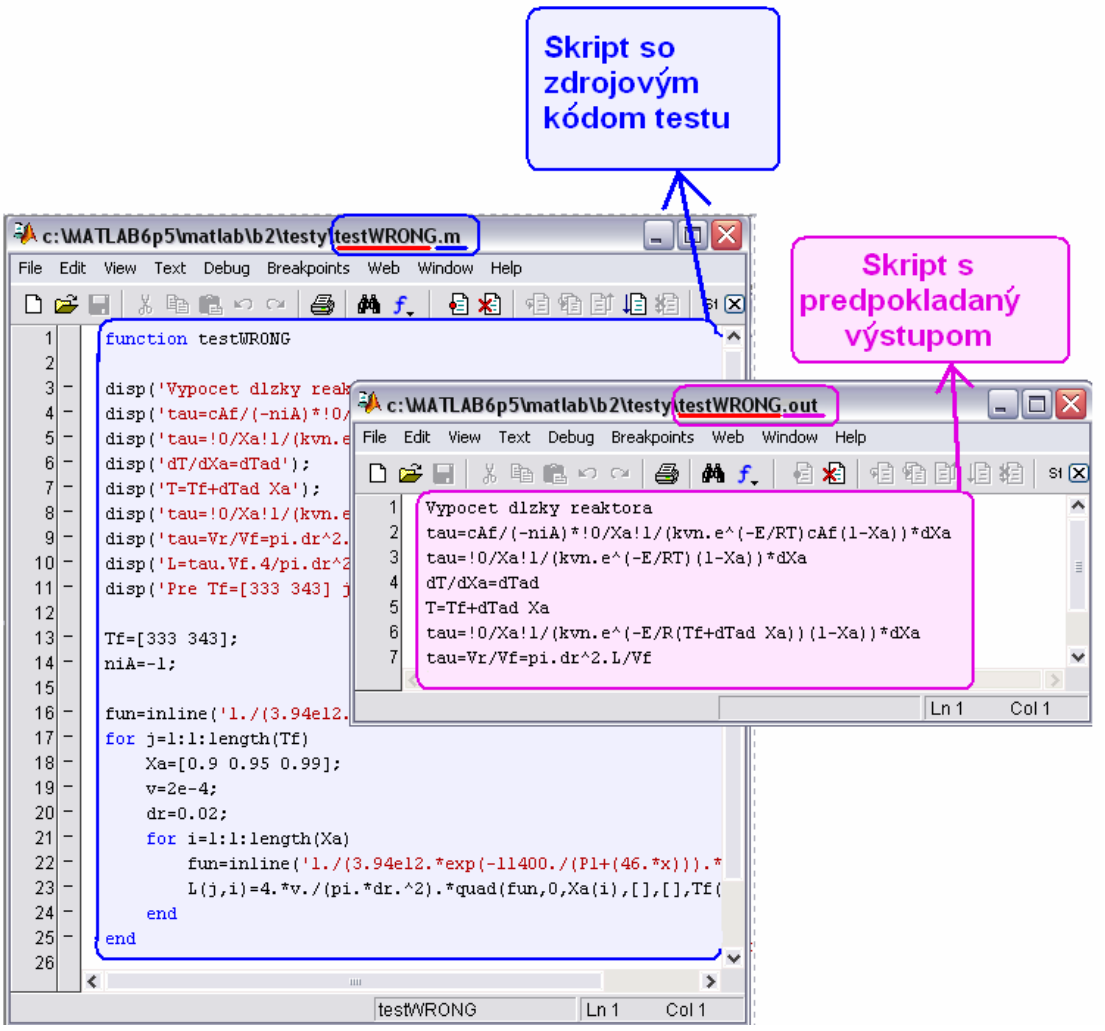
Testy kódov vytvárame na základe očakávaní, ktoré by mali tieto kódy spĺňať. Medzi očakávania zahŕňame aj ich predpokladaný výstup. Predpokladaný výstup testu ukladáme pod tým istým názvom ako daný test, ale pod inou príponou **.out.**

Predpokladaný výstup je jeden z dôležitých kritérií pri určovaní výsledného hodnotenia testu. Na základe predpokladaného výstupu sa určuje, či daný test je OK, keď skutočný výstup testu sa zhoduje s očakávaným výstupom alebo je WRONG, keď skutočný výstup testu sa nezhoduje s očakávaným výstupom. Nie každý test musí mať svoj predpokladaný výstup.

Rozpoznávame dva základe typy zápisu testov:

- **nazov_testu .out ,** ktorý obsahuje očakávaný výstup;
- **nazov_testu .m** , ktorý obsahuje vlastný kód testu.

Oba typy zápisu testu sú zobrazené na obr. 3.3. .



Obr. 3.4: Základne typy zápisu testov.

3.3 Konfiguračné položky

Pri testovaní sa snažíme, aby chybovo hodnotených testov bolo čo najmenej. Pri niektorých ale vieme predpokladať, že chybové hodnotenie nastane. V tom prípade je vhodné použiť konfiguračné položky, čím sa nám nenavŕši počet chybovo hodnotených testov. Konfiguračné položky je možné použiť aj v prípade, keď je test ešte len vo výstavbe a jeho prítomnosť pri testovaní je pre vývojára dôležitá.

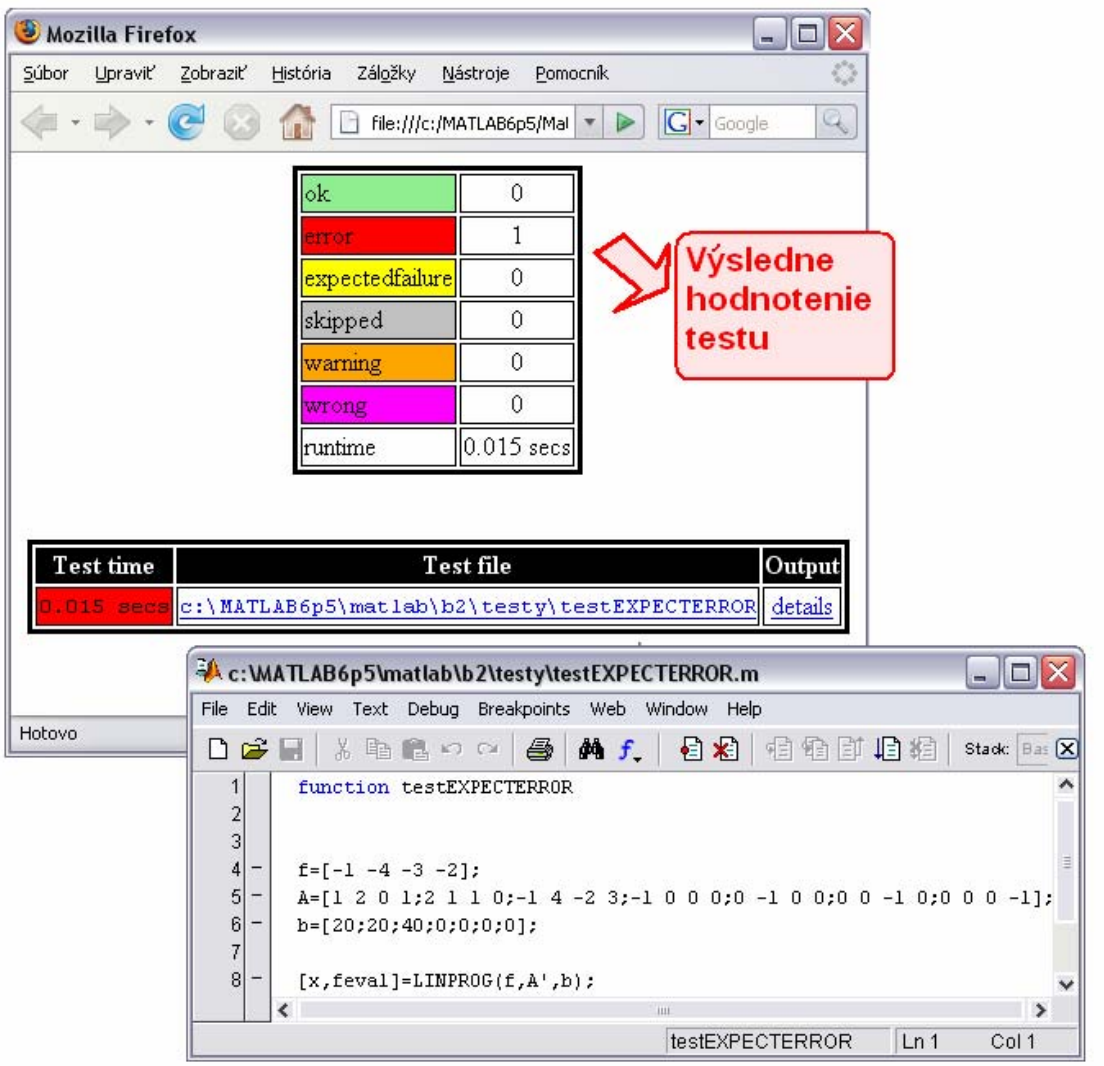
Nie vždy sa snažíme konfiguračnou položkou odstrániť chybovo hodnotený test. Taktiež možno ich zavedením do skriptu testu určiť metódu, pomocou ktorej sa bude uskutočňovať porovnávanie predpokladaného výstupu so skutočným (aktuálnym) výstupom.

Medzi prípustné konfiguračné položky patria:

- EXPECTERROR (predpokladane chybový);
- SKIP (preskočiť);
- STRICT (striktné porovnávanie).

3.3.1 EXPECTERROR

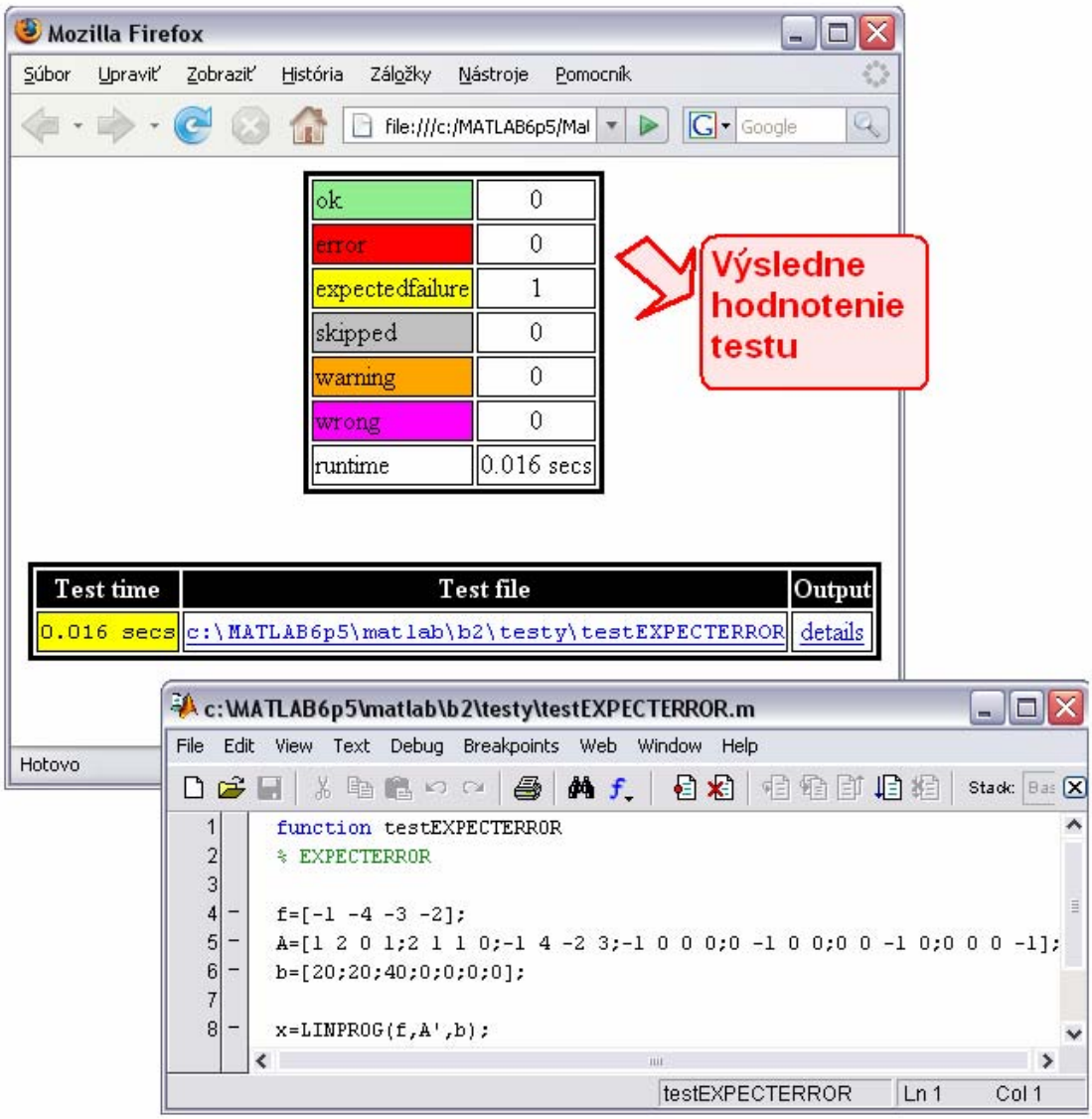
Ak po vytvorení testu vieme ešte pred spustením samotného testovania určiť, že test bude produkovať chybové hlásenie, a tým výsledné hodnotenie testu bude chybové, čo nechceme, je vhodné použiť konfiguračnú položku EXPECTERROR. Po zapísaní konfiguračnej položky EXPECTERROR do jeho skriptu a jeho prebehnutí, test nám vyprodukuje chybové hlásenie, ale celkové hodnotenie testu už nie je chybové. Tým sme uchránili testovanie pred ďalším chybovým hodnotením testu.



Obr. 3.5: Test s očakávanou chybou bez konfiguračnej položky a jeho výsledné hodnotenie.

V niektorých krajných prípadoch je naším cieľom, aby kód testu produkoval chybové hlásenie. Pretože test produkuje chybové hlásenie, po prebehnutí testovania sa nám test priráta medzi chybovo hodnotené testy, viď. obr. 3.5. . Tieto sú však pre nás neprípustné a nežiaduce.

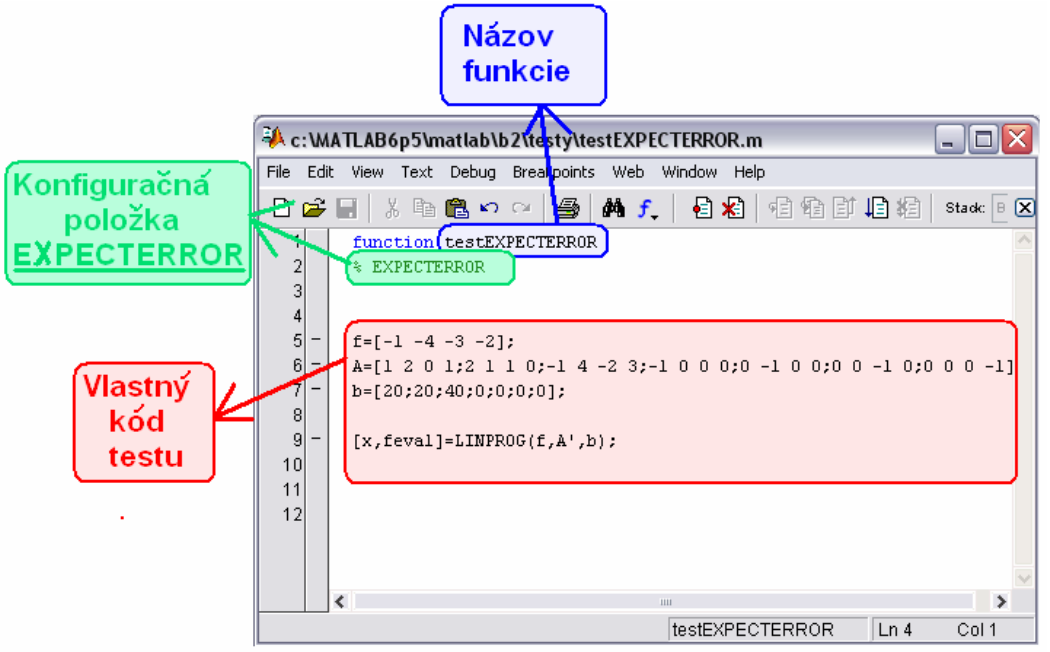
Na oddelenie testov s očakávanými chybovými hláseniami od testov, ktorých chybové hlásenia sú nechcené, používame konfiguračnú položku EXPECTERROR.



Obr. 3.6: Test s očakávanou chybou s konfiguračnou položkou a jeho výsledné hodnotenie.

Po zapísaní konfiguračnej položky do skriptu testu a jeho opätovnom spustení, test produkuje chybové hlásenie, len že nie je hodnotený ako chybový (error), ale ako predpokladane chybový (expected failure), viď obr. 3.6. .

Konfiguračnú položku EXPECTERROR zapisujeme do druhého riadku v komentári pod názov funkcie ako možno vidieť na obr. 3.7. .



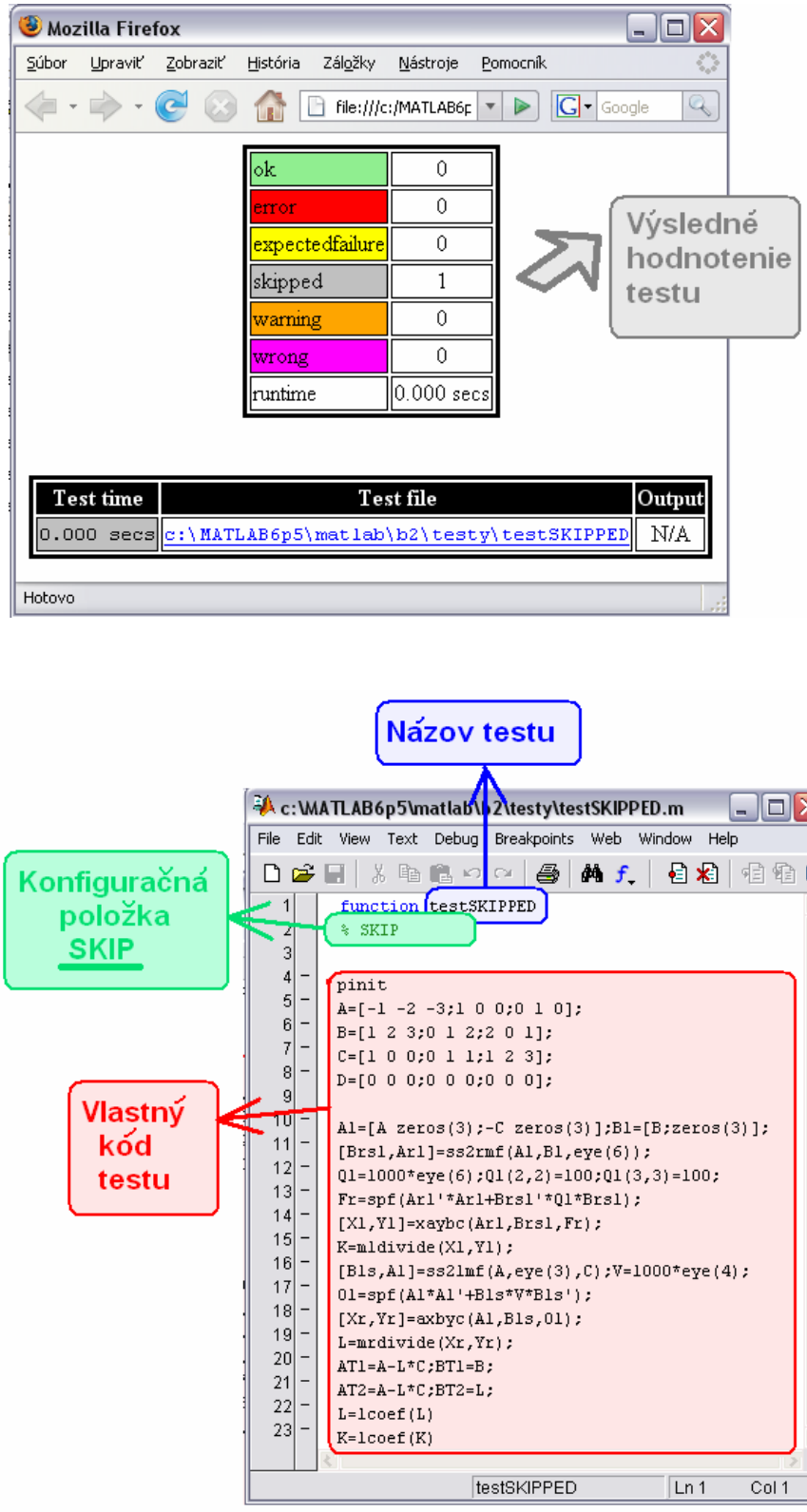
Obr. 3.7: Zápis testu s konfiguračnou položkou EXPECTERROR.

3.3.2 SKIP

Ak máme test ešte len vo výstavbe a jeho prítomnosť je pri testovaní potrebná, je vhodné použiť konfiguračnú položku SKIP.V tomto prípade sa nám nenavýši počet chybovo hodnotených testov.

Po spustení testu sa nám postupne načítavajú a vykonávajú jednotlivé riadky skriptu. Ak sa pri načítavaní testu načíta konfiguračná položka SKIP, test sa automaticky ukončí a nedochádza k načítavaniu vlastného kódu testu. Pri výslednom hodnotení test nie je považovaní za chybový.

Na obr. 3.8. je znázornený obsah testu s použitím konfiguračnej položky SKIP ako aj jeho výsledné hodnotenie.



Obr. 3.8: Zápis testu s konfiguračnou položkou SKIP a jeho výsledne hodnotenie.

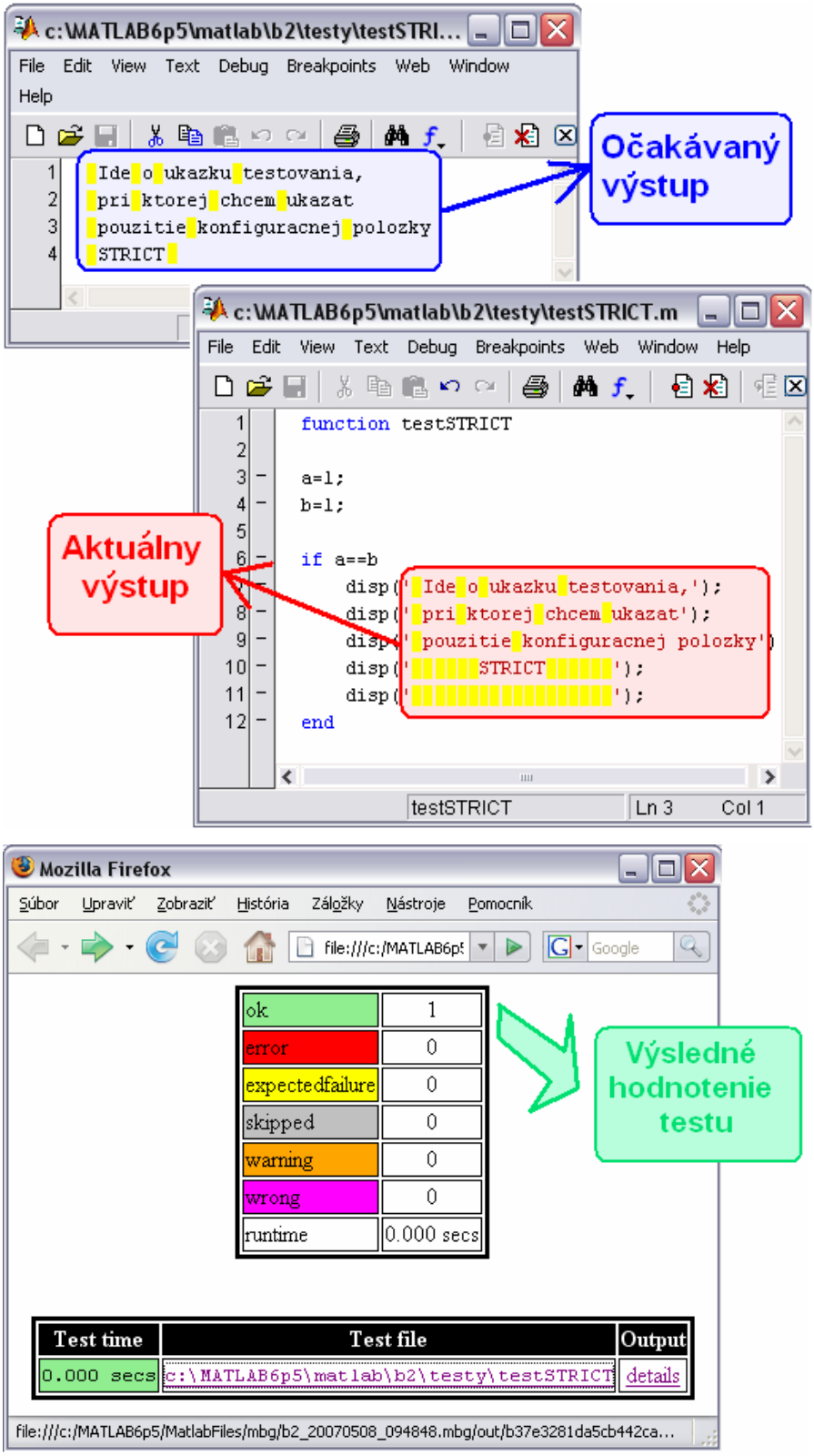
3.3.3 STRICT

Pri niektorých úlohách sa vyžaduje, aby náš výstup bol úplne totožný s predpokladaným výstupom. V tomto prípade sa vyžaduje, aby nielen samotný text, ale aj počet použitých medzier bol zhodný s predpokladaným výstupom. Pri klasickom porovnávaní sa nekladie dôraz na počet medzier, porovnáva sa iba obsah textu. Na uskutočnenie dôkladného porovnávania využívame konfiguračnú položku STRICT**.**

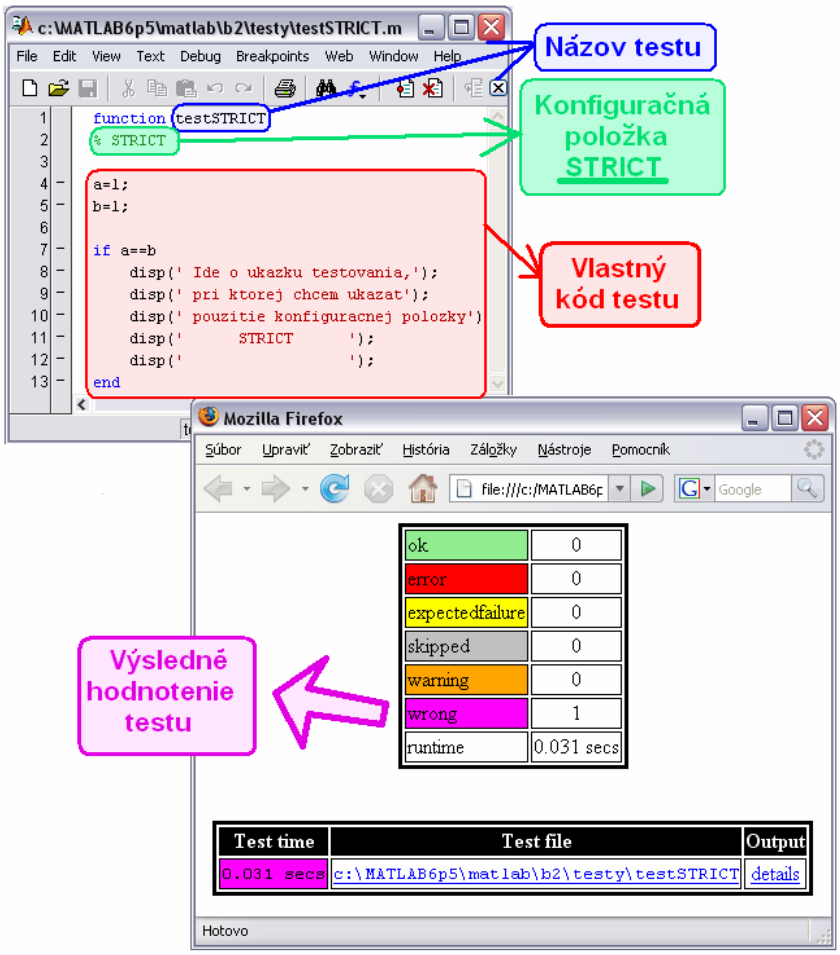
Pri použití konfiguračnej položky STRICT dochádza k striktnému porovnávaniu očakávaného a skutočného výstupu z testu. Pri porovnávaní sa kladie dôraz nie len na obsah textu, ale aj na počet medzier. Test, ktorý pri použití konfiguračnej položky STRICT produkuje rovnaký výstup ako je očakávaný, ale počet medzier je rôzny, sa považuje za WRONG.

Veľký dôraz treba dať v tomto špecifickom prípade aj na písanie testu očakávaného výstupu. Ak pri písaní očakávaného výstupu na koniec textu pripojíme prázdny riadok, tak tento prázdny riadok vyžadujeme aj od skutočného výstupu a naopak.

Ako príklad možno uviesť test s predpokladaným výstupom, ktorý je zobrazený na obr. 3.9.. Žltou farbou sú znázornené použité medzery v texte. Test produkuje 5 riadkov, v ktorých je zahrnutý jeden prázdny riadok. My ale očakávame len 4 riadky, pričom s menším počtom medzier ako je v skutočnom výstupe. Ak necháme prebehnúť testovanie s klasickým porovnávaním (bez použitia konfiguračnej položky STRICT), výsledné hodnotenie bude OK, aj keď nám test produkuje iný počet medzier ako očakávame. Porovnávame len znenie textu a to je totožné. Pri použitý konfiguračnej položky STRICT nám dochádza k striktnému porovnávaniu, čo má za následok, že test je hodnotený ako WRONG. Dochádza k odhaleniu rôzneho počtu medzier medzi jednotlivými výstupmi.



Obr. 3.9: Príklad testu s očakávaným výstupom a jeho výsledné hodnotenie.



Obr. 3.10: Zápis testu s konfiguračnou položkou STRICT a jeho výsledné hodnotenie.

3.4 Hodnotenie výsledkov testov

Ešte pred samotným spustením testovania je vhodné oboznámiť sa z možnými výsledkami testov. Hodnotenie výsledkov testov je v niektorých prípadoch pevne späté s vytváraním testov.

V testovacích frameworkoch rozpoznávame dve základné kategórie výsledkov testov:

- Error (neúspešný test)
- Ok (úspešný test)

Toto základné delene nie je objektívne. Neumožňuje určiť, či boli splnené všetky požiadavky kladené na kód, či sa počas testovania nezobrazili rôzne upozornenia, ako aj určiť, či chybové hlásenie bolo očakávané.

Už spomenuté základné kategórie sme preto rozšírili o ďalšie 4 kategórie, čo umožnilo dokonalejšie vyhodnotenie výsledkov testov. Testy tak rozdeľujeme do týchto šiestich kategórii:

- **ok**
- **error –** ak test produkuje chybové hlásenie;
- **expectedfailure –** ak sme použili konfiguračnú položku EXPECTERROR;
- **skipped –** ak sme použili konfiguračnú položku SKIP;
- **warning –** ak test produkuje varovania, napr. pri delení 0;
- **wrong –** ak predpokladaný výstup sa nerovná aktuálnemu výstupu.

3.4.1 OK

Výsledné hodnotenie testu je ok, ak sú splnené všetky požiadavky kladené na kód. Medzi tieto požiadavky patrí:

- vykonanie testu bez chybového (error) hlásenia;
- vykonanie testu bez upozornenia (warning);
- zhodnosť skutočného výstupu s očakávaným výstupom;
- pri striktnom porovnávaní zhodnosť počtu použitých medzier v skutočnom a očakávanom výstupe.

3.4.2 ERROR

Výsledné hodnotenie testu je error (chybovo hodnotené testy, neúspešné testy), ak počas testovania sa vyskytlo chybové hlásenie (error).

Jeho skrátený zápis je ERR.

3.4.3 EXPECTED FAILURE

V niektorých prípadoch je naším cieľom získať chybové hlásenie. V týchto prípadoch očakávame vznik chybového hlásenia. Na oddelenie testov s očakávaným chybovým hlásením od testov hodnotených error sme zaviedli novú kategóriu výsledkov testov nazvanú expected failure (predpokladane chybové).

Predpokladane chybové hodnotenie testu získame, ak pri písaní testu s očakávanou chybou uvedieme do skriptu konfiguračnú položku EXPECTERROR v komentári druhého riadku.

Jeho skrátený zápis je EXF.

3.4.4 SKIPPED

Pri testovaní je lepší škodlivý test než žiaden test, preto je dobré, aby sa testovania zúčastnili aj testy, ktoré sú nedokončené. Tieto testy nám zväčša produkujú chybové hlásenia. Pre testy, ktoré sú len vo výstavbe, sa preto zaviedla ďalšia kategória výsledkov testov nazvaná skipped (preskočené).

Výsledné hodnotenie skipped získame, ak pri písaní testu uvedieme do testu konfiguračnú položku SKIP v komentári druhého riadku. Počas testovania pri načítaní konfiguračnej položky SKIP, sa test automaticky ukončí. Jeho čas behu (runtime) je 0 sekúnd, a preto nám neovplyvňuje celkovú výkonnosť testovania.

Jeho skrátený zápis je SKP.

3.4.5 WARNING

Pri niektorých testoch sa nám zobrazuje varovné hlásenie. Napríklad hlásenie upozorňujúce delenie 0 „Warning: Divide by zero“. Pre tieto testy sme vytvorili samostatnú kategóriu výsledkov testov nazvanú warning. Tieto testy možno považovať za správne.

Jeho skrátený zápis je WNG.

3.4.6 WRONG

Počas testovania môže nastať situácia, pri ktorej test neprodukuje žiadne hlásenie, či chybové alebo varovné, ale aj tak test nemožno považovať za správny, pretože nevyprodukoval očakávaný výsledok. Tieto testy rátame medzi testy s výsledkom wrong (nesprávny).

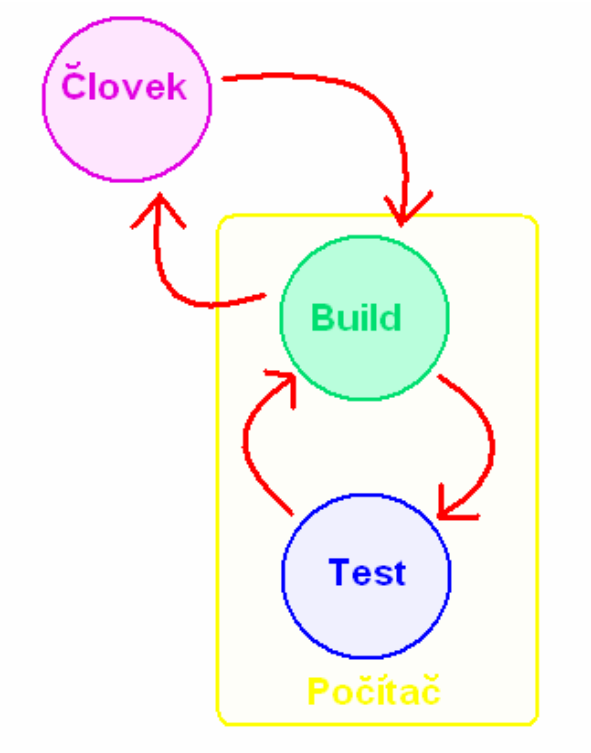
Testy s výsledkom wrong získame:

- Ak pri klasickom porovnávanísa nám nezhoduje obsah textu v predpokladanom (očakávanom) výstupe testu so skutočným výstupom.
- Ak pri striktnom porovnávaní sa nám nezhoduje obsah textu alebo počet použitých medzier v predpokladanom (očakávanom) výstupe testu so skutočným výstupom.

Jej skrátený zápis je WRO.

3.5 Testovanie

Pri písané kódu je potrebné najskôr napísať test, ten otestovať a následne napísať kód. Testovanie nám slúži hlavne na overenie funkčnosti a výkonnosti testovaného kódu. Pri testovaní netestujeme jednotlivé testy samostatne, ale testujeme presne lokalizovaný súbor testov, preto je vhodné ukladať testy do jednej veľkej skupiny, a tým urýchliť samotné testovanie.



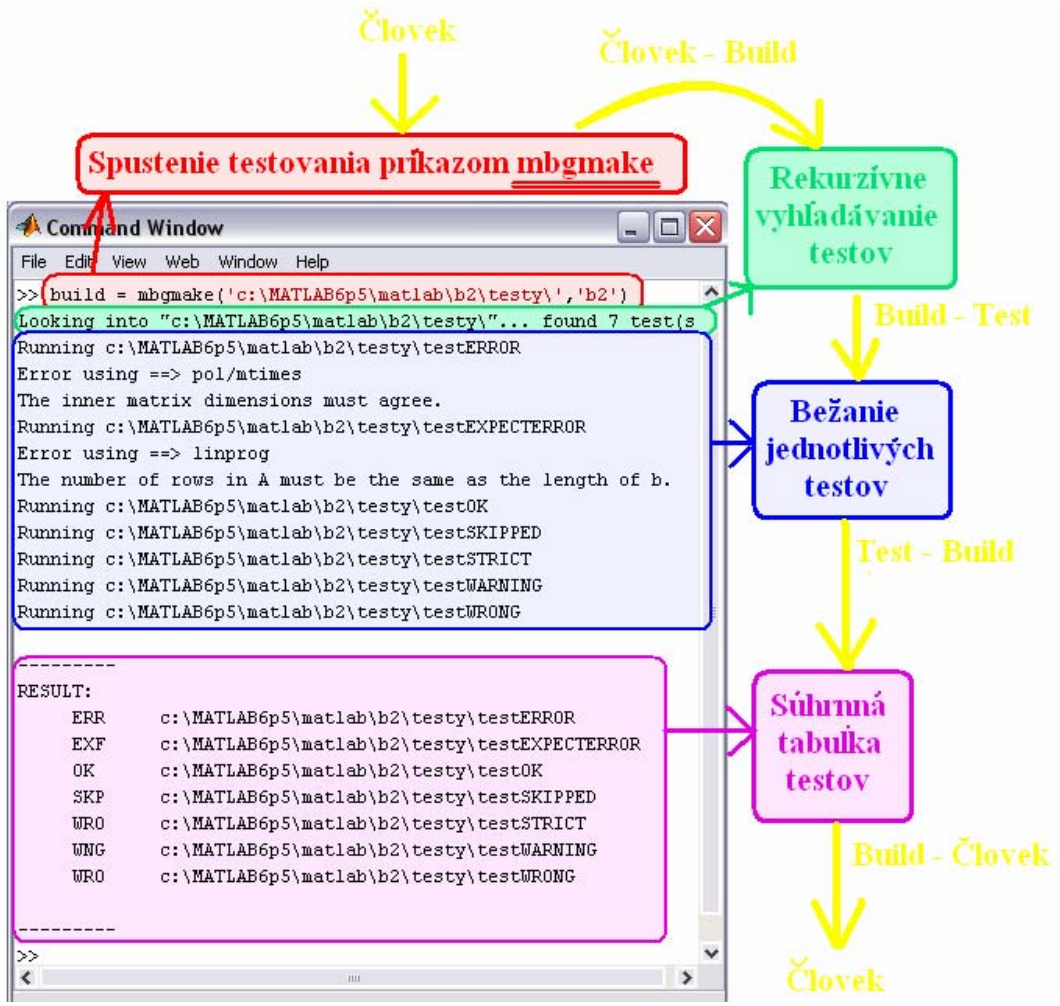
Obr.3.11: Tok informácií počas procesu testovania.

Počas testovania dochádza k intenzívnemu toku informácií medzi človekom (developerom), buildom a jednotlivými testami, viď. obr. 3.11. Test sprostredkúva výmenu informácii v rámci počítača a poskytuje základné informácie pre build, ktorý ich zhromažďuje. Build predstavuje prostriedok na obojsmernú výmenu informácii medzi počítačom a človekom, zhromažďuje jednotlivé výsledky testov, sumarizuje ich do súhrnnej tabuľky, ktorú poskytuje človeku. Umožňuje nám vidieť

správanie sa testov ako aj ich celový čas behu (runtime). Pod buildom taktiež možno rozumieť súbor testov, ktoré chceme spustiť, daných človekom.

Pri vytváraní programu sme vytvorili skupiny testov, ktorú sme uložili do spoločného adresára. Napríklad „*c:\MATLAB6p5\matlab\b2\testy*“. Samotné testovanie sme spustili zadaním príkazu *mbgmake* v príkazovom okne v znení:

>> build = mbgmake('c:\MATLAB6p5\matlab\b2\testy\','b2')

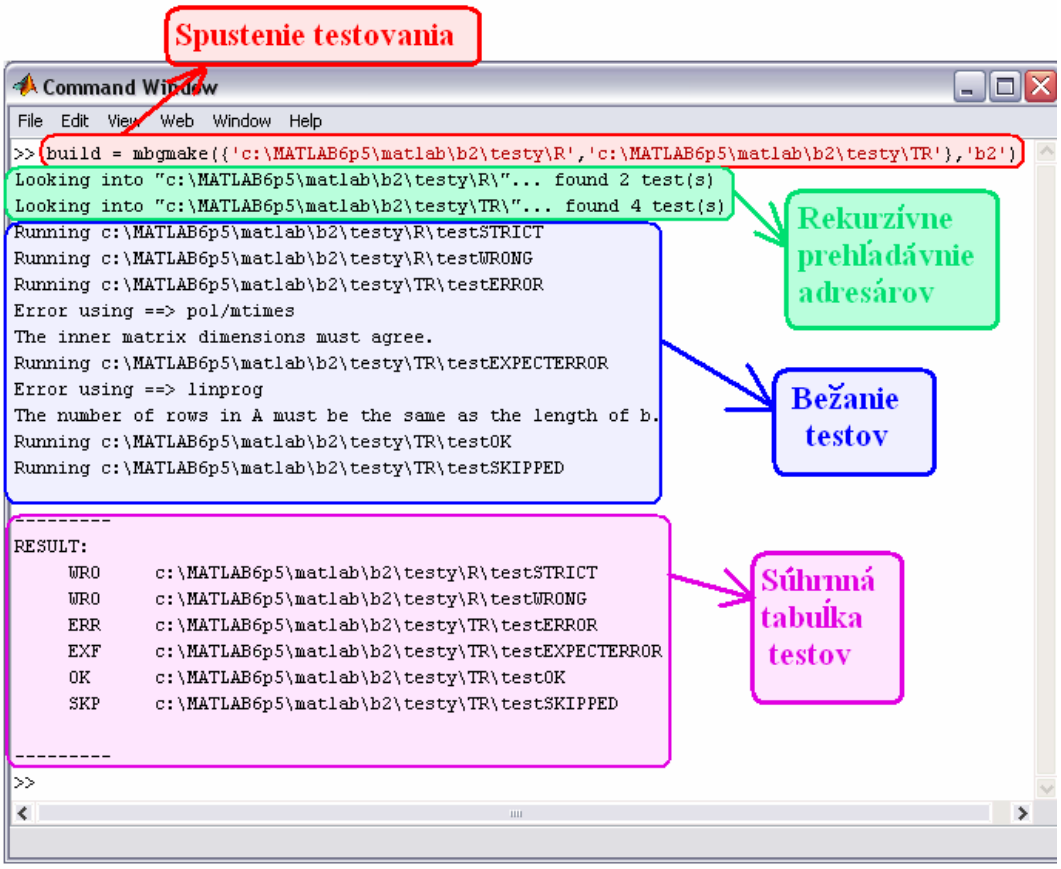


Obr. 3.12: Deje prebiehajúce počas testovania súboru testov.

Ako možno vidieť na obr. 3.12. po spustení testovania najskôr nastalo rekurzívne prehľadávanie testov v adresári a jeho podadresárov, a až následne došlo k spusteniu vyhľadaných testov. Po prebehnutí všetkých testov sme získali súhrnnú tabuľku všetkých testov. Žltou farbou je znázornený tok informácií počas testovania medzi človekom, buildom a testom.

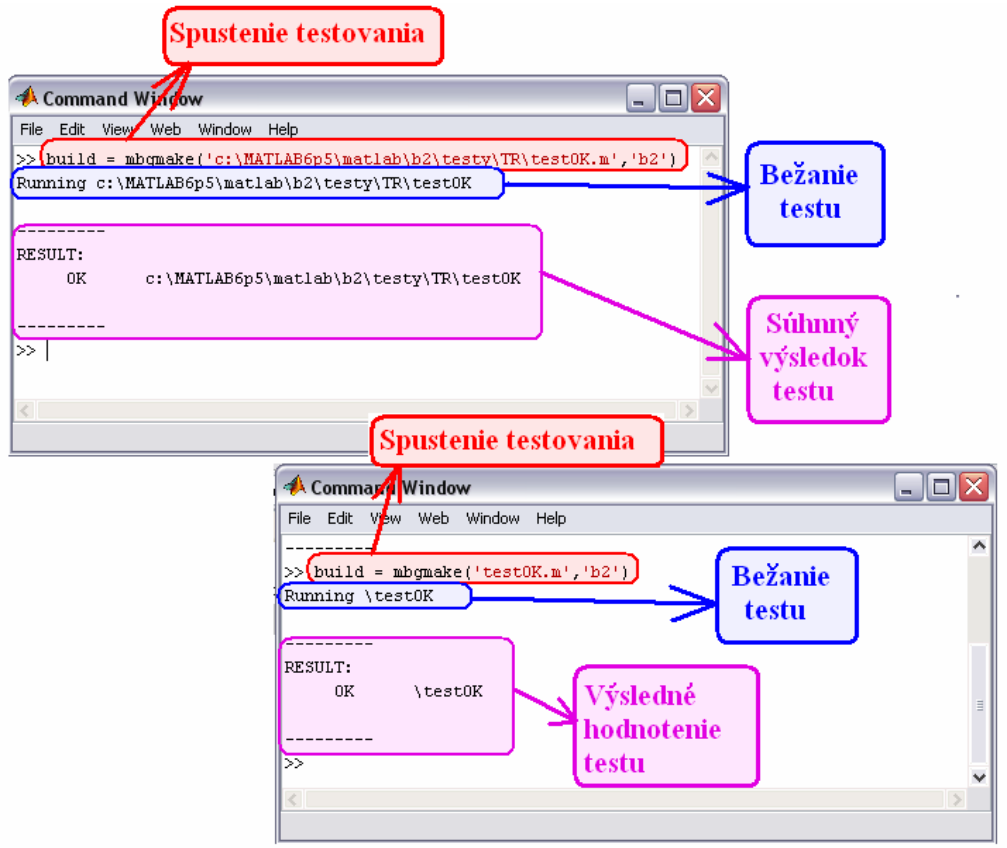
Príkazom *mbgmake* možno v prípade potreby spustiť, súčasne aj viac adresárov s testami, ktoré sú navzájom nezávisle. Napríklad „c:\MATLAB6p5\matlab\b2\testy\R“ , “c:\MATLAB6p5\matlab\b2\testy\TR “ . Znenie príkazu:

>> build = mbgmake ({ 'c:\MATLAB6p5\matlab\b2\testy\R' , 'c:\MATLAB6p5\matlab\b2\testy\TR '},'b2')



Obr. 3.13: Deje prebiehajúce počas testovania viacerých adresárov s testami.

Na obr. 3.13. je znázornení priebeh súčasného testovania dvoch adresárov Po rekurzívnom prehľadaní prvého adresára dochádza k rekurzívnemu prehľadaniu druhého adresára. Ak by sme mali zadaných viac adresárov, prehľadávanie nastáva v poradí, v akom boli zadané v príkaze *mbgmake*.



Obr. 3.14: Dlhý a skrátený zápis spustenia jedného testu.

Po prípadných opravách kódu v teste nemusíme spúšťať celú sadu testov, čo by bolo časovo náročné, ale môžeme spustiť testovanie konkrétneho testu. Napríklad po úpravách chceme opätovne spustiť testovanie testu „testOK.m“ s cestou „c:\MATLAB6p5\matlab\b2\testy\TR\testOK.m“. Syntax príkazu je nasledujúca:

>> build = mbgmake('c:\MATLAB6p5\matlab\b2\testy\TR\testOK.m','b2') ,

pre dlhý zápis spustenia testu.

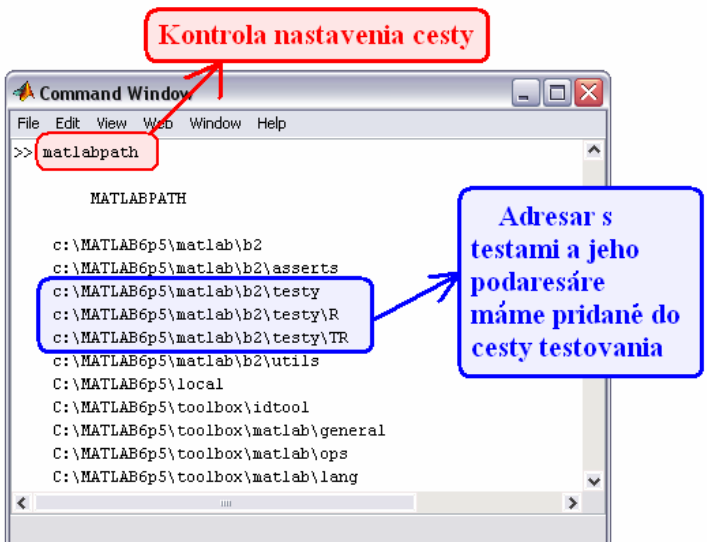
alebo

>> build = mbgmake('testOK.m','b2') , pre skrátený zápis spustenia testu.

Aj keď sme použili dva rôzne zápisy spustenia toho istého testu, získali sme úplne totožné výsledky, viď. obr. 3.14. Skrátený zápis môžeme použiť, ak adresáre s testami máme pridané do cesty testovania.

Na overenie, či testy máme v ceste testovania slúži príkaz *matlabpath.*

>> matlabpath



Obr. 3.15: Kontrola nastavenia cesty.

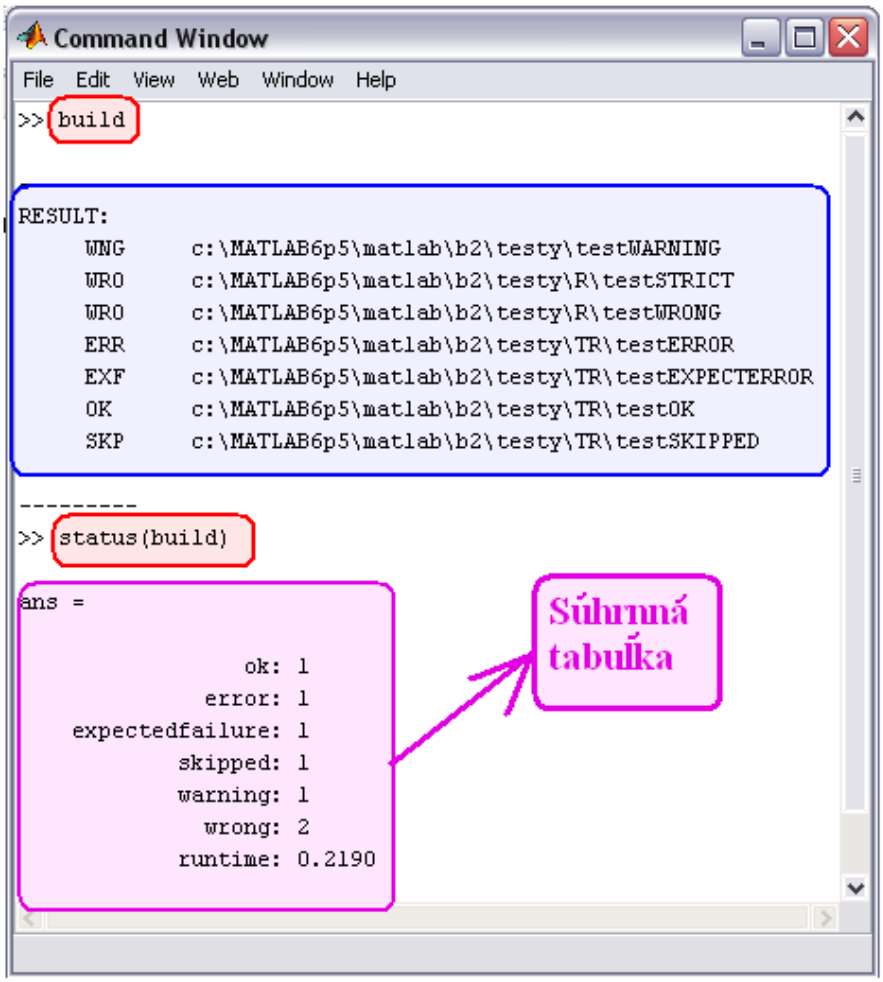
Rovnakými spôsobmi možno spustiť aj testovanie viacerých testov. Syntax príkazu je takmer totožná:

>> build = mbgmake({ 'c:\MATLAB6p5\matlab\b2\testy\TR\testOK.m' , 'c:\MATLAB6p5\matlab\b2\testy\TR\testERROR.m' , 'c:\MATLAB6p5\matlab\b2\testy\testWARNING.m' , 'c:\MATLAB6p5\matlab\b2\testy\R\testWRONG.m' }, 'b2') , pre dlhý zápis.

>> build = mbgmake({ 'testOK.m' , 'testERROR.m' , 'testWARNING.m' , 'testWRONG.m' } , 'b2') , skrátený zápis.

Oba zápisy nám vyprodukujú rovnaké výsledné hodnotenie testov, taktiež skrátený zápis možno použiť len ak ich cesty sú pridané do cesty testovania.

Na rýchle zistenie celkových výsledkov testov je príkaz *status(build),* ktorý nám umožňuje rýchly prehľad výsledkov ako aj zistenie výkonnosti testovania. Umožňuje nám určiť konkrétny počet testov, ktoré skončili daným hodnotením.



Obr. 3.16: Build a jeho status.

3.6 Hľadanie testov

Po prebehnutí testovania sa snažíme čo najrýchlejšie spracovať získané informácie, aby sme mohli izolovať, upraviť a opakovane otestovať testy, ktoré boli hodnotené ako error alebo wrong. Na základe súhrnnej tabuľky (statusu) vieme určiť iba koľko testov bolo hodnotených error alebo wrong, ale nie ktoré konkrétne sú to.

Pri malom počte testov je možné vyčítať chybové testy priamo z buildu, viď. obr. 3.16. Takéto ručné vyhľadávanie sa stáva pracné a hlavne časovo náročné pri väčších počtoch testov.

Omnoho elegantnejšie a hlavne časovo takmer nenáročné je vyhľadávanie s použitím príkazu *find*. Vyhľadávať môžeme na základe rôznych kritérií. Použitie príkazu *find* si môžeme uviesť na niekoľkých príkladoch vychádzajúcich s buildu znázorneného na obr. 3.16. Z neho chceme:

1. vypísať všetky testy, ktoré skončili hodnotením wrong a error. Syntax príkazu je nasledujúca:

>> build_n = find(build , 'status' , 'wrong' , 'status' , 'error')

resp.

>> build_n = find(build , 'status' , 'WRO' , 'status' , 'ERR')

Výsledok hľadania je zobrazený na obr. 3.17.

2. vypísať všetky testy, ktoré neskončili hodnotením ok a warning. Syntax príkazu je nasledujúca:

>> build_n = find(build , 'status' , '~ok' , 'status' , '~warning')

resp.

>> build_n = find(build, 'status' , '~OK' ,'status' , '~WNG')

Výsledok hľadania je zobrazený na obr. 3.18.

3. vypísať všetky testy, ktoré sa nachádzajú v adresári „*c:\MATLAB6p5\matlab\b2\testy\R*“. Syntax príkazu je nasledujúca:

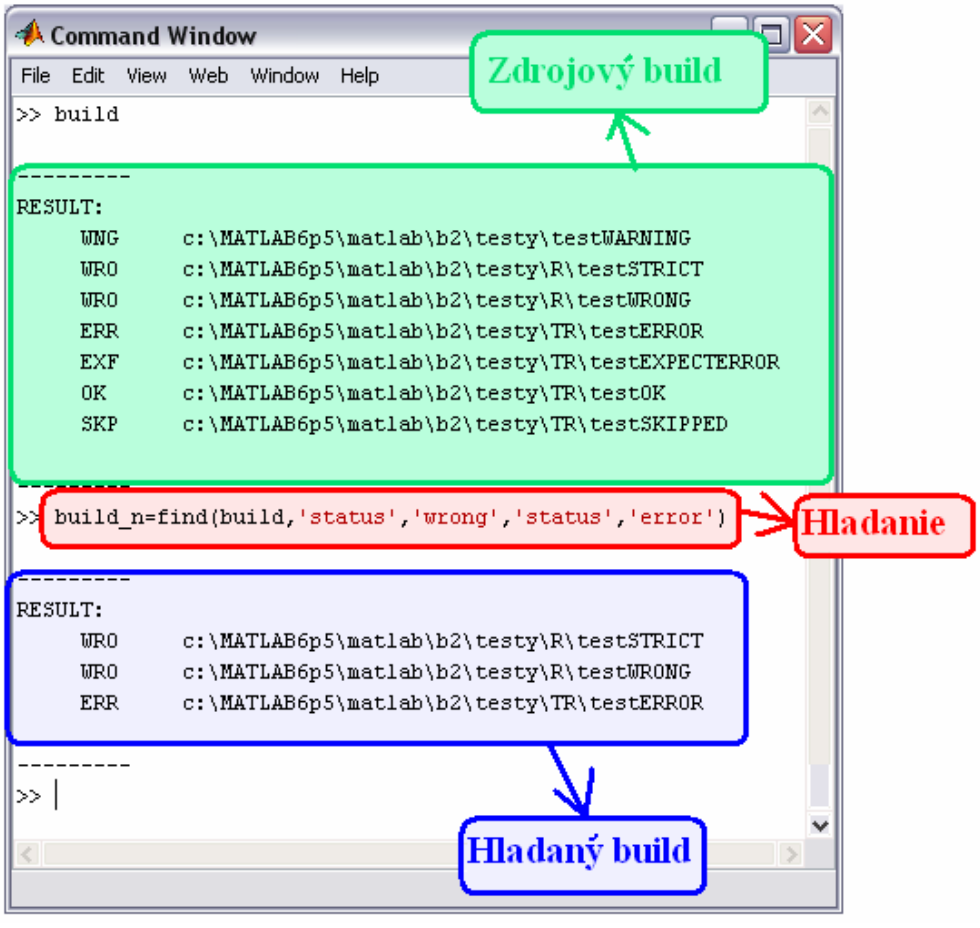
>> build_n = find(build , 'path' , 'c:\MATLAB6p5\matlab\b2\testy\R')

Výsledok hľadania je zobrazený na obr. 3.18.

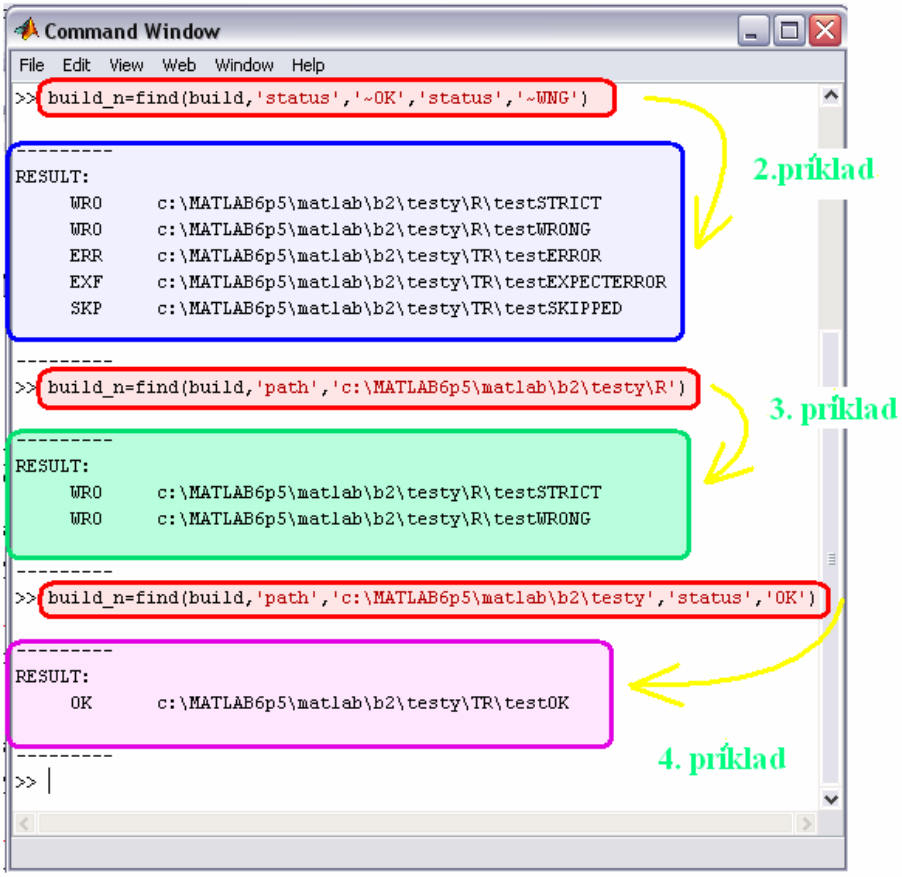
4. vypísať všetky testy, ktoré skončili OK v adresári „*c:\MATLAB6p5\matlab\b2\testy*“. Syntax príkazu je nasledujúca:

>> build_n = find(build , 'path' , 'c:\MATLAB6p5\matlab\b2\testy' , 'status' , 'OK')

Výsledok hľadania je zobrazený na obr. 3.18.



Obr. 3.17: Výsledok hľadania príkladu 1.



Obr. 3.18: Výsledok hľadania príkladu 2. , príkladu 3. a príkladu 4. .

3.7 Exportovanie výsledkov testovania

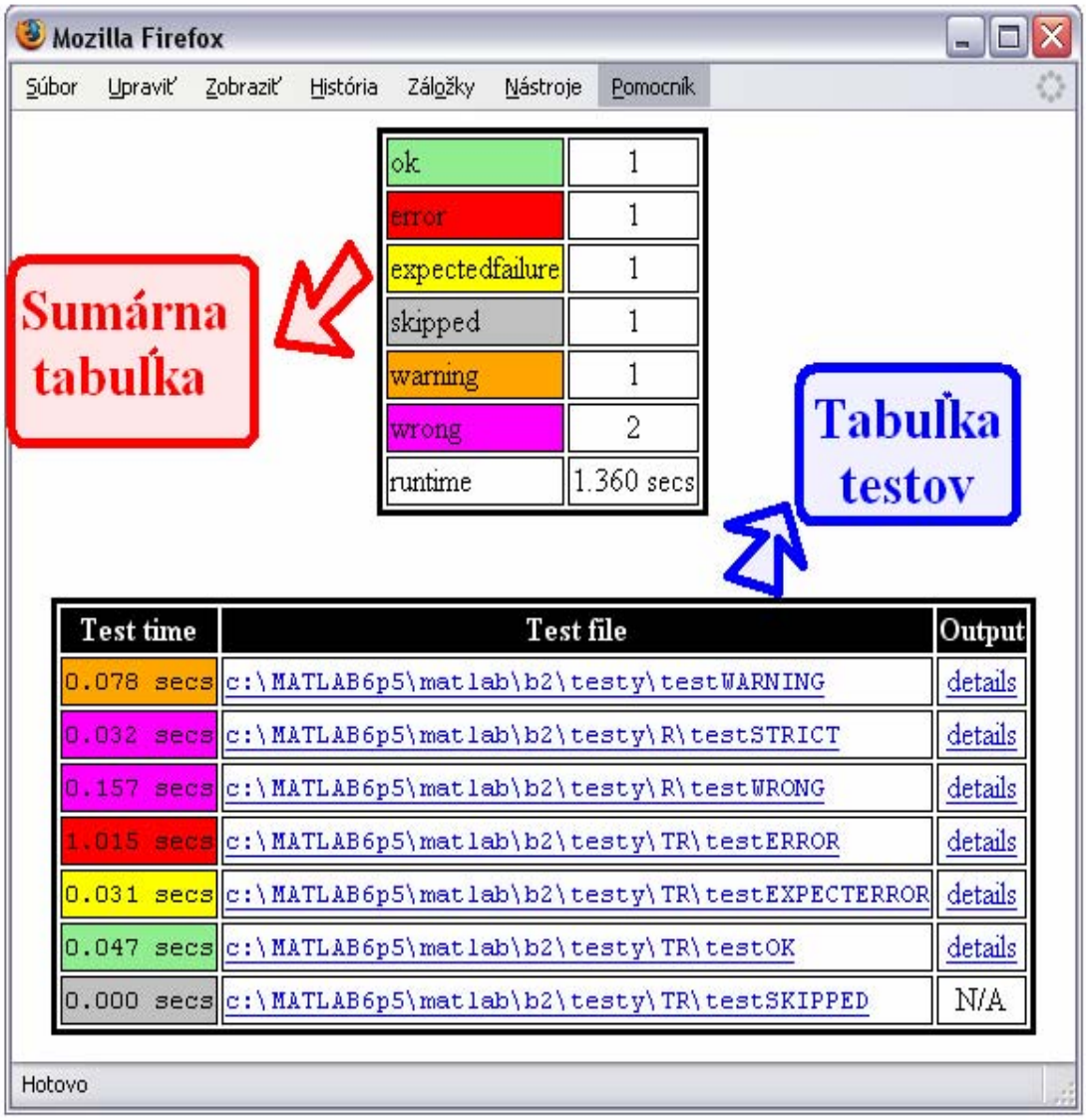
Exportovaním výsledkov testovania do HTML stránky dochádza k ich archivovaniu. Exportovanie nám umožňuje:

- neskoršie prezerania výsledkov testov;
- prezeranie obsahu jednotlivých testov s možnosťou ich stiahnutia a opätovného spustenia;
- grafické porovnávanie skutočného a očakávaného výstupu.

HTML stránka s výsledkami testov obsahuje dve hlavné časti:

- sumárnu tabuľku, nachádzajúcu sa v hornej časti stránky;
- tabuľku testov.

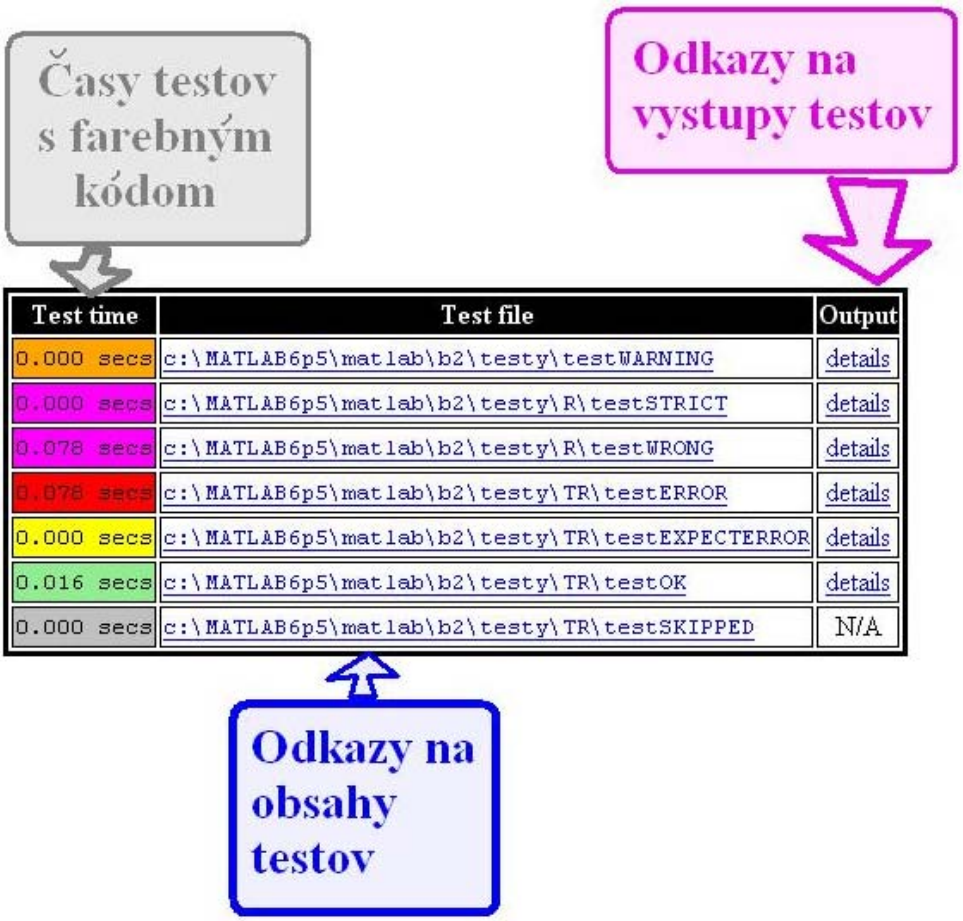
Sumárna tabuľka nás informuje o počte testov, ktoré skončili daným hodnotením a o celkovej výkonnosti daného testovania (runtime).



Obr. 3.19: HTML stránka s výsledkami testov.

Sumárna tabuľka a tabuľka testov využíva na rýchle rozlíšenie jednotlivých výsledkov testov farebný kód. K jednotlivým hodnoteniam (stavom) testov sú priradené charakteristické farby:

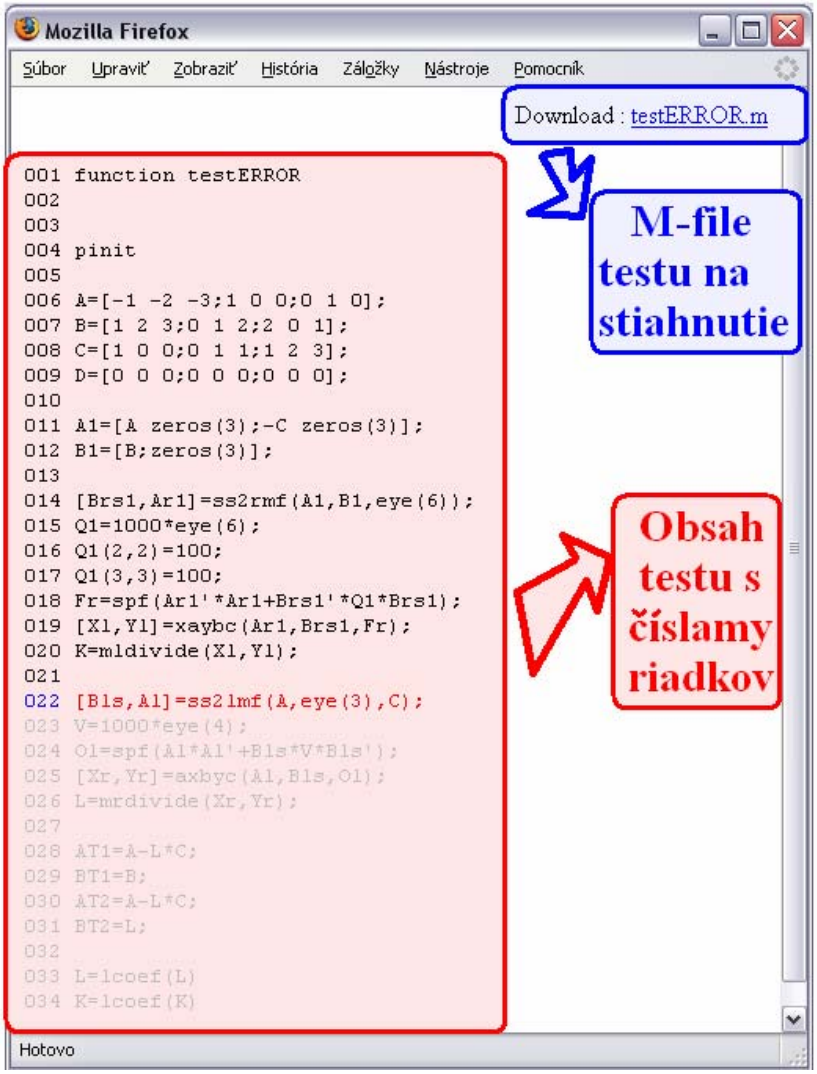
- zelená farba charakterizuje test hodnotený ako OK;
- červená farba charakterizuje test hodnotený ako ERROR;
- žltá farba charakterizuje test, v ktorom očakávame chybu (EXPECTED FAILURE);
- sivá farba charakterizuje test hodnotený ako SKIP;
- oranžová farba charakterizuje test s varovným hlásením (WARNING);
- fialová farba charakterizuje test hodnotení ako WRONG.



Obr. 3.20: Zloženie tabuľky testov.

Pod sumárnou tabuľkou sa nachádza tabuľka testov pozostávajúca s troch stĺpcov, pričom každý z nich má svoju osobitú funkciu, viď obr. 3.20. Každý riadok v tabuľke testov poskytuje informácie o konkrétnom teste.

Prvý stĺpec v tabuľke testov obsahuje jednotlivé časy testov (test time) s využitím farebného kódu. Umožňuje nám to rýchle vyhľadávanie testov v tabuľke na základe ich hodnotenia. Tento spôsob zobrazenia hodnotenia testov je oveľa efektívnejší než keby boli hodnotenia uvedené slovne vedľa názvu testu.



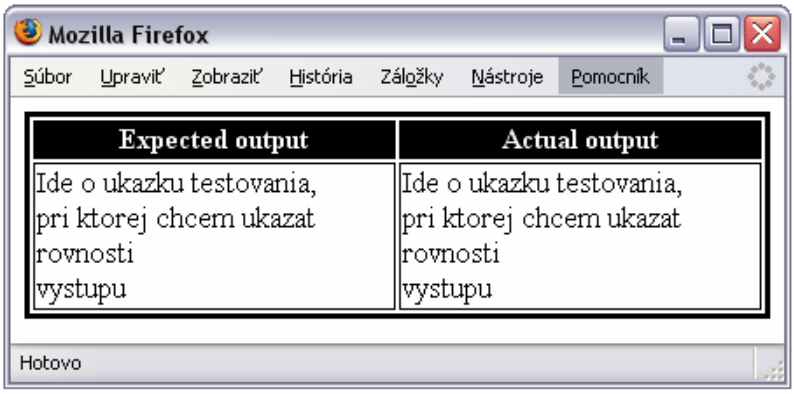
Obr. 3.21: Popis HTML stránky s obsahom chybového testu.

V druhom stĺpci sú odkazy na HTML stránky s obsahmi jednotlivých testov. Obsah zdrojového kódu testu je zobrazený s číslovaním riadkov. Má to veľký význam v prípade chybových testov, kedy sa nám riadok, na ktorom nastalo chybové hlásenie, spolu s jeho číslom farebne odlíšia. Ako možno vidieť na obr. 3.20 obsah riadku 22, na ktorom nastalo chybové hlásenie, sa zobrazilo červenou farbou a samotné číslo modrou farbou. Zvyšný kód, ktorý nám neprebehol sa zobrazil šedou farbou. Umožňuje nám to rýchlu lokalizáciu chybového kódu.

Táto HTML stránka nám umožňuje aj po určitej dobe v prípade potreby stiahnutie zdrojového m-file testu do nášho počítača. Takto možno rýchlo získať test na uskutočnenie opätovného testovania.

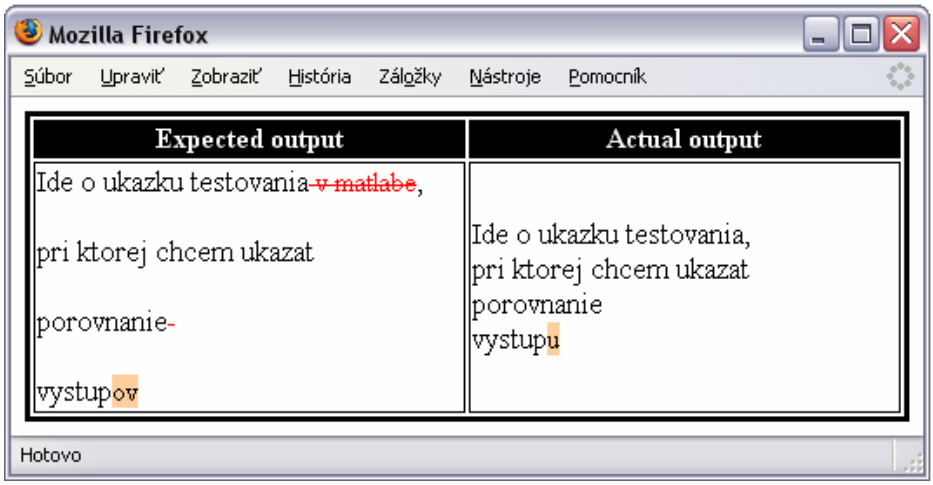
Tretí stĺpec v tabuľke testov je stĺpec výstupov. Ak test neprodukoval žiaden výstup, v stĺpci pri danom teste je napísané N/A, v opačnom prípade obsahuje odkaz na HTML stánku, v ktorej je zobrazené grafické porovnávanie očakávaného (Expected output) a skutočného výstupu (Actual output). Počas porovnávania môžu nastať niekoľko situácií.

Prvý prípad nastane počas zhodnosti očakávaného výstup so skutočným výstupom. V tomto prípade text je napísaný bez akého koľvek vyfarbenia, viď obr. 3.22.



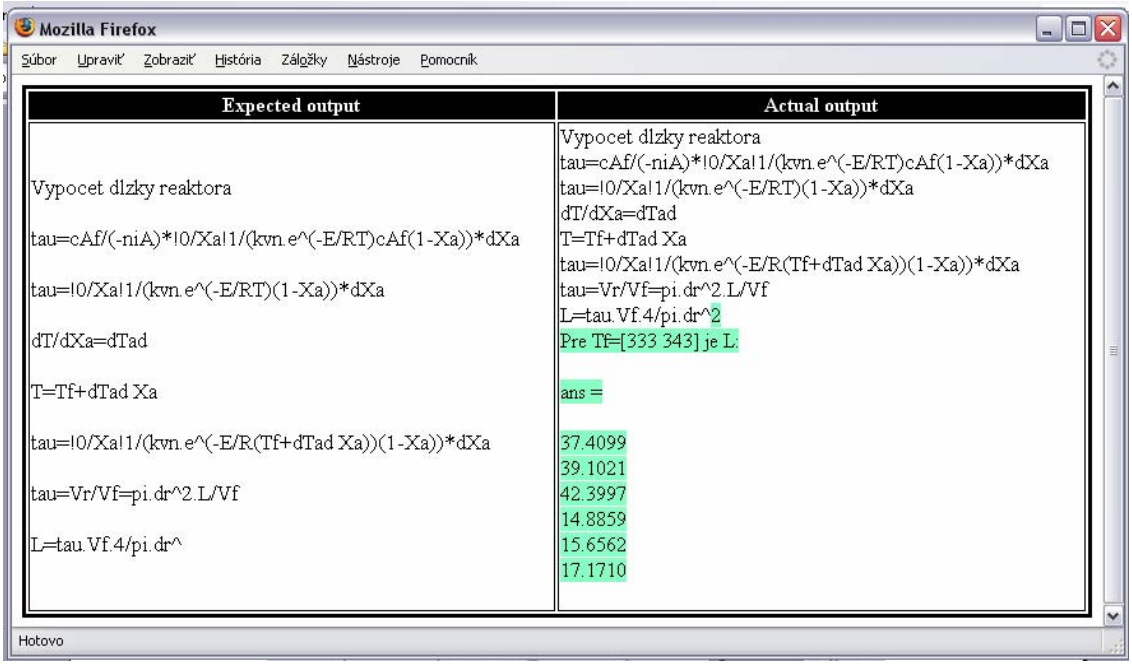
Obr. 3.22: Znázornenie zhodnosti výstupov.

Ak očakávaný výstup má viac slov než skutočný výstup, v tomto prípade slová, ktoré nie sú v skutočnom výstupe, sú prečiarknuté červenou farbou. Prípade odlišnosti časti slova medzi skutočným a očakávaným výstupom, táto časť je zvýraznená oranžovou farbou (obr. 3.23).



Obr. 3.23: Zobrazenia vymazávaných slov a zmenených častí slov.

Ak skutočnému výstupu boli pridané nové slová, tieto sú zvýraznené zelenou farbou (obr. 3.24).



Obr. 3.24: Zobrazenie pridaných slov v očakávanom výstupe.

Záver

Cieľom tejto diplomovej práce bolo navrhnúť a implementovať framework na testovanie programov v prostredí MATLAB. Testovací framework bol navrhnutí na základe metódy Test – Driven Development (TDD), ktorá spočívala v tom, že samotné napísanie kódu programu predchádzalo vytvorenie jeho testu a samotné testovanie. Testovací framework je šírení na základe GNU licencie.

Použitie navrhnutého testovacieho frameworku spočíva vo vytváraní, spravovaní a spúšťaní balíku testov, ktorý bol navrhnutý na overovanie funkčnosti, správnej činnosti a výkonnosti testovaného softvéru.

Testovací framework nám umožňuje archivovať získané výsledky testov na základe ich exportovania do HTML stránky. Na základe exportovanie možno opätovne prezerať získané výsledky testov ako aj prezerať obsahy jednotlivých testov a v prípade záujmu ich možno stiahnuť a uskutočniť ich opätovné prebehnutie. Počas prezerania obsahu chybových testov možno na základe farebného zvýraznenia rýchlo učiť číslo riadku, na ktorom vzniklo chybové hlásenie. Ďalšou výhodou je grafické porovnávanie skutočného a očakávaného výstupu.

Zavedenie rozšírenej kategorizácie výsledkov testov nám umožnilo presne určiť ich výsledné hodnotenie a ochrániť testovanie od zbytočných chybovo hodnotených testov. Ide o prípad nedokončených testov ako aj testov s očakávanou chybou.

Vylepšenie testovacieho frameworku spočíva v jeho napojení na databázu a následnom vytvorení webového systému jazyku PHP, ktorý bude umožňovať pohodlné prehliadanie a porovnávanie záznamov vyhotovených v rôznych časových okamihoch.

Literatúra

[1] URL: http://mlunit.dohmke.de/Unit_Testing_With_MATLAB (online, citované dňa 14.5.2007).

[2] URL: http://en.wikipedia.org/wiki/Software_testing (online, citované dňa 14.5.2007).

[3] URO: http://en.wikipedia.org/wiki/Test-driven_development (online, citované dňa 14.5.2007).

[4] URO: http://vivek.sanchivi.com/technology/se/ar01s02.html (online, citované dňa 14.5.2007).

[5] URO: http://c2.com/cgi/wiki?WhatIsRefactoring (online, citované dňa 14.5.2007).

[6] URO: http://en.wikipedia.org/wiki/Test-Driven_Development_by_Example (online, citované dňa 14.5.2007).

[7] URO: http://www.agiledata.org/essays/tdd.html (online, citované dňa 14.5.2007).

[8] URO: http://cattleshow.net/tom/blog/archives/23 (online, citované dňa 14.5.2007).

[9] IEEE Standard for Software Unit Testing: An American National Standard, ANSI/IEEE Std 1008-1987 (online, citované dňa 14.5.2007).

URL: http://iteso.mx/%7Epgutierrez/calidad/Estandares/IEEE%201008.pdf