Česká zemědělská univerzita v Praze
Technická fakulta

Metody testování software aplikací s vestavěnými mikrokontroléry

Bakalářská práce

Autor: Adam Franc
Vedoucí práce: prof. Ing. Zdeněk Bohuslávek, CSc.

Praha 2014

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra elektrotechniky a automatizace

Technická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Franc Adam

Informační a řídící technika v agropotravinářském komplexu

Název práce

Metody testování software aplikací s vestavěnými mikrokontrolery

Anglický název

Methods of testing software of applications with embedded microcontrollers

---

Cíle práce

Zpracování přehledu a popisu metod testování software mikropočítačových aplikací se specifikací jejich použití. Návrh a provedení testování firmware mikropočítačové váhy s přenosem dat do PC.

Metodika

1. Popis a hodnocení metod testování software mikropočítačových aplikací
2. Výběr vhodných testovacích metod a příklady testování aplikací se zabudovanými (embedded) mikropočítači.
3. Návrh a provedení testů software konkrétního zařízení se zabudovaným mikropočítačem komunikujícího s PC.
4. Zhodnocení a závěr.

Osnova práce

1. Studium literárních pramenů klasických i internetu, vyhodnocení.
2. Výběr podle analýzy efektivnosti testů, ,stručný popis příkladů testů na přístrojích a mobilních zařízeních.
3. Návrh testů a jejich realizace.
4. Hodnocení navrženého testu z hlediska efektivnosti a časové náročnosti..

Rozsah textové části

35 stran včetně obrázků, grafů a tabulek

Klíčová slova

testování software, stress testy, zátěžové testy, mikropočítačová váha

Doporučené zdroje informací

1. HLAVATÝ, T. – PŘEUČIL, L.: Formal Methods in Development and Testing of Safety-Critical Applications. In: Proceedings Process Control – ŘIP 2002. University of Pardubice, Pardubice, 2002, vol. 1, p. 104. ISBN 80-7194--452-1.
2. ŠTĚPÁN, P. – PŘEUČIL, L.: Methods for improving software quality. In: Proceedings of International Carpathian Control Conference. TU FEI Košice. Košice, 2000. Vol. 1, pp. 665–668.
3. Hlavatý, T.: Návrh a testování softwaru pro systémy s vysokými nároky na bezpečnost a spolehlivost., AUTOMA, 2003,č.2, on-line: http://www.odbornecasopisy.cz/index.php?id_document=28707
4. Metody testování - Testování a diagnostika SW, on-line: http://www.certicon.cz/content/metody-testovani

Vedoucí práce

Bohuslávek Zdeněk, prof. Ing., CSc.

Termín zadání

listopad 2012

Termín odevzdání

duben 2014

prof. Ing. Jaromír Volf, DrSc.
Vedoucí katedry

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE
-3-
TECHNICKÁ FAKULTA

prof. Ing. Vladimír Jurča, CSc.
Děkan fakulty

V Praze dne 18.3.2013

Prohlášení

Prohlašuji, že jsem bakalářskou práci na téma „Metody testování software aplikací s vestavěnými mikrokontroléry„ vypracoval samostatně s použitím odborné literatury a dalších informačních zdrojů. Zdroje, které jsou v práci použity a citovány, jsou uvedeny v seznamu použitých zdrojů na konci práce.

V Praze dne ..... Podpis .....

Poděkování

Tímto bych rád poděkoval vedoucímu práce prof. Ing. Zdeňku Bohuslávkovi, CSc. za odbornou pomoc a praktické rady při vypracování této bakalářské práce. Dále bych chtěl touto cestou poděkovat své rodině za podporu.

Název

Metody testování software aplikací s vestavěnými mikrokontroléry

Abstrakt a klíčová slova

Abstrakt: Tato bakalářská práce se zabývá problematikou testování embedded zařízení v praktickém využití. Cílem práce je analýza testovacích metod, výběr nejvhodnější z nich a aplikování na váhový počítač „Computer scale RM plus“ firmy AP-EL. Teoretická část popisuje embedded zařízení a jednotlivé metody testování. V praktické části jsou navržené testy použity na váhový počítač. V závěru práce jsou výsledky testů vyhodnoceny.

Klíčová slova: testování software, embedded systémy, váhový počítač

Title:

Methods of testing software of applications with embedded microcontrollers

Abstract

Summary: This bachelor thesis deals with the testing of embedded systems in practical use. The target of this thesis is analysis of test methods, choose the best of them and use it on computer scale „Computer scale RM plus“ by AP-EL. Theoretical part describes embedded devices and testing methods. In practical part are proposed test used on computer scale. In the end of thesis are test results evaluated.

Key words: software testing, embedded systems, computer scale

Obsah

1. Úvod

První embedded systémy se začínají vyvíjet pro první automatizované lety do kosmu, kde je od zařízení vyžadována vysoká spolehlivost a samostatnost. Od té doby uběhla dlouhá doba a vestavné systémy jsou dnes nedílnou součástí každodenního života lidí, bez kterých se dnes neobejdeme. Škála výrobků, ve kterých tyto systémy můžeme najít, je velice rozsáhlá, od běžných spotřebičů, jako jsou ledničky, pračky nebo zabezpečovací systém domu po složité průmyslové systémy jako CNC stroje nebo automatizované výrobní linky. Aby byla zajištěna vysoká spolehlivost zařízení, je nezbytné tyto systémy podrobit dostatkem testů a simulací, které odhalí nedostatky.

Tato práce se zabývá především testováním firmware váhového počítače „Computer scale RM plus“ firmy AP-EL, který je určen pro přesné navažování krmiv zvířatům v živočišné výrobě. V následujících kapitolách se tato práce zabývá vlastnostmi embedded systémů a nejpoužívanějšími metodami testovaní a jejich testy. V praktické části jsou pak návrhy vybraných testů a jejich aplikace na testované zařízení.

V závěru práce jsou pak shrnuty výsledky jednotlivých testů a vyhodnocení z hlediska efektivnosti a časové náročnosti.

2. Teoretická část

2.1. Co je Embedded systém

Embedded (vestavěný) systém je jednoúčelový systém, obsahující řídící počítač, který poskytuje zařízení potřebnou inteligenci. Dnes jsou embedded systémy nedílnou součástí většiny domácích zařízení (praček, mobilních telefonů, televizí, aut, atd.), průmyslových zařízení nebo kosmických stanic. Pro představu jak rychle narůstá počet těchto zařízení - do roku 2010 bylo podle odhadů na světě kolem 16 ti miliard programovatelných mikrosoučástek. Hlavním požadavkem těchto systémů je, aby byli autonomní a plnily své funkce bez zásahu člověka po poměrně dlouhou dobu. Tento požadavek s sebou zároveň nese další nároky na systém a to především nízkou spotřebu energie, spolehlivost a určitou robustnost. Autonomnost těchto systémů se využívá tam, kde reakce člověka jsou příliš pomalé, nebo nedostatečně předvídatelné. Výhodou je jejich práce v reálném čase, kde je kladen důraz na rychlost prováděných funkcí. Ve většině případů použití embedded systémů je jejich požadavkem neviditelnost vůči uživateli, který by neměl poznat, že jde o počítač. (Mrštíková, 2005), (Krumnikl, 2006)

Důležité vlastnosti embedded systémů

- Jsou konstruovány pro konkrétní činnosti
- Schopnost pracovat v reálném čase
- Nízké náklady na výrobu
- Funkčnost s omezenými prostředky
- Každé zařízení může mít jiné uživatelské rozhraní

2.1.1 Z historie

Prvním skutečným embedded systémem byl naváděcí počítač pro Apollo (Apollo Guidance Computer), vyvinutý v laboratoři MIT Charlesem Starkem Draperem. Dalším významným milníkem v historii embedded systémů byl první masově vyráběný počítač Autonetics D-17 pro nukleární balistickou střelu LGM-30 Minuteman vyráběný v roce 1961. O deset let později se začal vyrábět první

obchodně úspěšný 4-bitový mikroprocesor Intel 4004, jenž byl součástí kalkulačky Busicom 141-PF. (Krumnikl, 2006), (Mrštíková, 2005), (Javůrek, 2011)

2.1.2 Architektury procesorů

Na rozdíl od stolních PC, kde na trhu je jen pár velkých výrobců procesorů (AMD, Intel) s nejznámější platformou x86, existuje u embedded systémů daleko větší výběr. Díky chytrým mobilním telefonům je asi nejznámější architektura ARM (ve verzích 6,7,9E nebo Xscale), ale i v dalších zařízení lze nalézt procesory architektury MIPS, Coldfire/68k, AVR , PIC a mnoho jiných. V této škále si mohou výrobci jednoduše vybrat, zda pro svoje zařízení potřebují výkonný dražší procesor nebo jim pro jejich účely postačí levnější varianta. (Brtník, 2011), (Krumnikl, 2006)

2.1.3 Firmware

Software, který je uložen přímo v hardwarovém zařízení, například ve flash paměti se nazývá firrmware. Firmware je navrhován, tak aby běžel v reálném čase s omezenými zdroji a neustále. Systém musí být schopen se sám restartovat, pokud dojde k selhání nebo zamrznutí systému. Restartování systému a jeho hlídání má na starosti speciální periferie „watchdog timer“, která hlídá signál ze systému. Pokud signál nepřichází v pravidelných intervalech (řádově milisekund) watchdog restartuje systém. (Mrštíková, 2005)

Příkladem jednoduchého a běžně dostupného mikropočítače je populární vývojový systém Arduino, který umožňuje běžným uživatelům jednoduše sestavit a naprogramovat vlastní embedded systém. Na obrázku č. 1 je ukázka kódu, kdy se při startu systému nastaví elektronické součásti (void setup), a poté se spustí nekonečná smyčka (void loop). Smyčka se zpožděním neustále mění stav na digitálně vstupně - výstupním pinu (blikání led diody). Zdrojový kód ve vývojovém prostředí Arduina je psán v jazyce C nebo C++ a následně zkompilován pomocí avr-gcc. Tento kód je následně nahrán přes sériovou linku do mikroprocesoru. (Banzi, 2011)

```
//uložení čísla 13 do proměnné led
int led = 13;

// načtení nastavení při resetu:
void setup() {
  // inicializace digitálního pinu jako výstup
  pinMode(led, OUTPUT);
}

// začátek nekonečné smyčky:
void loop() {
  digitalWrite(led, HIGH);   // nastaví digitální pin číslo 13 na logickou 1
  delay(1000);               // čeká sekundu
  digitalWrite(led, LOW);    // nastaví digitální pin číslo 13 na logickou 0
  delay(1000);               // čeká sekundu
}
```

Obrázek 1: Ukázka zdrojového kódu Arduina

V souvislosti se spolehlivostí systému je důležité dbát na kvalitu firmware, který se do zařízení nahraje. Proto je nedílnou součástí při psaní programu neustálé testování a ladění kódu. Z historie je známo několik případů, kdy nepatrná chyba ve firmware vedla ke ztrátám na životech a materiálním škodám. Příkladem takovéto chyby bylo špatné časování systémových hodin na americkém obranném raketovém systému Patriot, kde se chyba neustále akumulovala a při útoku nebyl systém schopen raketu správně zaměřit. (Patton, 2002)

2.1.4 Softwarové architektury

V závislosti na účelu, pro který je konkrétní systém navržen se volí typu procesoru a softwarová architektura. Pro různé účely existuje několik softwarových architektur.

- Jednoduchá nekonečná smyčka

  Program neustále běží ve smyčce bez přerušení.
- Přerušením řízený systém

  Běh smyčky můžeme přerušit například stiskem tlačítka. Po stisknutí program ihned vykoná předdefinovanou úlohu a nezáleží, na kterém místě smyčky se zrovna nachází.

- Ne-preemptivní multitasking
  Každá spuštěná úloha musí v určitých intervalech vracet řízení operačnímu systému, který tak může spouštět další úlohy.
- Preemptivní multitasking
  Přepíná mezi procesy časovačem, umožňuje oddělení procesů a synchronizaci. Stále běží v nekonečné smyčce.
- Mikrojádra
  Umožňují používat základní služby operačního systému, správu paměti, řízení procesů. K dispozici je podpora souborového systému a správa síťových rozhraní.
- Monolitická jádra
  Podpora souborových systémů, síťových služeb, ovladačů zařízení.
  Větší požadavky na hardware
  (Krumnikl, 2006)

2.1.5 Operační systémy pro embedded zařízení

Na různá zařízení můžeme použít některý z operačních systémů určených pro embedded zařízení. Podmínkou pro použití daného operačního systému je kompatibilita s procesorovou platformou. Někdy jsou nutné úpravy samotného operačního systému, aby na zařízení správně komunikoval se všemi periferiemi.

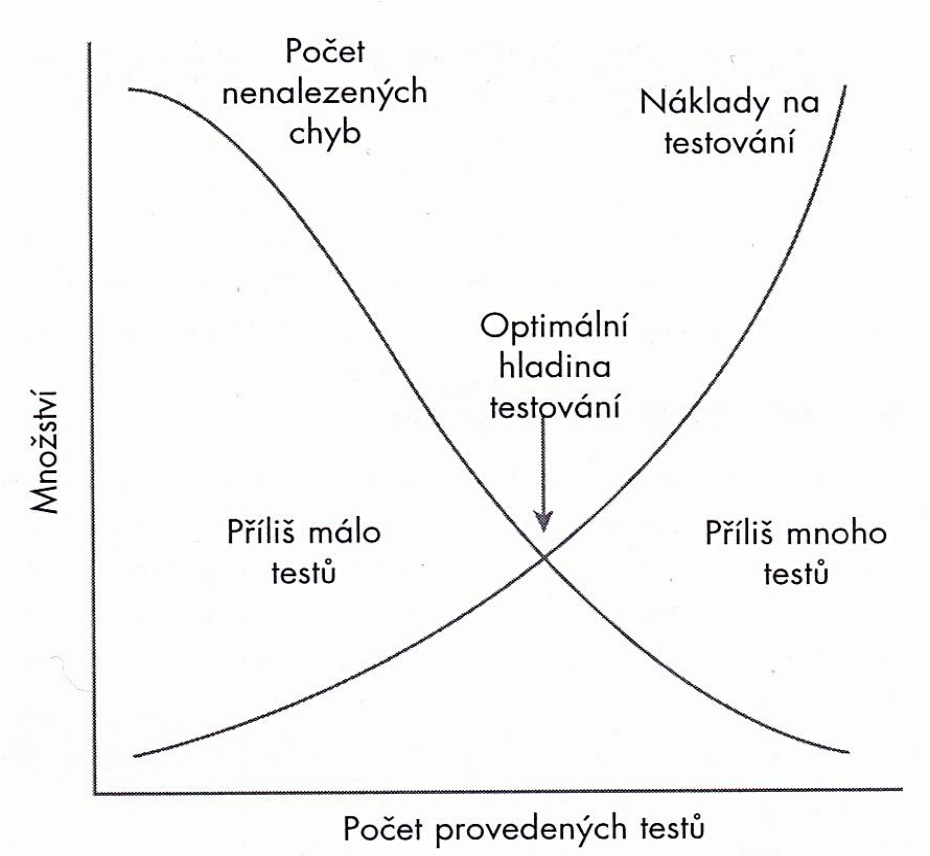
Nejpoužívanější operační systémy

- Linux
  Využívá se především v síťových zařízeních (routery), ale jeho uplatnění je daleko větší např. u autobusů, chytrých televizí, multimediálních přehrávačů a jiných dalších přístrojů.
- Windows CE
  To samé jako u Linuxu. Používá se například v bankomatech, infokioscích.
- Systémy pro chytré telefony (Android, iOS, Windows Phone atd.)

(Krumnikl, 2006), (Bednář, 2002)

2.2. Metody Testování software

Existuje mnoho metod jak testovat software a odhalovat tak jeho kritická místa. Vzhledem k obrovskému množství různých testů, je nejdůležitější částí každého testování zvolení si optimální hladiny testování. Jak je vidět z obrázku č. 2 testů musí být tak akorát, aby byly vůbec uskutečnitelné.



Obrázek 2: Optimální hladina testování (Patton, 2002)

2.3. Testování software metodou White Box (WB)

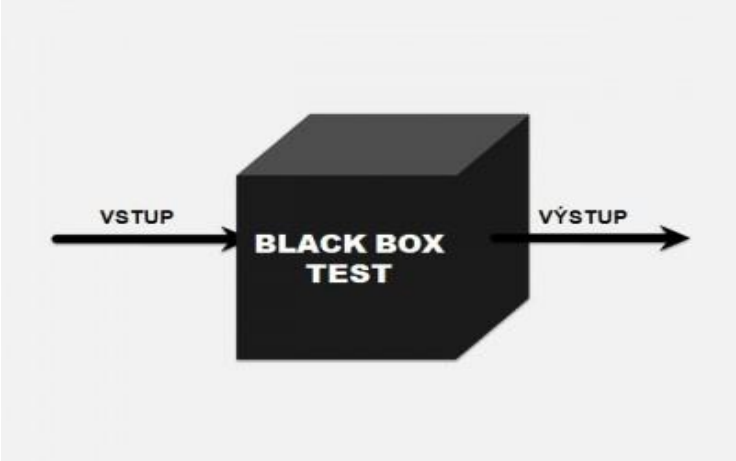
White Box (glass box, clear box nebo open box) testování, jak je už z názvu patrné jde o otevřenou vnitřní strukturu, do které vidíme. Pro testování daného software je tedy k dispozici podrobná dokumentace, specifikace a zdrojový kód. V první řadě je potřeba porozumět samotnému kódu. Za tímto účelem se provede analýza zdrojového kódu, kdy se z dokumentace zjistí, co která část kódu dělá. Poté

může testování probíhat buď zcela automaticky, nebo manuálně. Většinou se oba tyto postupy vhodně kombinují. (Čermák, 2008)

Při testování software metodou White Box lze nalézt velké množství závažných chyb přímo v kódu. Většinou se každý test zaměřuje na ověření, zda aplikace funguje správně a dělá to co má podle daných specifikací. Nedílnou součástí testů by měla být i řádná revize kódu, jestli neobsahuje nějakou škodlivou část nebo zapomenuté ladící části kódu, které programátor zapomněl odstranit. (Čermák, 2008)

2.4. Testování software metodou Black Box (BB)

Black Box, neboli černá skříňka *někdy se také označuje pomocí synonym: behavioral (testing), functional (testing), opaque-box (testing), a closed-box (testing),* je uzavřená soustava, do které nevidíme a nevíme co se uvnitř děje. Black Box má vždy nějaký vstup a výstup (obr. 3) na základě vztahu vstupu a výstupu je možné odvodit, jak daný systém funguje uvnitř. (Hlava, 2011)



Obrázek 3: Schéma černé skříňky (Čermák, 2008)

2.4.1 Dynamické testování

Při dynamickém testování testujeme software za běhu, kdy nemáme k dispozici přístup ke zdrojovým kódům. Metodou Black Box se tedy snažíme zjistit, při jakých vstupních hodnotách se zařízení chová jinak, než jak je uvedeno ve specifikaci. (Muknšnábl, 2001)

Testování probíhá pomocí testovacích scénářů, které jsou buď předem známy, nebo si je tester vytváří sám. Testovací scénáře jsou založeny na znalostech daného systému. Obvykle jsou definovány typy a rozsahy hodnot přípustných a nepřípustných pro daný systém. Na základě zadávaných vstupů očekáváme od aplikace určité výstupy. (Čermák, 2008)

2.4.2 Test splněním (test to pass)

Testování, zda je software schopen dělat, to co dělat má podle specifikace za normálních okolností. U tohoto testu není záměrem vyvolání chyb.

2.4.3 Test selháním (test to fail)

Při testování selháním se vytvářejí situace, kdy je záměrem vynucené vyvolání chyb nebo shozením celého software. Testují se obvyklá a známá slabá místa v software. (Patton, 2002)

2.4.4 Rozdělení tříd na ekvivalence (Equivalence Partitioning)

Metodou „rozdělení ekvivalentních případů“ se redukují rozsáhlé (někdy i nekonečné) množiny možných testových případů na mnohem menší podmnožiny. S těmito podmnožinami se otestují nejkritičtější místa a ignorují se nepodstatné hodnoty, které by zbytečně prodlužovaly testovací čas.

Chová-li se program korektně pro danou reprezentativní hodnotu, předpokladem je, že se bude chovat korektně i pro všechny další hodnoty podmnožiny. Důležité je zařadit do souboru hodnoty platné i hodnoty neplatné, které jsou mimo množinu ze specifikace software. (Muknšnábl, 2001), (Michálek, 2006)

2.4.5 Testování hraničních podmínek

V software se vyskytuje nejvíce chyb v blízkosti hraničních hodnot, proto je důležité vybírat data z obou oblastí koncových hodnot. V praxi se vybírají podmnožiny dat, u kterých ještě předpokládáme správnost výsledků (nacházejí se v intervalu platných dat) a data, která jsou již mimo platný interval. (Patton, 2002)

2.4.6 Subhraniční podmínky

K testování subhraničních podmínek je zapotřebí mít alespoň základní znalosti o činnosti testovaného software. Jako příklady těchto podmínek jsou mocniny dvou (tabulka č. 1) a tabulka ASCII znaků.

Výraz	Interval nebo hodnoty
Bit	0 nebo1
Nibble	0 až 15
Bajt	0 až 255
Slovo	0 až 65 535 nebo 0 až 4 294 967 295
Kilo	1024
Mega	1 048 576
Giga	1 073 741 824
Tera	1 099 511 627 776

Tabulka 1: Softwarové mocniny dvou (Patton, 2002)

Intervaly hodnot v tabulce představují kritické hodnoty, které je potřeba otestovat jako hraniční podmínky. Pokud je ve specifikaci software dán určitý rozsah vstupních dat, testují se subhraniční podmínky v tomto rozsahu. Vysvětlení, proč jsou tyto hodnoty kritické, spočívá v tom, že software pracuje s binárními hodnotami neboli bity. Složením osmi bitů vzniká bajt a ze čtyř bajtů je složeno slovo. Toto uspořádání může být v určitých situacích zdrojem problémů. (Patton, 2002)

Mezi další skupinu subhraničních podmínek můžeme zařadit tabulku ASCII znaků (tabulka č. 2). Kde každý znak má přidělen svůj kód, pokud například aplikace na vstupu přijímá pouze znaky v rozsahu a-z, měla by se aplikace testovat na velká písmena, nebo znaky @,/ .

Znak	Hodnota ASCII	Znak	Hodnota ASCII
Prázdný	0	B	66
Mezera	32	Y	89
/	47	Z	90
0	48	[	91
1	49	`	96
2	50	a	97
9	57	b	98
:	58	y	121
@	64	z	122
A	65	{	123

Tabulka 2: Subhraniční podmínky ASCII znaků (Patton, 2002)

2.4.7 Prázdné údaje

Pokud nastane situace, kdy aplikace žádá vstupní data a uživatel žádná data nezadá nebo zadá nulu, nastává zde problém s vyhodnocením. Zadání nesmyslných vstupů by mělo být správně ošetřeno a neměla by vzniknout chybová situace. Řešením je například doplnění nejmenší platné hraniční hodnoty či zobrazení chybové hlášky. Nicméně i toto je třeba otestovat. Tento soubor hodnot je třeba oddělit od ostatního testování a vytvořit tak samostatnou třídu ekvivalence. Je to kvůli tomu, že software s takovými hodnotami pracuje odlišně než s hodnotami standardními. (Patton, 2002)

2.4.8 Nesmyslné údaje

Zadávání nesmyslných údajů simuluje uživatele, který se chová nelogicky. V případě, že se program ptá na číselné údaje nebo chce od vstupu pouze kladná čísla, zadají se mu přesně opačné hodnoty. Dalším možným testováním je stisk více kláves najednou nebo na vstup, který vyžaduje datum zadat nesmyslně vysoký rok. Test nesmyslných údajů slouží pouze jako doplňkový k ostatním testům. (Michálek, 2006)

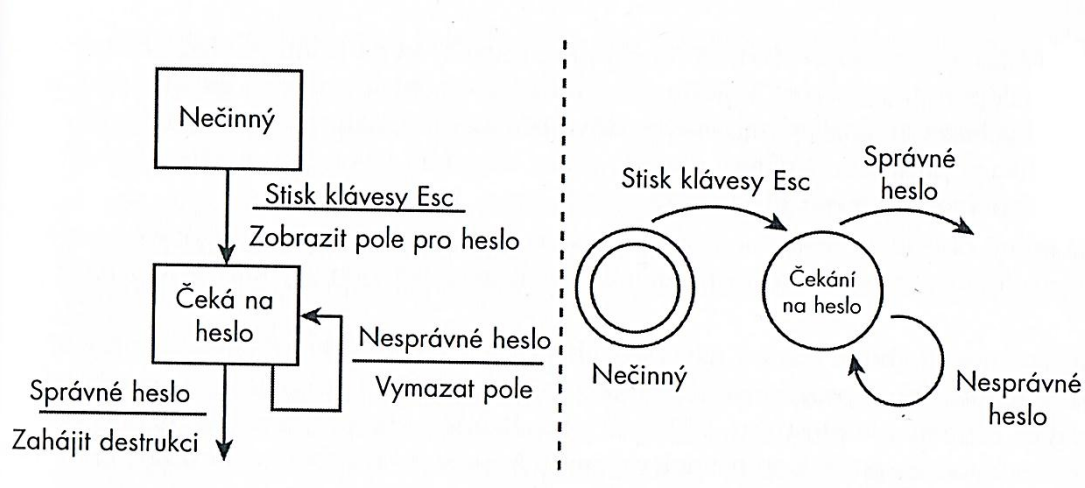
2.4.9 Testování logiky a stavů software

Dalším okruhem testování software je testování stavů (operací), jenž se provádí uvnitř programu. U jednoduchého programu může existovat jen několik stavů, ale se zvyšující se složitostí a variabilitě aplikace počet různých stavů exponencionálně roste. Z toho vyplývá, že otestovat všechny stavy nelze, stejně tak jako otestovat všechna vstupní data. Pro odstranění nepodstatných stavů se použije metoda rozdělení tříd ekvivalence. Tímto krokem se sice zvyšuje riziko neodhalení některých chyb, ale umožňuje zkontrolovat většinu kritických stavů v relativně krátkém čase. Pro přehlednost se sestavuje mapa stavů a přechodů, která zpřehledňuje celý testovací proces. (Muknšnábl, 2001), (Hlavatý, 2003)

Mapa by měla obsahovat:

- Všechny stavy, ve kterých se software může nacházet
- Přechody mezi stavy, které mohou být podmínkou nebo vstupem
  Vstupem může být například vstup ze senzoru nebo stisknutí klávesy.
- Akce, která se vykonává mezi jednotlivými stavy
  Akcí může být například spuštění motoru, pokud z čidla přijde informace o překročení určité hodnoty

Jednoduchá mapa je vidět na obrázku č. 4, která se dá zapsat různými způsoby.



Obrázek 4: Diagramy stavů a přechodů (Patton, 2002)

2.4.10 Testy splněním stavů (test to pass)

Při redukci množiny možných testovaných stavů do rozměrů, které se dají otestovat v přijatelném čase. Postupuje dle několika způsobů.

- Každý stav navštívíme alespoň jednou
  V tomto případě je nutné otestovat každý stav a žádný nevynechat.
- Testování přechodu stavů, které se zdají být neobvyklé
  Účelem tohoto testu je testovat stavy, které jsou nejméně obvyklé nebo se sebou příliš nesouvisí.
- Testování nejméně obvyklé cesty mezi dvěma stavy
- Testy chybových stavů a návraty z nich
  Je to v podstatě testování různých ošetření v programu, například prázdných vstupů nebo nesmyslných podmínek. Po každém vystoupení z chyby by měl být software schopný pracovat dál.
- Náhodné testování přechodů stavů
  Tento způsob se používá spíše u automatizovaných testů, kdy se testují náhodně vybrané přechody.
  (Patton, 2002)

2.4.11 Testy selháním stavů (test to fail)

Testování stavu závodění

Problém závodění vzniká při takzvaném multitaskingu, kdy vedle sebe běží hned několik samostatných procesů najednou. Těmito procesy mohou být samostatné programy nebo program, ve kterém se vykonává více úloh najednou. Ve skutečnosti operační systém přepíná rychle mezi běžícími procesy a z pohledu uživatele se zdá, jako by vedle sebe běžely dva programy najednou. Testování závodění se provádí zejména kvůli správnému načasovaní přerušení zpracování. Pokud se po sobě vyskytne několik událostí, u kterých program nepředpokládá jejich ukončení v průběhu zpracování, vzniká tak situace závodění.

Rizikové situace, kdy může vzniknout závodění:

- Ukládání a načítání ve stejném okamžiku ze dvou různých programů
- Sdílení stejné periferie či komunikačního portu
- Současný přístup různých programů k databázi
- Spouštění stejného programu vícekrát ve stejný čas

Typickým příkladem dnešních embedded zařízení s multitaskingem jsou chytré telefony a tablety.

(Patton, 2002),(Pavlíček, 2012)

Testy opakováním

Testuje především problémy se zpracováním stavů, kdy mohou vznikat díry v paměti (memory leaks). Ke vzniku děr v paměti dochází při vymezení určité části paměti programem pro nějakou operaci a její následné uvolnění, kdy se nemusí uvolnit celá. Při opakovaném spouštění pak dochází k zaplnění paměti a následné zpomalení programu, problémy se stabilitou programu a někdy i se stabilitou celého operačního systému. Testy opakování spočívají v opakování nějaké operace stále dokola (ukládání a načítání dat, spouštění a zavírání programu atd.), dokud se neprojeví chyba. (Patton, 2002)

Testy Zátěží

U těchto testů je důležité otestovat, zda je software schopen pracovat s velkým zatížením při dostatečně výkonném hardware. Účelem je simulovat co možná největší zatížení pro nepřetržitý provoz. Zde je zapotřebí velké množství času a dlouhodobé testování. (Patton, 2002)

Stress testy

K prověření provozu software při nedostatečném výkonu hardware slouží stress testy. Tyto testy omezí prostředky, jako je množství paměti, takt procesoru pro běh programu na minimum. Tímto postupem se zjistí, na kterých externích prostředcích je software nejvíce závislý. (Patton, 2002)

2.4.12 Další techniky testování

Mezi další druhy testů se řadí testy, které slouží pouze jako doplňky testů z předchozích kapitol a měli by odhalit případné další chyby v software.

Hloupý uživatel

Další chyby se mohou projevit, pokud se posadí k zařízení uživatel, který software nezná a po chvíli se mu podaří software shodit zcela nelogickým používáním. Napodobováním chování tohoto uživatele se mohou odhalit zcela skryté chyby neodhalené předchozími metodami.

Hledání chyb, tam kde již nějaké jsou

Chyby, které odhalily předešlé testy na určitých místech software, mohou pravděpodobně v tomto místě obsahovat některé další chyby. Tato místa je nutno zkontrolovat na podobný druh chyb a hraniční podmínky.

Hledání intuicí a předtuchou

Tato tacitní znalost je získána především zkušenostmi a dlouhodobou praxí v testování software. Podle předchozích zkušeností z testování má tester software určitou intuici na chyby a neřídí se přesnými postupy a metodami.
(Michálek, 2006), (Patton, 2002)

2.5. Testování software metodou Grey Box (GB)

Grey Box (translucent box) testování je kombinací obou dvou předchozích testovacích metod. Pro účely testování je známa určitá část vnitřní struktury software avšak bez znalosti zdrojových kódů. V praxi to může představovat testování software přes jeho uživatelské rozhraní. Správnost operací pak ověřujeme pomocí dotazů do databáze. Znalost vnitřních procesů uvnitř programu napomáhá k lepšímu sestavování testovacích scénářů pro metodu Black Box.
(Čermák, 2008), (Hlava, 2011)

2.6. Shrnutí testovacích metod

2.6.1 Metoda White box

Výhody

- Včasné odhalování chyb
  Při analýze zdrojového kódu je možné
  odhalit chyby ještě před jeho kompilací.
- Odhalení nežádoucího kódu

Nevýhody

- Náročnost na znalosti
  Jsou zapotřebí znalosti cílového systému, programovací jazyk a testovací nástroje.
- Vysoké náklady
  Je vyžadován speciální software na analyzování kódu.

(Čermák, 2008)

2.6.2 Metoda Black Box

Výhody

- Snadnost
  Není zapotřebí znalosti operačního systému nebo programovacích jazyků
- Rychlost
  Umožňuje v krátké době otestovat i rozsáhlejší systémy

- Transparentnost
  Spočívá ve snadné pochopitelnosti testovacích scénářů bez větších znalostí programovacího jazyka.
- Testování není závislé na platformě
  Nezáleží na operačním systému, hardware nebo programovacím jazyku, testy budou probíhat vždy stejně.
- K testům nejsou potřeba zdrojové kódy

Nevýhody

- Nižší kvalita kódu
  Vzhledem k tomu že není k dispozici zdrojový kód, aplikace může fungovat správně, ale kód může být napsán neefektivně.
- Nežádoucí chování aplikace
  Aplikace může provádět akce, které nejsou uvedeny ve specifikaci, a na standardním výstupu se hodnota této akce neobjeví.

(Čermák, 2008)

2.6.3 Metoda Grey Box

Výhody

- Slučování výhod metody White Box a Black Box
- Není potřeba znát Zdrojový kód
  Je však známa specifikace a vnitřní architektura aplikace

Nevýhody

- Nižší kvalita kódu
  Stejný problém jako u Black Boxu, kde není možné otestovat všechny datové toky.
- Aplikace může obsahovat nedoladěný nebo nežádoucí kód

(Čermák, 2008)

3. Praktická část práce

Jak už bylo zmíněno v úvodu, testovaným objektem této práce je váhový počítač Computer scale RM plus firmy AP-EL, který slouží především pro nakládky a vykládky krmiva v zemědělství. V praxi je váhový počítač o rozměrech 260×170×70 mm umístěn v kabině nebo na boku traktoru. Napájení váhového počítače je bráno přímo z baterie traktoru. Kabely od jednotlivých tenzometrických senzorů nejdříve vedou do slučovacího boxu a poté z tohoto boxu vede už pouze jeden kabel přímo do váhového počítače. Při testování firmware tohoto zařízení a jeho obslužného programu se předpokládá správné fyzické zapojení všech komponentů.

Při testování může být použita pouze metoda Black Box, jelikož nejsou dostupné zdrojové kódy a není známo ani vnitřní fungování software. K testování jsou dostupné pouze obecné specifikace v podobě manuálu k váhovému počítači a obslužnému programu. S přihlédnutím k těmto skutečnostem byly zvoleny takové testy, které lze provést za rozumný čas a s relevantními výsledky.

3.1. Předpokládané výsledky

Vzhledem k několika případům, kdy zákazník narazil na chyby při běžném používání je pravděpodobné, že jak v samotném firmware váhového počítače, tak i obslužném software budou testy odhaleny nějaké další chyby.

3.2. Technické parametry zařízení

Technické parametry zařízení, na kterých byly testy prováděny, jsou uvedeny v dalších podkapitolách.

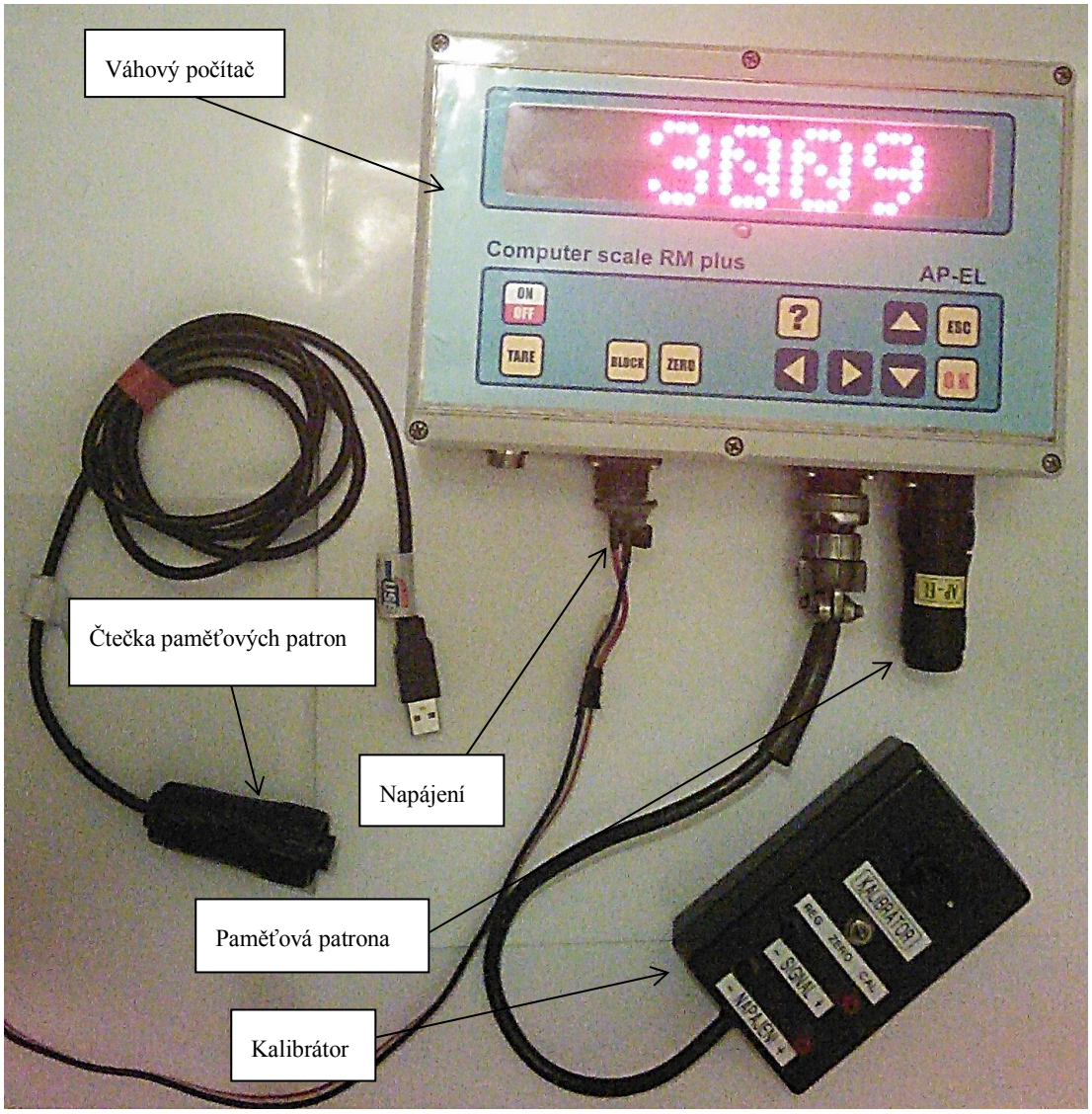
3.2.1 Váhový počítač (Computer scale RM plus)

Zařízení je prototypem z roku 2008 s novějším firmware ve verzi 0.67.
V zásadě se ale jedná o srovnatelný produkt, který se běžně prodává. Bližší technické parametry jsou v tabulce č. 3.

Parametr	Hodnota
Rozsah vážení	0 až 65535 kg
Nastavitelné rozlišení	1-2-5-10 kg
Přesnost (max. odchylka)	<+/-0,015% full scale
Počet receptů	40
Počet komponent	40
Počet stájí	20
Počet komponent na recept	20
Dynamický maticový LED displej	zobrazení 10 čísel i textu v matici 5×7
Připojení externího displeje	Displej PTM typ AV20/5 IM
Hmotnost	950 g
Rozměry	260×170×70 mm
Napájecí napětí	9 – 25 V ss
Pracovní teplota	-20 / +50 °C

Tabulka 3: Technické parametry váhového počítače (Computer scale RM plus)

K váhovému počítači je možné připojit externí paměť tak zvanou paměťovou patronu, která má kapacitu paměti 512 kB, což představuje nahrání 40 receptur a 2771 událostí. Patronou je možno přenášet recepty do váhového počítače nebo opačně. Pokud je patrona připojena, všechny události se zaznamenávají do její paměti. Při odpojené patroně jsou události zaznamenány do vnitřní paměti váhového počítače a jejich následné přenesení do PC lze provést právě přes patronu. Na obrázku č. 5 je vidět zapojená testovací sestava s kalibrátorem.



Obrázek 5: Testovaný váhový počítač (Computer scale RM plus)

3.2.2 PC s obslužným programem

Vytváření vlastních receptů pro vážení lze provádět buď přímo v uživatelském rozhraní váhy, nebo obslužným programem „Precizní krmení“ na PC. Z pohledu přehlednosti a pohodlnosti je lepší vytvářet recepty na PC a následně je pomocí paměťové patrony přenést do váhy. Základní konfigurace PC, na kterém se software a hardware pro váhový počítač testoval je uvedena v tabulce č. 4.

Parametr	Hodnota
Výrobce	Lenovo
Model	G560
CPU	Intel Core i3 M350 2,27GHz
RAM	4GB
Grafická karta	Nvidia GeForce 310M
HDD	1TB
OS	Microsoft Windows Ultimate SP1

Tabulka 4:Konfigurace testovacího PC

Na tomto PC byl testován nainstalovaný obslužný program „Precizní krmení“ ve verzi 1.0.2.20.

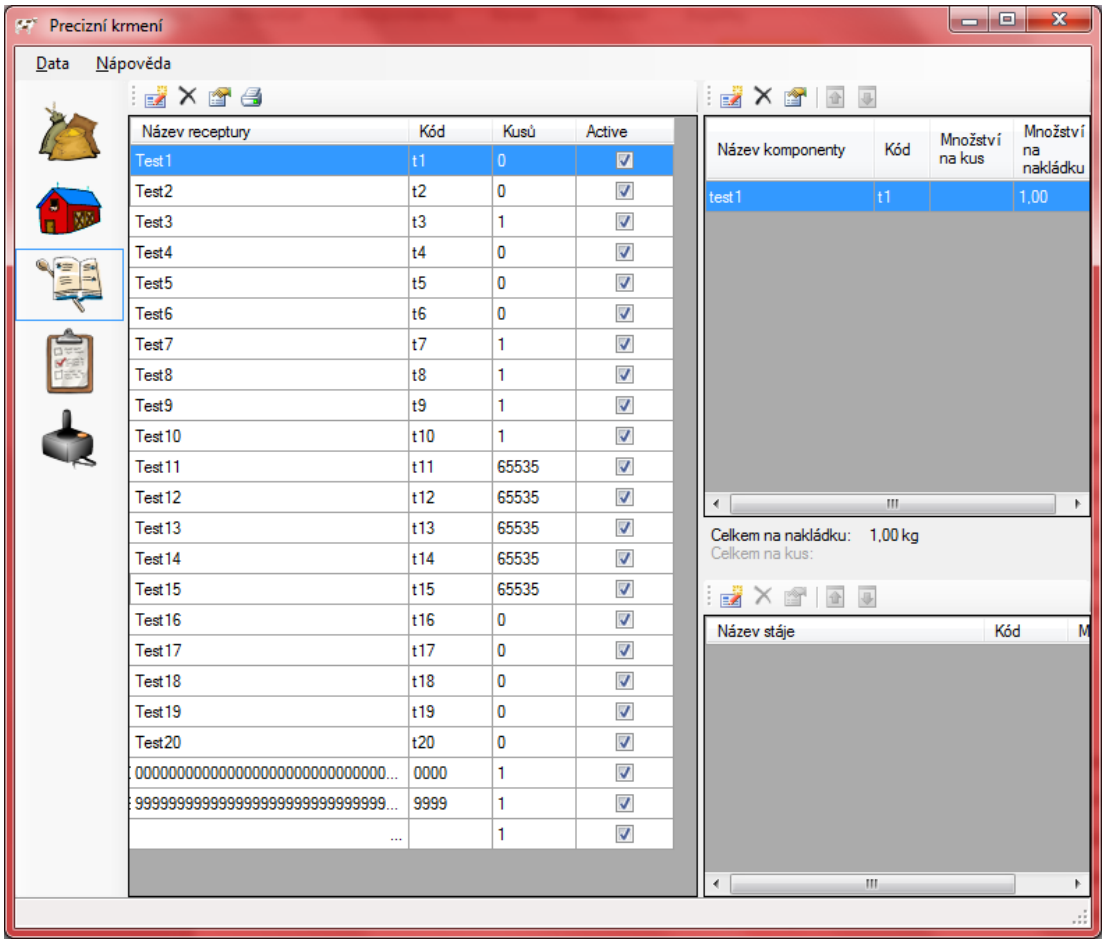
3.1. Testování firmware váhového počítače

Sestavování vhodných testovacích množin se provádělo na základě vybraných testovacích metod zmíněných v teoretické části této práce. Samotné testování proběhlo na váhovém počítači (Computer scale RM plus), kde nebyli použity žádné automatizované testovací nástroje. Jelikož zadávání testovacích souborů probíhalo přes obslužný program „Precizní krmení“, byl tento program rovněž otestován na stejné testovací sady jako firmware váhového počítače. Nalezené chyby v obslužném programu jsou uvedeny zvlášť v kapitole 3.2.

3.1.1 Testování datových hraničních podmínek

Názvy testů jsou pro přehlednost stejné, jako názvy receptur uložené v obslužném programu. Na obrázku č. 6 je ukázka programového prostředí pro tvorbu receptur s připravenými testy k nahrání do paměťové patrony. První část testů se týkala především číselných hraničních podmínek. Vzhledem k nestandardním

hodnotám, které byly při testu použity, je téměř nepravděpodobné, že by taková data uživatel zadal. Nicméně i takovéto hodnoty je třeba prověřit.



Obrázek 6:Ukázka uživatelského prostředí programu "Precizní krmení"

Popis průběhu testu:

V obslužném programu pro váhový počítač bylo vytvořeno několik testovacích komponent a stájí. Poté se na kartě receptury vytvořily příslušné testovací receptury se zadanými hraničními podmínkami, viz obrázek č. 6. Poté se celý soubor testovacích receptur nahrál do paměťové patrony. Paměťová patrona se následně vložila do váhového počítače. Po načtení patrony se v menu zvolil režim nakládky a jednotlivé recepty se postupně navážily. Z hlediska časové úspory se tam, kde to vyloženě test nevyžadoval, zadávaly nulové hodnoty navážení.

Po otestování všech nahraných receptů se data z paměťové patrony stáhla do PC z důvodu ověření správného zápisu na paměťovou patronu.

Časová náročnost všech testů by neměla přesáhnout tří hodin čistého času.

Testovací soubor - Test1, Test2

Testování dolních hraničních podmínek, kdy údaj o počtu kusů a množství za kus je nulový. Množství na nakládku a celková hmotnost nakládky u Testu1 byla 1 kg a u Testu2 0 kg.

Předpokládané chování software:

Při zvolení položky Test1 v menu nakládka se nakládání automaticky ukončí a nic se nenaměří, při volbě Testu2 na nakládce bude vyžadovat odvážení jednoho kilogramu

Skutečné chování software:

Skutečné chování software je stejné jako u předpokladu.

Testovací soubor - Test3,Test10

Testování dolních hraničních podmínek, kdy údaj o počtu kusů a množství za kus je jedna. Množství na nakládku a celková hmotnost nakládky u Testu3 byla 1 kg a u Testu10 0 kg.

Předpokládané chování software:

Při zvolení položky Test3 v menu nakládka se nakládání automaticky ukončí a nic se nenaměří, při volbě Testu10 na nakládce bude vyžadovat odvážení 1 kg.

Skutečné chování software:

Skutečné chování software je stejné jako u předpokladu.

Testovací soubor - Test4,Test5,Test6

Horní hraniční podmínky se testují na krajní hodnotu 65535. Tato hodnota je největší hodnotou, kterou povolí obslužný program zapsat jako hodnotu množství na nulu kusů. Testovací soubor obsahuje samotnou krajní hodnotu 65535 a dvě nejbližší

krajní hodnoty (65534,65536). Přičemž hodnota 65536 je celková hmotnost nakládky zadaná jako dvě komponenty. První komponenta má hmotnost 65535 kg a druhá 1kg.

Předpokládané chování software:

Při zvolení položky Test4 bude software vyžadovat naložení 65535 kg první komponenty. Zvolením položky Test5 by software měl vyžadovat naložení 65534 kg první komponenty. A zvolením položky s názvem Test6 by měl software zahlásit chybu kvůli překročení maximální hodnoty. Další možností je zobrazení navážení první komponenty 65535 a následné navážení druhé komponenty s hodnotou 1.

Skutečné chování software:

V případě Testu4 a Testu5 se software chová, jak bylo uvedeno v předpokladu. U Testu6 software zobrazí na displeji první komponentu (65535 kg) a po potvrzení komponentu 1kg. Test skončí bez chyb.

Testovací soubor - Test7,Test8,Test9

V podstatě se jedná o podobný test, jako test předešlý. Změna je pouze v počtu kusů, kdy u tohoto testu je zadán jeden kus. Testovací soubor obsahuje krajní hodnotu 65535 a dvě nejbližší krajní hodnoty (65534,65536). Hodnota 65536 je celková hmotnost nakládky zadaná jako dvě komponenty. První komponenta má hmotnost 65535 kg a druhá 1kg.

Předpokládané chování software:

Při zvolení položky Test7 bude software vyžadovat naložení 65535 kg první komponenty. Zvolením položky Test8 by software měl vyžadovat naložení 65534 kg první komponenty. A zvolením položky s názvem Test9 by měl software zahlásit chybu kvůli překročení maximální hodnoty. Další z možností je zobrazení navážení první komponenty 65535 a následné navážení druhé komponenty s hodnotou 1.

Skutečné chování software:

Software váhového počítače v případě Testu7 a Testu8 vyžaduje naložení 655 kg. U Testu6 software rovněž vyžaduje nakládku 655 kg, po následném potvrzení vyžaduje již správnou hodnotu komponenty 1 kg. Test končí opětovným návratem do hlavního menu.

V tomto testu se předpokládaný výsledek rozchází s výsledkem skutečným. Požadavek váhového počítače na nakládku komponenty o hodnotě 655 kg se neshoduje s hodnotou, která byla zadána v obslužném programu (655535 kg). Tato rozdílnost požadavků může být způsobena omezením maximální hmotnosti (655 kg) komponenty na jeden kus. Podle výše zmíněných skutečností se jedná o chybu, jelikož požadovaná hodnota komponenty na váhovém počítači se liší od hodnoty zadávané uživatelem v obslužném programu.

Testovací soubor - Test11

V tomto testu se testuje hraniční podmínka u počtu kusů, kdy největší dovolený počet kusů je 65535 a nulová hmotnost na kus. Celková hmotnost nakládky tedy činí 0 kg.

Předpokládané chování software:

Vzhledem k celkové nulové nakládce komponent by se měla nakládka automaticky přeskočit a ukončit.

Skutečné chování software:

Chování je stejné jako u předpokladu.

Testovací soubor - Test12

Recept Test12 testuje hraniční podmínku počtu kusů, kdy je zadán největší dovolený počet kusů (65535 ks) a hmotnost na kus činí 1 kg. Celková hmotnost nakládky je tedy 65535 kg.

Předpokládané chování software:

Při nakládce by měla váha vyžadovat naložení 65535 kg dané komponenty.

Skutečné chování software:

Chování software je stejné jako u předpokladu.

Testovací soubor - Test13

Testovací soubor Test13 navazuje na předchozí testovací soubory, kdy počet kusů je 65535. Hmotnost komponenty na kus jsou dva kilogramy a celková hmotnost činí 131070 kg.

Předpokládané chování software:

Při zvolení položky Test13 v menu nakládka by měl váhový počítač zahlásit chybu o překročení maximální hodnoty nebo požadovat nakládku celkové hmotnosti 131070 kg.

Skutečné chování software:

Váhový počítač zobrazí na displeji chybu „Hmotnost komponenty je větší než 65535kg!“ a automaticky skončí s nulovou hmotností.

Testovací soubor - Test14

Test14 je zaměřen na horní hraniční hodnoty kusů a hmotnosti na kus. U položky kus je zadaný počet kusů 65535 a hmotnost komponenty na kus je rovněž 65535. Celková hmotnost je tedy 4294836225 kg na jednu komponentu. V receptu je pouze tato jedna komponenta, celková hmotnost nakládky je tedy stejná jako hmotnost komponenty.

Předpokládané chování software:

Zvolením Testu14 v menu nakládky by měla váha zahlásit chybu o překročení maximální hodnoty nebo požadovat nakládku celkové hmotnosti 4294836225kg.

Skutečné chování software:

Na displeji se zobrazí chybová hláška „Hmotnost komponenty je větší než 65535kg!“, poté se automaticky ukončí nakládka.

Testovací soubor - Test15

Posledním testem podobným předchozím čtyřem typům je test, který testuje největší hmotnost celkové nakládky 85896724500 kg, a kterou nám obslužný software dovolí zadat a následně nahrát do paměťové patrony. Této hodnoty lze docílit při naplnění maximálního možného počtu komponent na jeden recept, kdy každá komponenta nabývá hodnoty 65535 kg a počet kusů má rovněž tuto hodnotu. Program dovoluje maximálně 20 komponent na recept.

Předpokládané chování software:

Tak jako u předchozích dvou testů by měla váha zahlásit chybu o překročení maximální hodnoty nebo požadovat nakládku celkové hmotnosti 85896724500 kg.

Skutečné chování software:

Potvrzením nakládky receptu Test15 se na displeji zobrazí chybová hláška „Hmotnost komponenty je větší než 65535kg!“, poté váhový počítač začne procházet recepty od položky receptu T2 (Test2 se nachází na druhé pozici) a při každém průchodu zobrazí chybovou hlášku „Hmotnost komponenty je větší než 65535kg!“, takto prochází všechny pozice popořadě, až do T20 (Test20 = dvacátá pozice)- v tomto bodě zobrazí hlášku“ Nakládka ukončena. Celkem naloženo: 0kg“. Po této hlášce se zobrazí hlavní menu.

V tomto případě se rozhodně jedná o nestandardní chování software. Jelikož jsou zadávány extrémní okrajové hodnoty, tak by tato chyba neměla při běžném užívání software nastat.

Testovací soubor - Test16

Nulové hodnoty na všech pozicích (počet ks, množství na ks, množství na nakládku, množství na stáj)

Předpokládané chování software:

Nakládka komponenty by měla být ihned ukončena, jelikož obsahuje nulovou hmotnost.

Skutečné chování software:

Chování software je stejné jako u předpokladu.

Testovací soubor - Test21, Test22,Test23

Ve všech třech testech budou všechna pole receptu zaplněna stejnou hodnotou. U Testu21 budou zaplněna všechna pole receptury nulou. V Testu22 budou všechna pole obsahovat hodnotu 9 a Test23 bude mít pole zaplněna všechna pole mezerníkem.

Předpokládané chování software:

U Testu21 by měly být v kódech receptury a komponenty zobrazeny čtyři nuly. V kódu Testu22 by to měly být čtyři devítky a v Testu23 čtyři prázdná pole. U všech těchto testů by se měla nakládka automaticky ukončit s nulovou hmotností.

Skutečné chování software:

V případě Testu21 a Testu22 se vše zobrazí tak, jak bylo řečeno v předpokládaném chování software. Na místě zobrazeného receptu Testu23 se zobrazí číslo 23 a nakládka se automaticky ukončí.

Testování vnitřního data a času váhy

Dalším kritickým místem, kde by se mohly vyskytnout chyby v hraničních podmínkách je datum a čas. V případě váhového počítače je možné datum a čas nastavit v servisním menu. Časové údaje se pokaždé ukládají s příslušným záznamem o vážení, a to jak do vnitřní paměti, tak i na paměťovou patronu.

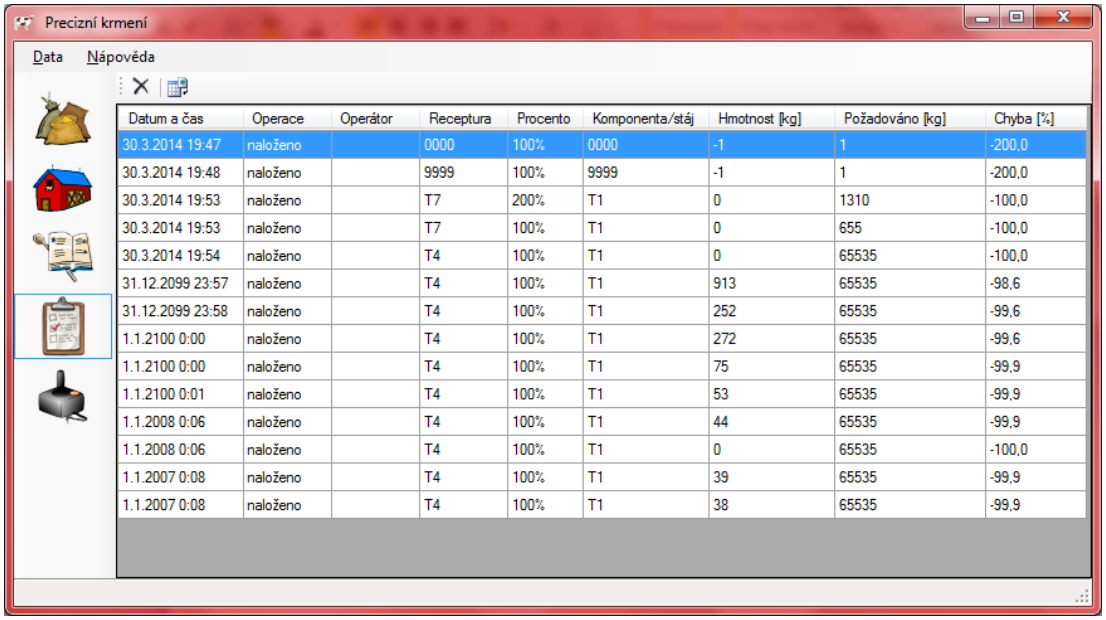
V servisním menu se nastaví čas a datum na 31.12.2099 23:59, tyto údaje nám dovolí ve svém prostředí váhový počítač nastavit jako nejvyšší.

Předpokládané chování software:

Po uplynutí jedné minuty se datum a čas resetuje na 1.1.2000 00:00 nebo přejde do následujícího roku 1.1.2100 00:00.

Skutečné chování software:

Na obrázku č. 7 je vidět výpis žurnálu z paměťové patrony, kde je zobrazen přechod z roku 2099 na rok 2100. Datum se tedy nastaví na rok 2100, tento rok se v servisním menu zobrazuje jako 00. Pokud se ale nastaví takovýto rok ručně, na datovou patronu se nezapisují údaje o vážení. To platí i pro roky následující (01,02,03,04,05,06) od roku 07, což je rok 2007 je zápis na datovou patronu opět funkční.



Datum a čas	Operace	Operátor	Receptura	Procento	Komponenta/stáj	Hmotnost [kg]	Požadováno [kg]	Chyba [%]
30.3.2014 19:47	naloženo		0000	100%	0000	-1	1	-200,0
30.3.2014 19:48	naloženo		9999	100%	9999	-1	1	-200,0
30.3.2014 19:53	naloženo		T7	200%	T1	0	1310	-100,0
30.3.2014 19:53	naloženo		T7	100%	T1	0	655	-100,0
30.3.2014 19:54	naloženo		T4	100%	T1	0	65535	-100,0
31.12.2099 23:57	naloženo		T4	100%	T1	913	65535	-98,6
31.12.2099 23:58	naloženo		T4	100%	T1	252	65535	-99,6
1.1.2100 0:00	naloženo		T4	100%	T1	272	65535	-99,6
1.1.2100 0:00	naloženo		T4	100%	T1	75	65535	-99,9
1.1.2100 0:01	naloženo		T4	100%	T1	53	65535	-99,9
1.1.2008 0:06	naloženo		T4	100%	T1	44	65535	-99,9
1.1.2008 0:06	naloženo		T4	100%	T1	0	65535	-100,0
1.1.2007 0:08	naloženo		T4	100%	T1	39	65535	-99,9
1.1.2007 0:08	naloženo		T4	100%	T1	38	65535	-99,9

Obrázek 7: Výpis událostí z paměťové patrony

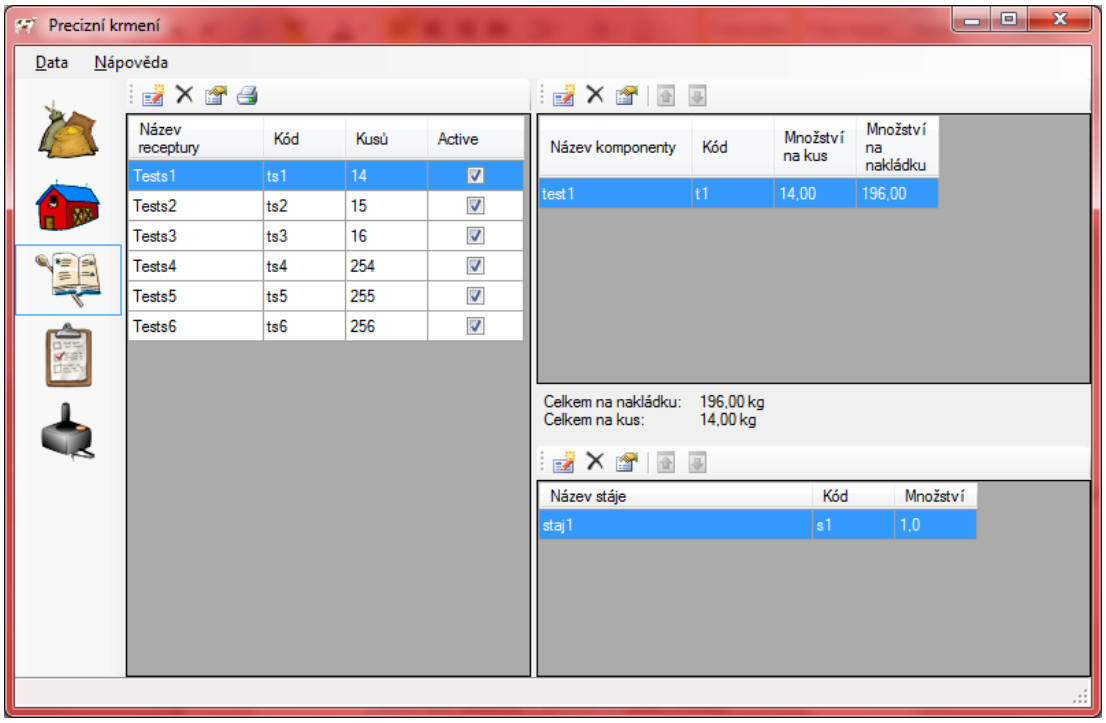
Při testování data v roce 2005 (05) se data na patroně pravděpodobně špatně zapsala a při stahování žurnálů se zobrazila chybová hláška zobrazená na obrázku č. 8 Tato chyba se podruhé nepodařila zopakovat, avšak při tomto nastaveném roce se na datovou patronu stále nic nezapisovalo. Vlivem chybného zápisu na paměťovou patronu nebylo již možné číst předchozí uložená data a datová patrona musela být zformátována pro další testování.

Obrázek 8: Chybová hláška při načítání dat z paměťové patrony

Celkový čas všech provedených testů hraničních podmínek nepřesáhl dvě hodiny.

3.1.2 Testování číselných subhraničních podmínek

Testování číselných subhraničních podmínek je zaměřeno v těchto testech na mocniny dvou, které mohou způsobovat chyby v programu. Testovací soubory mocnin dvou jsou brány v rozsahu váhového počítače od 0 až do 65535. Ukázka souboru testů pro testování subhraničních podmínek se nachází na obrázku č. 9.



Obrázek 9: Testování subhraničních podmínek

Odhadovaná časová náročnost testů na subhraniční podmínky je 45 minut.

Testovací soubor - Tests1,Tests2,Tests3

Test subhraničních podmínek pro hodnoty 14,15,16 (Nibble). Do všech polí, kde obslužný software a váhový počítač nějakým způsobem provádí další operace, se zapíší tyto hodnoty.

Předpokládané chování software:

Ve všech případech dojde k navážení hmotnosti komponent a zápisu vážení do paměti.

Skutečné chování software:

Jednotlivé komponenty jsou postupně naváženy bez chyb.

T**estovací soubor - Tests4,Tests5,Tests6**

Test subhraničních podmínek pro hodnoty 254, 255, 256 (Bajt). To samé jako u předchozího testu, akorát s jinými hodnotami mocnin dvou.

Předpokládané chování software:

Ve všech třech případech by mělo dojít k navážení dané hmotnosti komponent a zápisu vážení do paměti.

Skutečné chování software:

Vše proběhlo totožně, jako u testu předchozího.

Subhraniční podmínky pro okolí čísla 65535 bylo testováno již v předchozím testování hraničních podmínek.

3.1.3 Testování znakové sady (subhraniční podmínky)

K testování znakové sady byla zvolena znaková sada unicode s vybranými sadami znaků. Testovací soubory tvoří vždy čtveřice znaků (displej váhového počítače dokáže zobrazit čtyři znaky), které jsou vloženy do všech polí, které umožňují vkládání znaků. Testování se provádí jak v režimu nakládky, tak i vykládky. Jednotlivé testovací sady znaků jsou uvedeny v tabulce č. 5.

Pořadí testovací sady	Hodnota
1. sada	/@{[
2. sada	]'"`
3. sada	$#%&
4. sada	*+,¢
5. sada	£©§Ã
6. sada	~µ£Й
7. sada	¡¢§
8. sada	ª¬º2
9. sada	»ßö→
10. sada	Ω♪►π
11. sada	ěšůá

Tabulka 5: Sady testovacích znaků

Testovací soubor – 1. sada, 2. sada, 3. sada

U první až třetí testovací sady se znaky na displeji zobrazují korektně pouze u nakládaných komponent. V menu s receptem jsou místo znaků zobrazeny podtržítka. Nezobrazení příslušných znaků receptů nemá vliv na správnost provedené nakládky.

Testovací soubor – 4. sada, 6. sada

Kód komponenty se u 4. sady zobrazí korektně až na poslední znak, kdy se namísto čtvrtého znaku zobrazí symbol otazníku. V kódu komponenty 6. sady je zobrazen správně pouze první znak. Samotné navažování nakládky je bezproblémové.

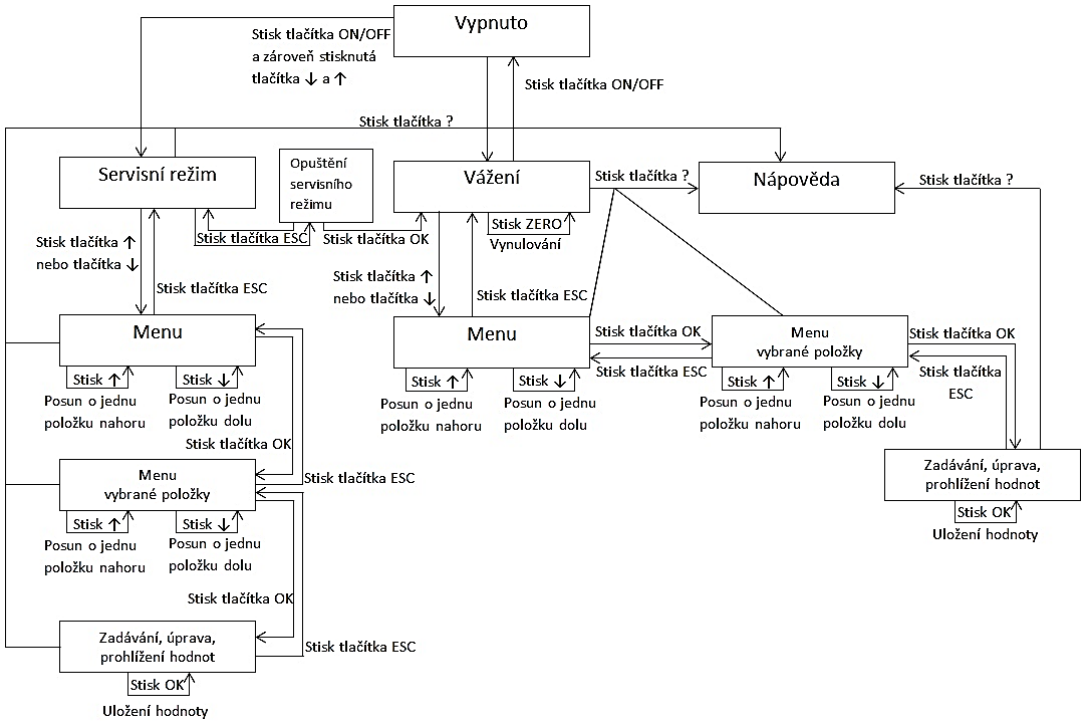
Testovací soubor – 5. sada, 7. sada, 8. sada, 9. sada, 10. sada, 11. sada

V případě tohoto souboru jsou ve všech kódech komponent místo příslušných znaků zobrazeny symboly otazníků v kódu receptu to pak jsou podtržítka. Chybné zobrazení nemá vliv na správné navážení nakládky.

Závažnější chyba nastala při testování 11. sady, kdy nešly žurnály z paměťové patrony stáhnout do PC. Obslužný program zobrazil chybovou hlášku „chybný formát dat“ obrázek č. 8 z patrony poté už nešlo číst žádná data a musela být zformátována. Tuto chybu se napodruhé nepodařilo zopakovat, tak jako v předchozím případě u testování časových údajů v kapitole 3.1.1.

3.1.4 Testování stavů

Pro testování stavů, jak na váhovém počítači, tak v obslužném programu byla zvolena metoda testu splněním (test to pass), kdy každý stav, který by mohl nastat, musí být navštíven alespoň jednou. Diagramu testovaných stavů pro váhový počítač je uveden na obrázku č. 10.

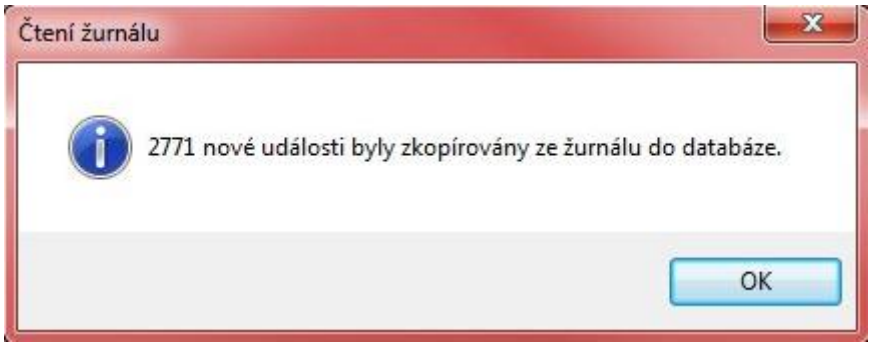


Obrázek 10: Stavový diagram váhového počítače (Computer scale RM plus)

V průběhu testování přechodů a stavů na váhovém počítači nebyly nalezeny žádné chyby. Čistý čas testování stavů se pohybuje okolo 1,5 hodiny.

3.1.5 Test zaplnění paměti

Zaplněním paměťové patrony a vnitřní paměti váhového počítače se testovalo ukládání dat, které probíhá na principu cyklického bufferu, kdy po zaplnění celé paměti se první uložená data začnou mazat a vzniká tak místo pro data nová. Nevýhoda této metody spočívá ve ztrátě prvních uložených dat, pokud se neprovádí pravidelná záloha ještě před úplným zaplněním. Jak je patrné z obrázku č. 11 počet událostí, které se vejdou na paměťovou patronu je 2771. Zaplnění paměti bylo testováno i na hraničních podmínkách, kdy počet událostí při čtení byl 2770,2771. Testování zaplnění paměti proběhlo jak u vnitřní paměti váhového počítače, tak u paměťové patrony bez chyb. Časová náročnost tohoto testu je bezesporu největší. Jedno zaplnění paměti nulovými hodnotami trvá vzhledem k nemožnosti urychlení tohoto testu kolem jedné hodiny. Po každém přečtení a uložení dat z paměťové patrony do PC je celá paměť smazána a musí se začít plnit znovu. Celková náročnost tohoto testování ve výsledku dosahovala něco kolem šesti hodin neustálého plnění paměti.



Obrázek 11: Informační okno obslužného programu při plně zaplněné paměťové patroně

3.2. Chyby nalezené v obslužném programu

Obslužný program nebyl přímo náplní této práce, ale jen během používání bylo zaznamenáno několik chyb. K odhalení chyb docházelo především při testování různých přechodů do stavů metodou splnění (test to pass).

Časově testování stavů v obslužném programu odpovídá testování stavů u váhového počítače.

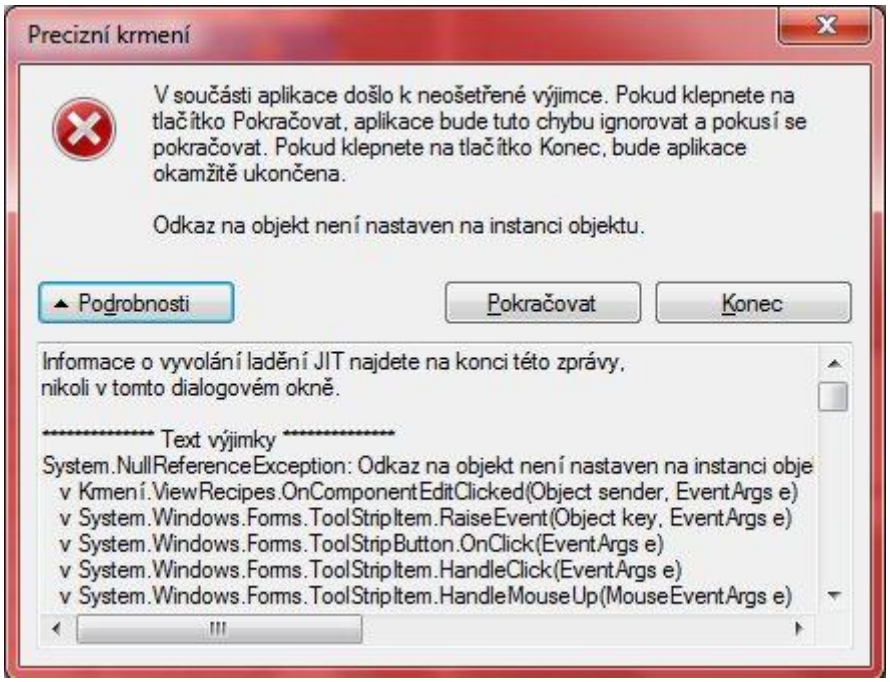
Minimalizace okna programu

Při minimalizaci okna na lištu a následného vyvolání programu z lišty se okno zmenší na nejmenší možnou míru viz obrázek č. 12. Tento problém není kritický, ale práci s programem značně komplikuje, co se týče rychlosti. Po každé minimalizaci musí uživatel znovu roztahovat okno na rozumnou velikost.

Obrázek 12:Chybové zobrazení okna po minimalizaci do lišty

Neošetřený přechod u změny komponenty

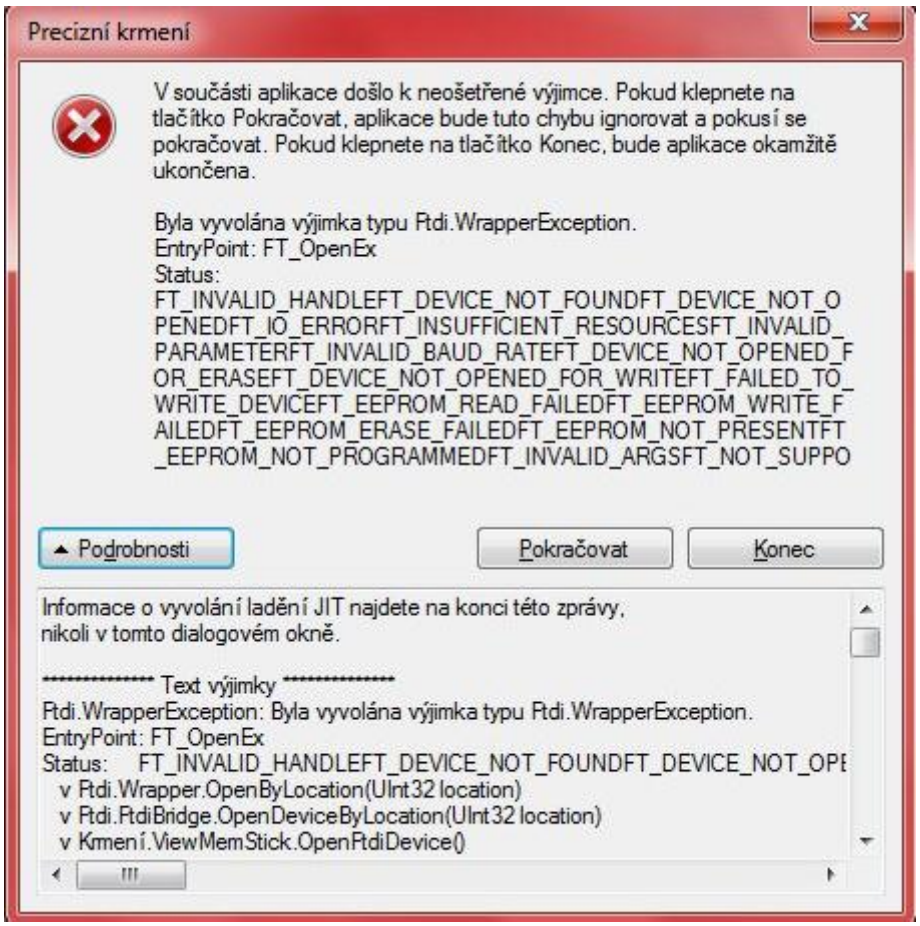
Chyba nastává při kliknutí na tlačítko „Změnit množství“ u komponenty, pokud je v receptuře uveden nulový stav kusů. Program automaticky zobrazí kritickou chybovou hlášku viz obrázek č. 13 .



Obrázek 13: Chybová hláška při pokusu o změnu množství u komponenty

Čtení z více spuštěných programů

Pokud má uživatel spuštěn program „Precizní krmení“ vícekrát a začne z obou programů číst najednou data z paměťové patrony, automaticky se zobrazí hláška viz obrázek č. 14 o neošetřené výjimce.



Obrázek 14: Chyba při vícenásobném pokusu čtení z paměťové patrony

Další chyby

Mezi další chyby jsou zařazené ty chyby, které se vyskytly během testování a používání programu, ale napodruhé se je již nepodařilo vyvolat. Takovouto chybou byl pád programu při vytváření nové komponenty, kdy před touto komponentou bylo vytvořeno šest různých komponent. Další kritickou chybou bylo zamrznutí programu při přechodu z jednoho receptu na následující. Program nejde ukončit běžnou cestou, jako je ukončení procesu. Tyto chyby mohly být způsobeny též danou konfigurací PC a je možné, že se u jiných PC nevyskytnou.

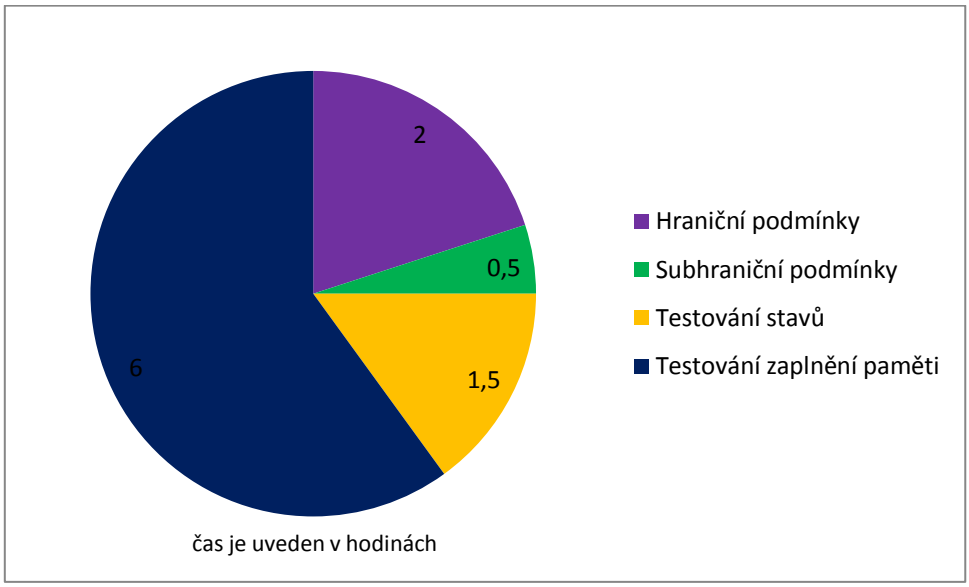
4. Zhodnocení výsledků testů

Jak už bylo uvedeno v předchozích kapitolách, jedinou možnou metodou, která se hodí na testování cílového objektu této práce je metoda Black Box. I přes neznalost zdrojového kódu a fungování vnitřní struktury software byl nalezen dostatek různých chyb. V zásadě se však nejednalo o chyby, které by poškodily zařízení nebo donutily watchdog timer k restartu. Ve většině případů se jednalo o neošetřené hraniční podmínky nebo povolení zadání nesmyslných hodnot k vážení. Příkladem může být testovací soubor s Testem15, kdy obslužný program dovolil nahrát na paměťovou patronu recept s celkovou nakládkou o hmotnosti 85896724500 kg. Další větší problém nastává s nastavením starého data, kdy se při zadání roku 2006 a níž přestanou na paměťovou patronu zapisovat události z vážení. Chyby v obslužném programu, které se dají znovu nasimulovat, jsou rovněž způsobeny neošetřenými výjimkami.

4.1. Časová náročnost použitých testů

S rozsáhlostí software stoupá i náročnost použitých testů na čas. Nejvíce času se stráví nad sestavováním testovacích souborů a následným testováním. V případě této práce nebyly použity žádné automatizované nástroje k testování. Všechny testy byly napsány a ručně zadány jak do váhového počítače, tak i obslužného programu. Tento postup není zdaleka nejefektivnějším, ale postačuje na nepříliš rozsáhlé systémy a je z hlediska ekonomické náročnosti přijatelný. Odpadá tedy investice do profesionálního testovacího nástroje. Z grafu č. 1 je vidět, že nejnáročnějším testem, co se týče časové náročnosti byl test zaplnění paměti a hned za ním testování jednotlivých přechodů a stavů. Naopak nejméně času zabral test subhraničních podmínek mocnin dvou.

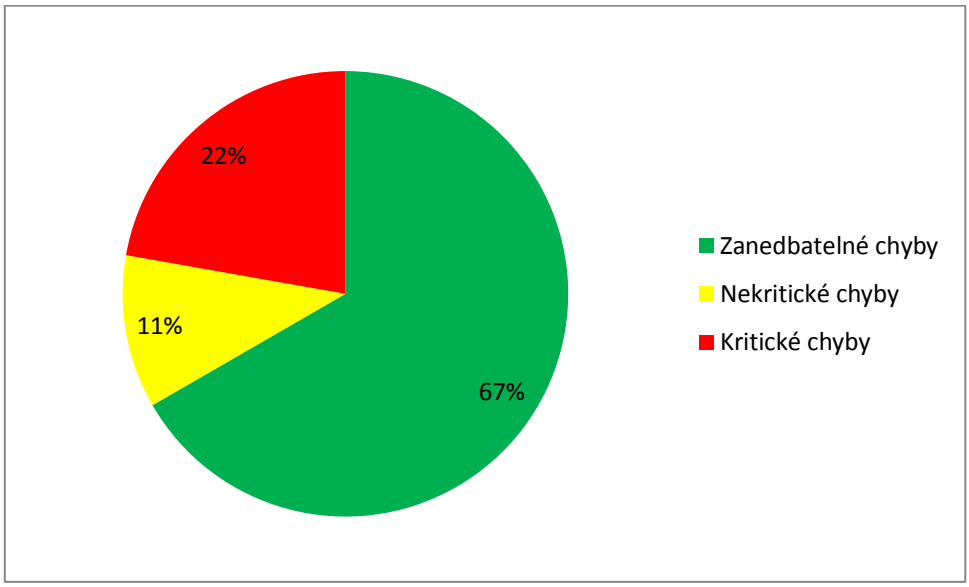
Graf 1:Časová náročnost testů



4.2. Klasifikace nalezených chyb

Chyby lze klasifikovat dle jejich závažnosti, a na základě těchto dat se může vývojář rozhodnout, zda je chyba natolik kritická, aby byla opravena, nebo ji lze v software ponechat. V grafu č. 2 je zobrazen podíl různě klasifikovaných nalezených chyb, kdy kritické chyby vyjadřují z celkového počtu chyb procentní podíl 22%. Tento podíl chyb je daleko menší než u obslužného programu pro Windows.

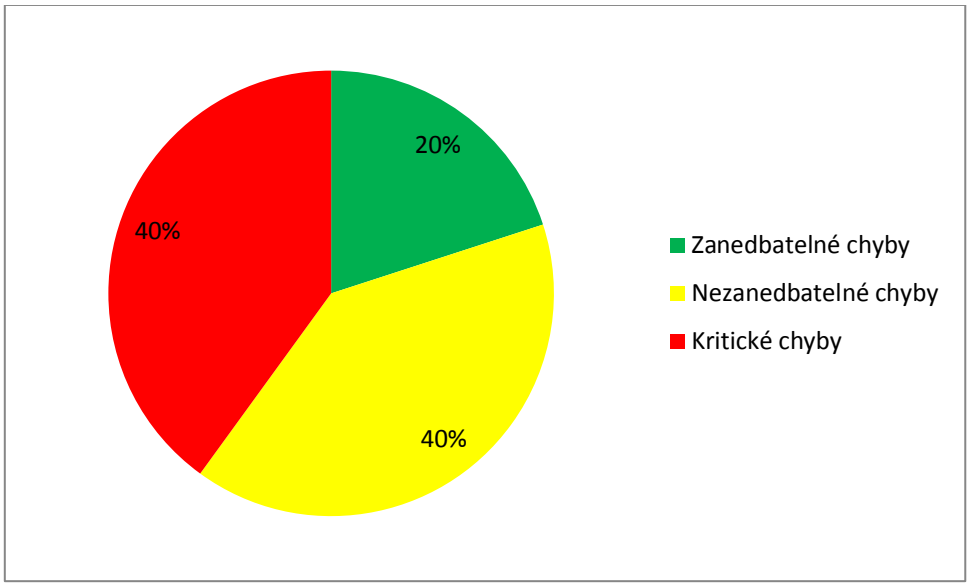
Graf 2:Přehled chyb (váhový počítač)



Chyby v obslužném programu

Z grafu č. 3 vyplývá, že procentní podíl kritických chyb je stejný jako u nezanedbatelných chyb. Obslužný program tedy obsahuje větší podíl kritických chyb, než tomu bylo u váhového počítače. Tyto data potvrzují, že na vývoj firmware pro embedded zařízení jsou kladeny větší nároky na spolehlivost než u programů na PC.

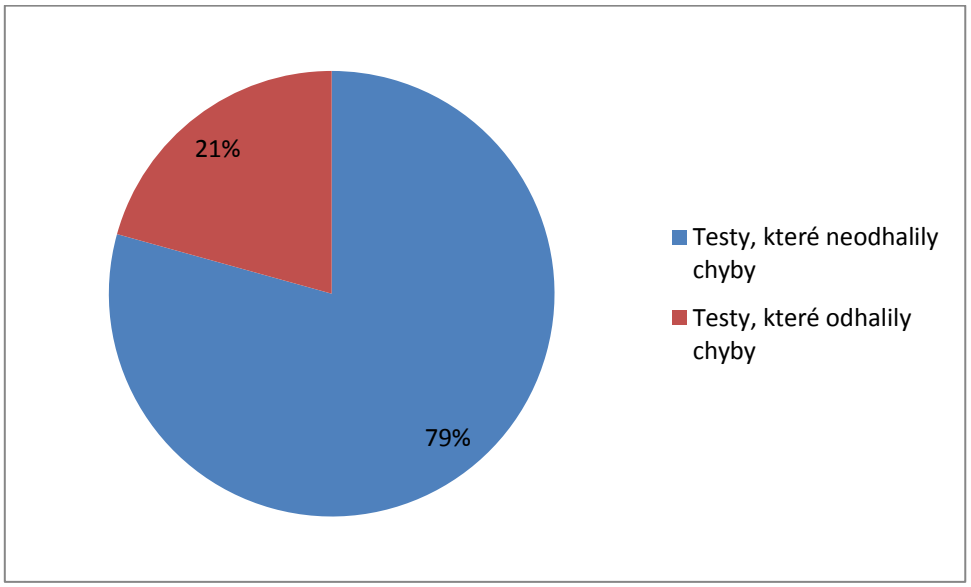
Graf 3:Přehled chyb (obslužný program)



4.3. Podíl testů, které odhalily chybu a testů bez úspěchu

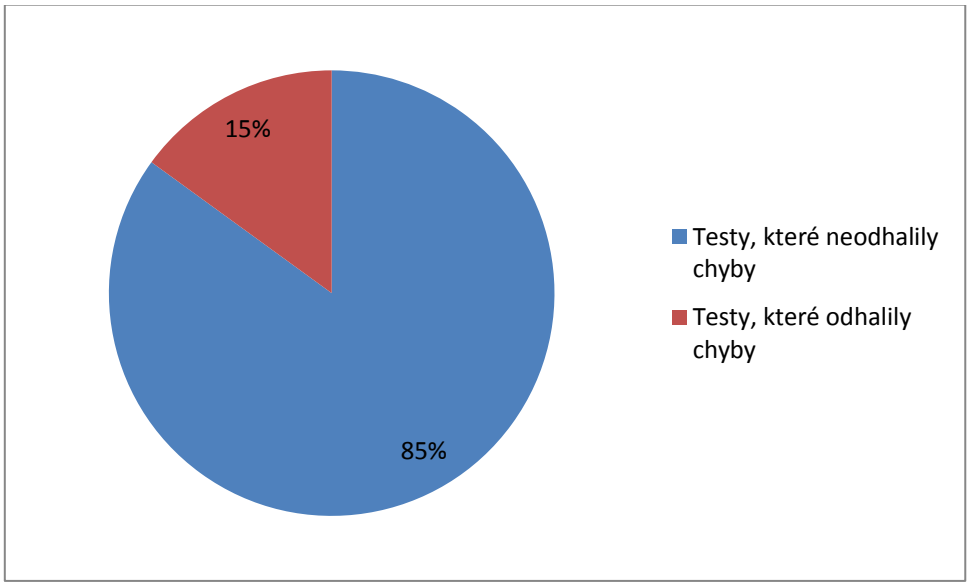
V grafu č. 4 je zobrazen procentní poměr mezi celkovým počtem provedených hraničních testů a počtem testů, které odhalily chyby.

Graf 4:Testy hraničních podmínek



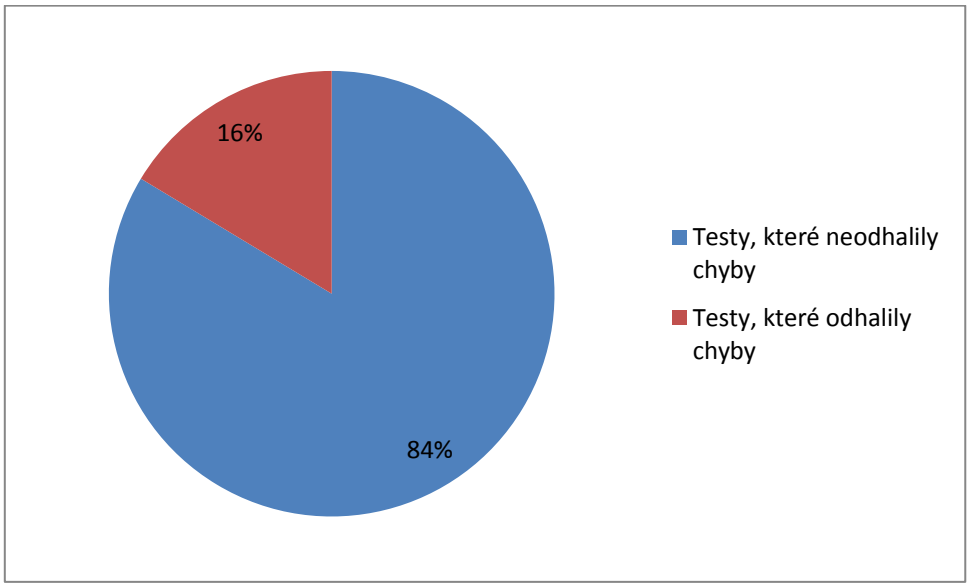
Jak vyplývá z grafu č. 5, procentní podíl testů subhraničních podmínek, které odhalily chyby je 15% z celkového počtu testů na subhraniční podmínky.

Graf 5:Testy subhraničních podmínek



Graf č. 6 zobrazuje na menší kruhové výseči procentní podíl všech testů, které odhalily chyby o hodnotě 16%. U zbylých 84% testů nebyly žádné chyby nalezeny.

Graf 6: Celkový procentní podíl úspěšnosti testů



Ze všech tří uvedených grafů plyne, že všechny použité druhy testů mají podobnou úspěšnost. Proto je nezbytné při testování používat co možná nejvíce různých testů, které jsou určené pro odlišné oblasti. Vhodným kombinováním testů lze docílit vyšší úspěšnosti při odhalování chyb.

5. Závěr

Vysoký podíl na spolehlivosti vestavného zařízení nenese pouze samotná „fyzická“ konstrukce, ale i softwarová výbava, bez které není zařízení schopné pracovat. Proto je velice důležité dbát i na kvalitu provedení firmware, a pokud možno nalezené chyby opravit a co nejdříve zákazníkovi poskytnout novou verzi. V praxi jsou aktualizace firmware velkým problémem, a tak se zákazník musí ve velkém množství případů naučit chybu ignorovat nebo si koupit nové zařízení. Tento nedostatek už začíná řešit nový trend, kdy výrobci přidají do zařízení síťové rozhraní a nejnovější aktualizace firmware se mohou automaticky stáhnout a nainstalovat. Implementací síťových rozhraní ale vzniká bohužel bezpečnostní riziko vniknutí útočníka do samotného zařízení. Typickým příkladem jsou routery, které se stávají cílem útoků každý den.

Při testu firmware váhového počítače byla použita metoda Black Box, která má široké uplatnění napříč různými platformami, jelikož nevyžaduje znalost vnitřní struktury software a programovacího jazyka. Navíc dokáže v poměrně krátkém čase otestovat i rozsáhlejší systémy. Naopak nevýhodou je nemožnost ověření příčiny nalezené chyby ve zdrojovém kódu a neznalost vnitřní struktury software. Situací, kdy se využívá tato metoda, může být celá řada - typickým příkladem je, kdy firma nechce umožnit přístup ke zdrojovému kódu software z obavy před zkopírováním. Ať už se použije jakákoliv metoda nebo postup, nikdy není stoprocentní, že je software bez chyb. Vhodným zkombinováním metod a testů se snižuje riziko chyb.

Z výsledků testů bylo zjištěno několik různých druhů chyb, ale žádná z nich nevedla k nestabilitě systému nebo jeho poškození. Úspěšnost testovacích souborů v rozdílných oblastech se pohybovala kolem 16 %. Výjimkou byl test zaplnění paměti, který skončil bez chyby. Několik chyb bylo nalezeno i v obslužném programu „Precizní krmení“, které byly většinou způsobeny neošetřením výjimek.

Tato práce může posloužit jako základní přehled testů a testovacích metod, které se provádějí. Zvláště se pak zabývá testovací metodou Black box. Zároveň je zde ukázáno praktické využití na konkrétním přístroji.

6. Seznam použitých zdrojů

Banzi, Massimo. 2011. *Getting Started with Arduino.* Sebastopol : O'Reilly Media, 2011. ISBN 1449309879.

Bednář, Radim. 2002. Automa. *Automa.* [Online] FCC Public s.r.o., 2002. [Citace: 15. 3 2014.] http://www.odbornecasopisy.cz/index.php?id_document=28632.

Brtník, Bohumil. 2011. *Mikroporcesorová technika.* Praha : BEN, 2011. ISBN 9788073004064.

Čermák, Miroslav. 2008. Clever and Smart. *cleverandsmart.* [Online] 23. 12 2008. [Citace: 15. 3 2014.] http://www.cleverandsmart.cz/.

Hlava, Tomáš. 2011. testovanisoftware.cz. [Online] 28. 8 2011. [Citace: 20. 3 2014.] http://testovanisoftwaru.cz/tag/black-box/.

Hlavatý, Ing. Tomáš. 2003. Automa. *Automa.* [Online] FCC Public s. r. o., 2003. [Citace: 3. 26 2014.] http://www.odbornecasopisy.cz/index.php?id_document=28707.

Javůrek, Karel. 2011. Zive.cz. [Online] 3. 10 2011. [Citace: 14. 3 2014.] http://www.zive.cz/clanky/intel-4004-nahodny-procesor-ktery-zmenil-svet/sc-3-a-158983/default.aspx.

Krumnikl, Michal. 2006. arg.vsb.cz. [Online] 19. 12 2006. [Citace: 20. 3 2014.] arg.vsb.cz/msita/downloads/embedded.ppt.

Michálek, Lukáš. 2006. *Bakalářská práce: Použití metod testování softwaru v praxi.* Praha : Vysoká škola ekonomická v Praze, 2006.

Mrštíková, Hana. 2005. fi.muni.cz. [Online] 2005. [Citace: 23. 3 2014.] http://www.fi.muni.cz/usr/jkucera/pv109/2005/xmrstik.htm.

Muknšnábl, Ing. Josef. 2001. *reboot.cz.* [Online] 25. 1 2001. http://reboot.cz/howto/programovani/testovani-programu/articles.html?id=94.

Patton, Ron. 2002. *Testování softwaru.* Praha : Computer Press, 2002. ISBN 8072266365.

Pavlíček, Michal. 2012. Moblinet.cz. *Mobilnet.* [Online] 21. únor 2012. [Citace: 15. březen 2014.] http://mobilenet.cz/clanky/multitasking-co-to-je-a-jak-funguje-na-jednotlivych-systemech-8602.

7. Seznam obrázků

8. Seznam grafů a tabulek