ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

Diplomová práce

Vytvoření specifikace pro testování řídicí jednotky

autor: **Bc. Lukáš Dědeček**
vedoucí práce: **Ing. Michal Kubík, Ph.D.**
konzultant práce: **Ing. Tomáš Síč**

2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš DĚDEČEK**
Osobní číslo:	**E10N0154P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Dopravní elektroinženýrství a autoelektronika**
Název tématu:	**Vytvoření specifikace pro testování řídicí jednotky**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Definice a zpracováni obecného konceptu pro testování zdrojového modulárního C kódu.
2. Vytvoření koncepční metody klasifikačního stromu za pomocí použití grafického rozhraní.
3. Ověření strategie pomocí testování na reálné platformě.

Rozsah grafických prací:	**dle doporučení vedoucího**
Rozsah pracovní zprávy:	**dle doporučení vedoucího**
Forma zpracování diplomové práce:	**tištěná/elektronická**
Seznam odborné literatury:	

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce a konzultanta.

Vedoucí diplomové práce:	**Ing. Michal Kubík** Katedra aplikované elektroniky a telekomunikací
Konzultant diplomové práce:	**Ing. Tomáš Síč** MBtech Bohemia s.r.o.
Datum zadání diplomové práce:	**17. října 2011**
Termín odevzdání diplomové práce:	**11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Západočeská univerzita v Plzni * Fakulta elektrotechnická *

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Vytvoření specifikace pro testování řídicí jednotky

Anotace

Diplomová práce se zabývá problematikou testování a bezpečnosti softwarových aplikací. Testování probíhá pomocí programu Tessy 2.9 společnosti Razorcat. Zdrojové kódy jsou psané v jazyce C, jejich struktura odpovídá používaným kódům, které se implementují do řídicích jednotek. Největší část práce se zaměřuje na modulové funkční testování – Komponent test, které je v tomto odvětví stěžejní. Vyhodnocení testů probíhá podle specifikace testování softwaru modulu, kde jsou uvedeny veškeré potřebné informace pro řádný průběh testování. Pro ověření funkčního testu je použit scenario editor, který je plně integrován v Tessy 2.9. Diplomová práce je psána ve spolupráci se společností MBtech Bohemia s.r.o., která dodala veškeré potřebné softwarové a hardwarové vybavení.

Klíčová slova:

Sémantická chyba, Syntaktická chyba, SW Tessy, Dynamické testování, Statické testování, Pokrytí kódu, Funkční testování, ASIL, SIL, Vývoj softwaru, V-model, Unit testování, SW Understand, SVN server, Simulátor Trace32

Testing specification of the control unit

Abstract

This master thesis deals with the testing and safety software applications. The testing procedure is performed using Tessy 2.9 made by company Razorcat. The source codes are written in C language, their structure corresponds to used codes that are implemented in control units. The largest part of the work is focused on module´s functional testing – Component test, which is fundamental in this sector. Evaluation of tests proceeds according to specification of software testing module, where is stated all necessary information for smooth testing. A scenario editor is applied for verification of functional test, that is fully integrated into Tessy 2.9. This thesis was written in cooperation with the company MBtech Bohemia Ltd, which supplied all the necessary software and hardware.

Keywords:

Semantic error, Syntax error, SW Tessy, Dynamic testing, Static testing, Code coverage, Functional testing, ASIL, SIL, Software development, V-model, Unit testing, SW Understand, SVN server, Simulator Trace32

Prohlášení:

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.
Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.
Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce je legální.

V Plzni dne 8.5.2012

Jméno příjmení

……..

Poděkování

Na tomto místě bych rád poděkoval Ing. Marku Bogovi, Ph.D. za cenné profesionální rady, připomínky, metodické vedení práce a úpravu zdrojových kódů pro testování. Dále bych chtěl poděkovat konzultantovi diplomové práce Tomáši Síčovi za úpravu zdrojových kódů a profesionální rady při zpracování práce. Ing. Michalu Kubíkovi, Ph.D. bych rád poděkoval za formální korekturu práce. V neposlední řadě bych poděkoval rodině a přítelkyni za psychickou a morální podporu po celou dobu psaní diplomové práce.

Obsah

Seznam použitých symbolů, zkratek a anglických termínů:

ASIL	Automotive Safety Integrity Level – stupeň integrity bezpečnosti v automobilovém průmyslu
BDM	Background Debug Mode – rozhraní společnosti Motorala
CTE	Classification Tree Editor – klasifikační stromový editor funkcí
FBD	Function Block Diagram – funkční blokové schéma
FPL	Fixed Program Language – pevný programovací jazyk
FVL	Full Variability Language – jazyk s plnou variabilitou
HAZOP	Hazard and Operability study – studie rizik a provozuschopnosti
HW	Hardware – technické vybavení
IL	Instruction List – seznam instrukcí
IT	Information Technology – informační technologie
JTAG	Joint Test Action Group – rozhraní standardu IEEE 1149.X
LD	Ladder Diagram – kontaktní schéma
LVL	Limited Variability Language – jazyk s omezenou variabilitou
PLC	Programmable Logic Controller – programovatelný automat
SCE	Scenario Editor – editor scénáře
SFC	Sequential Function Chart – sekvenční funkční schéma
SIL	Safety Integrity Level – stupeň integrity bezpečnosti
ST	Structured Text – strukturovaný text
SVN	Subversion – nástroj pro správu verzí souborů
SW	Software – programové vybavení
TCL	Tool Confidence Level – úroveň důvěry zařízení
TD	Tool Error Detection – detekce chyb
TDE	Test Data Editor – editor testovaných hodnot
TEE	Tessy Environment Editor – editor prostředí Tessy
TI	Tool Confidence – dopad nástroje na bezpečnostní požadavek
TIE	Test Interface Editor – editor testovaného rozhraní

1 Úvod

Testování softwarových aplikací je v dnešní době stejně důležité jako jejich samotné psaní. Pouze člověk neznalý oboru by mohl namítat, že to je zbytečné plýtvání financí a času. Opravdový programátor chybu neudělá. To je omyl – každý může udělat chybu, a proto se toto odvětví stále více dostává do popředí zájmu společností vyvíjecí SW. Moderní architektury vývoje SW již automaticky počítají s testovacími fázemi, které jsou vhodně začleněny do celého komplexu vytváření programu. Velký vliv na tyto změny má ekonomické prostředí, které zasahuje do všech oblastí průmyslu. Odstranění chyb již v počátku vývoje ušetří obrovské finanční prostředky, než odhalení chybové funkce softwaru po nasazení výrobku na trh. Jakákoliv společnost, která chce být ve svém oboru na nejvyšších místech, si nemůže dovolit žádnou chybu. Ztratila by své zákazníky a pracně vybudovanou pověst.

Téma testování softwaru mě natolik zaujalo, že jsem se rozhodl proniknout do tohoto rychle se rozvíjejícího oboru více. Je to dáno i mojí zkušeností s psaním zdrojových kódů, ale téměř žádnými zkušenostmi s jejich testováním. Myslím, že mi tato diplomová práce dá cenné zkušenosti, které později využiji ve své praxi, ať již jako programátor, který bude vědět, co se s jeho napsaným kódem děje v další fázi projektu, nebo jako softwarový tester, který provádí samotné testování aplikací.

Diplomová práce se snaží uvést čtenáře do problematiky testování softwarových aplikací psaných v programovacím jazyce C, který je jedním z nejpoužívanějších nástrojů programování. V automobilovém průmyslu a v dopravním odvětví obecně se používají modulární C kódy, které se implementují přímo do řídicích jednotek dopravních prostředků. Modularitou je dána přímo i struktura kódu, který je vázán na určitý hardware a typ použitého procesoru. Struktuře modulárního kódu musí být upraveno i testování, a proto se používají speciální aplikace, které jsou přímo navrženy pro tento typ testů.

Jedním s používaných nástrojů je program Tessy od společnosti Razorcat, který má v praxi široké zastoupení ve společnostech zabývajících se testováním softwaru. Mezi největší výhody tohoto softwaru patří zobrazení pokrytí kódu, využití pro regresní testování a standardizované dokumenty pro zobrazení výsledků testů.

Práce je členěna do tematických kapitol, kde postupně přecházím od teoretických poznatků k praktickému využití SW Tessy. V druhé kapitole uvádím čtenáře do problematiky vývoje a testování SW, jelikož je důležité vymezit základní pojmy, které se v tomto oboru vyskytují. Vysvětlím nejpoužívanější modely vývoje SW, mezi které patří vodopádový model a V-model. Další významnou částí kapitoly je problematika funkční bezpečnosti a plnění legislativních norem, kde se nejvíce zaměřím na normu ISO 26262, která definuje pojem ASIL. Hlavní obsah práce se věnuje problematice funkčního testování ve scenario editoru (SCE) pomocí softwarové aplikace Tessy 2.9 a následné využití pro konkrétní případy. Vysvětlím postup od vytvoření databáze projektu, až po spuštění a vyhodnocení testu. Testy budu vytvářet na základě dodané specifikace testování SW pro každý testovaný modul, kde využiji tabulky parametrů pro jednotlivé proměnné. Mezi další SW produkty, které využiji, patří manager zdrojového kódu Understand a simulátor procesorů řady C166 TRACE32 od společnosti Lauterbach Development tools. Popis a funkce zmíněných produktů popisuji v kapitolách tři a čtyři, simulátor TRACE32 až v kapitole pět, kde je uveden i přímý postup nastavení procesoru. V páté kapitole realizuji testování dodaných modulů na základě vypracovaného postupu v předešlých částech práce. Věnuji se zde i přípravným fázím, které předchází testům, popisu funkce SW SVN, vyhodnocení testů a vytvoření dokumentace z celého testovacího procesu.

Diplomová práce je zpracovávána pro společnost MBtech Bohemia s.r.o. Společnost se specializuje na unit testování a s tím související pokrytí zdrojového kódu. Zákazníci společnosti ale začínají požadovat i funkční testování a během příštích několika málo let bude komponent test běžným standardem. Bylo by přínosné předběhnout konkurenci a tento typ testů nabízet již nyní. Nejenom, že by to vedlo k zvýšení prestiže a obratu společnosti, ale přispělo by to i k bezpečnějším aplikacím, což je v dopravním odvětví bráno jako nejdůležitější vlastnost výrobku. Společnost MBtech Bohemia s.r.o. má s testováním softwaru bohaté zkušenosti, a proto jsem rád, že mohu spolupracovat s odborníky z praxe a získané teoretické poznatky si ověřit v běžném provozu.

2 Testování Softwaru

Teoretický základ je velmi důležitý pro správnou orientaci v dané problematice, a proto je důležité vymezit základní pojmy, se kterými se můžete setkat, ať v této diplomové práci, nebo i později v praxi. V této kapitole osvětlím pojmy, jako jsou chyba softwaru, vznik chyb a funkční testování. Následně přejdu k pojmům SIL, ASIL a problematice integrity bezpečnosti. Pro pochopení správného začlenění testování je důležité si ujasnit proces vývoje SW, jemuž se také budu věnovat v této kapitole.

2.1 Chyba softwaru[1]

Definice chyby softwaru se může na první pohled zdát docela jednoduchá, ale jak ukáži na následujících řádcích, není tomu tak. Pokud bych chtěl nazývat chybou jakýkoliv problém, který se může vyskytnout v softwarové aplikaci, musel bych definovat další pojem. Další definice by tedy byla, co je to problém programu. Dospěl bych tím k tomu, že bych pouze definoval nové pojmy a skutečnou podstatu věci bych se nedozvěděl. Nadefinuji proto pouze jeden pojem a tím je *specifikace produktu*. Co si pod tímto pojmem představit? Specifikace produktu je podrobný popis jeho funkcí, jak se má produkt chovat a jaké chování už je bráno za nestandardní (produkt neplní přesně funkci, pro kterou byl vytvořen). Jelikož software je v dnešní době brán jako produkt, platí pro něj výše zmíněná definice. Definování chyby SW by se tak dalo shrnout do několika pravidel:

1) Softwarová aplikace se chová tak, že její výsledný stav, není zahrnut ve specifikaci.
2) Softwarová aplikace se chová tak, jak by se podle specifikace chovat neměla.
3) Softwarová aplikace se nechová podle uvedení ve specifikaci.
4) Softwarová aplikace se nechová podle specifikace, ale mělo by to být ve specifikaci uvedeno.
5) Software je podle názoru testera nesrozumitelný, pro koncového uživatele nepoužitelný.

[1] Čerpáno ze dvou zdrojů:
STEPHENS, Matt a Doug ROSENBERG. *Testování softwaru řízené návrhem*. Vyd. 1. Brno: Computer Press, 2011, 336 s. ISBN 978-80-251-3607-2.

PATTON, Ron. *Testování softwaru*. Vyd. 1. Praha: Computer Press, 2002, 313 s. Programování. ISBN 80-722-6636-5.

Pro lepší pochopení výše uvedených pravidel uvedu několik příkladů. Nejprve je třeba vymezit specifikaci produktu. Je vytvořen program, který má ve specifikaci uvedeno, že se nepřetíží a umí počítat obvod kruhu. Pokud se podaří pomocí programu spočítat i obsah kruhu, je to už chyba podle pravidla 1 (tato funkce mohla být do aplikace přidána, ale pokud to není uvedeno ve specifikaci, musíme to brát jako chybné chování programu). Při zadání velkého čísla poloměru aplikace přestane fungovat, porušil jsem pravidlo 2 (přetížení aplikace). Pokud při správném zadání poloměru dostanu chybný výsledek, nastane chyba podle pravidla 3. Čtvrté pravidlo uvádí chyby, na které se ve specifikaci zapomnělo, například funkčnost aplikace při vytíženém procesoru počítače (chybný výsledek, dlouhá odezva výpočtu). Poslední pravidlo je spíše už otázkou pro vzhled aplikace a její snadné použití. Zde je vhodné uvést pojem *uživatelsky příjemná funkce*. Vyřešit tento poslední bod bývá nejtěžší úkol pro softwarového testera, protože vytvořit dokonalý program pro všechny uživatele je téměř nemožný úkol.

2.2 Nejvýznamnější zdroje chyb[2]

Zvídavý čtenář si musí v tento moment klást zásadní otázku. Kde vzniká nejvíce chyb? Většina laické společnosti by mohla na tuto otázku odpovědět špatně a říci, že nejvíce chyb vzniká ve zdrojovém kódu. Není tomu tak. Nejvíce chyb se vytváří ve specifikaci a následném návrhu aplikace. Jak je to možné? V mnoha případech specifikace chybí, a proto nejsme schopni identifikovat správné chování aplikace. Když už se specifikace uvede, bývá často neúplná, nebo dochází k častým změnám a tím se vytváří chyby v softwarové aplikaci. Pro chyby vytvořené při návrhu platí podobné věci, bývá často nedostatečný, nerozmyšlený, uspěchaný.

Dalším významným zdrojem chyb je již zmiňovaná chybovost ve zdrojovém kódu, ať již to jsou chyby *syntaktické*, které se vytvoří různými překlepy při psaní kódu, nebo opomenutím důležitého znaku (př. v jazyce C nenapsání středníku za příkazem). Tyto chyby odhalí už překladač a jsou velmi rychle opraveny.

Sémantická chyba neboli chyba významová je přestupek proti požadovanému chování kódu a způsobuje jeho chybnou funkci. Tyto chyby jsou mnohem hůře dohledatelné, protože psaný kód projde bez problémů překladem. Pouze autor kódu je schopen odhalit, že softwarová aplikace neplní požadovanou funkci (př. aplikace

[2] Čerpáno z jednoho zdroje:
HARPER, Allen, Shon HARRIS, Chris EAGLE, Jonathan NESS a Michael LESTER. *Hacking*: *manuál hackera*. 1. vyd. Praha: Grada, 2008, 399 s. ISBN 978-80-247-1346-5.

na výpočet obvodu kruhu při správném zadání poloměru vypočte chybný výsledek). Zde musí programátor prozkoumat kód pečlivěji a uvědomit si chování požadované funkce. Chyba může nastat například při špatném znaménku a neošetření formátu zadávaných proměnných, případně chybná deklarace a inicializace proměnných, přepisu parametru v průběhu vykonávání funkce aj. V některých případech to může vést k nestabilitě aplikace a jejímu následnému „ zamrznutí“.

U některých vyšších programovacích jazyků se může vyskytnout ještě takzvaná *logická* chyba, která se projeví pouze za specifických okolností. Příkladem mějme zdrojový kód v jazyku PHP, kde pro sloučení řetězců je nutné použít znak (.). Pokud programátor použije znak (+) – syntakticky i sémanticky je vše v pořádku, jenže překladač si nyní myslí, že pracuje s čísly. Provede přetypování na čísla a výsledek je 0 namísto vypsání řetězce. Tato chyba se může projevit při následném použití výsledného řetězce jako logické proměnné.

2.3 Vývoj softwarové aplikace[3]

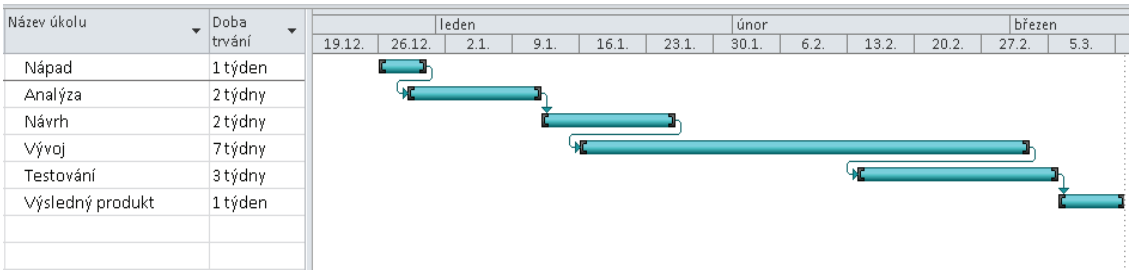
V této kapitole uvedu postupy při vývoji softwaru, stručně popíši jednotlivé procesy a ukáži, jak testování softwaru zapadá do konceptu vývoje. Představím jednotlivé modely při návrhu aplikací a jejich výhody a nevýhody.

Než začnu s popisem fází při vývoji SW, chtěl bych zmínit jednu důležitou věc, která je se samotným návrhem provázána. Touto věcí jsou tzv. náklady na chyby. Ekonomická stránka věci je zde namístě, jelikož je software brán jako produkt generující zisk a chyby tento zisk snižují, ať to je jejich samotné odstraňování, nebo hledání. Platí zde pravidlo, že čím dříve se chyba v procesu vývoje najde, tím méně musíme vynaložit finančních prostředků na její odstranění. Jak se pokračuje s procesem vývoje, tak se cena na odstranění chyb zvyšuje 10ti násobně. Jestliže by objevená chyba při návrhu stála 1 Kč, stejná chyba objevená až při sestavování kódu by stála 10 Kč a tato částka by byla nejvyšší, kdyby tu samou chybu objevil až zákazník (tisíce Kč). Pokud by tato chyba způsobila zákazníkovi škody, nebo by po dobu odstranění chyb nemohl pokračovat v práci, byla by tato částka několikanásobně vyšší. Zde již je vidět důležitost a oprávněnost testování.

Nyní přistoupím k samotnému procesu vývoje softwaru. Většina softwaru se vytváří přímo pro zákazníka, a proto první zjišťovanou věcí jsou požadavky zákazníka,

[3] Čerpáno z jednoho zdroje:
EELES, Peter a Peter CRIPPS. *Architektura softwaru*. Vyd. 1. Brno: Computer Press, 2011, 328 s. ISBN 978-80-251-3036-0.

jeho představy, co by produkt měl umět, aby byl uživatelsky příjemný, přehledný (důležité je vědět, kdo s produktem bude pracovat, jestli to budou odborníci, nebo laici a je nutné k tomu i produkt přizpůsobit – toto lze ošetřit i různými verzemi softwaru). Na základě těchto informací se vytvoří specifikace produktu, kde se podrobně popíší veškeré funkce, chování produktu za standardních i nestandardních podmínek. Tento krok je velmi svazující pro vývojovou firmu, jelikož nesplnění některého bodu ze specifikace je bráno jako chyba produktu a zákazník se může domáhat odškodnění (finanční, materiálové). Neméně důležitým krokem jakým je vytvoření specifikace SW, je určení časového plánu. Časový plán je rozvrh, kdy se budou jednotlivé činnosti provádět. Tento rozvrh slouží pro orientaci, v jaké fázi se vývoj nachází a k určení data uvedení produktu na trh.



Obr. 1 Ganttův diagram

Velkým pomocníkem při vytváření časového harmonogramu jsou různé grafické aplikace. Jedním z příkladů může být Ganttův diagram vytvořený pomocí SW MS Office. Příklad využití Ganttova diagramu je vidět na Obr. 1. Je pravda, že se plán může při vývoji i několikrát změnit. Je to dáno neodhadnutím složitosti aplikace, odhalováním velkého množství chyb a jejich následné opravy, nebo i takové faktory, jako je pracovní morálka vývojářů, nemoc, případně práce na několika projektech najednou. Další struktura vývoje aplikace se již týká psaní zdrojového kódu. Zde je nutná dokumentace pro návrh softwaru, mezi níž patří *architektura* – celková struktura softwaru, vzájemná komunikace částí aplikace, *diagram toku dat* – popisuje pohyb dat v aplikaci (bublinový diagram), *diagram stavů a přechodů* – definuje způsob přechodu softwaru mezi stavy, *diagram toku řízení* – popisuje logiku řízení aplikace, *komentáře u zdrojového kódu* – lepší čitelnost a pochopení kódu osobou, která vidí kód prvně (nebo autor, který vidí svůj kód po několika letech) a v neposlední řadě dokumentace testů, která zahrnuje *plán testování*, *seznam testovaných případů*, *záznamy o chybách*, *statistiky o postupech*

testování (grafy, diagramy). V následující kapitole ukáži, jak jdou jednotlivé části vývoje chronologicky za sebou, pro postup vývoje SW se vžil pojem *model*.

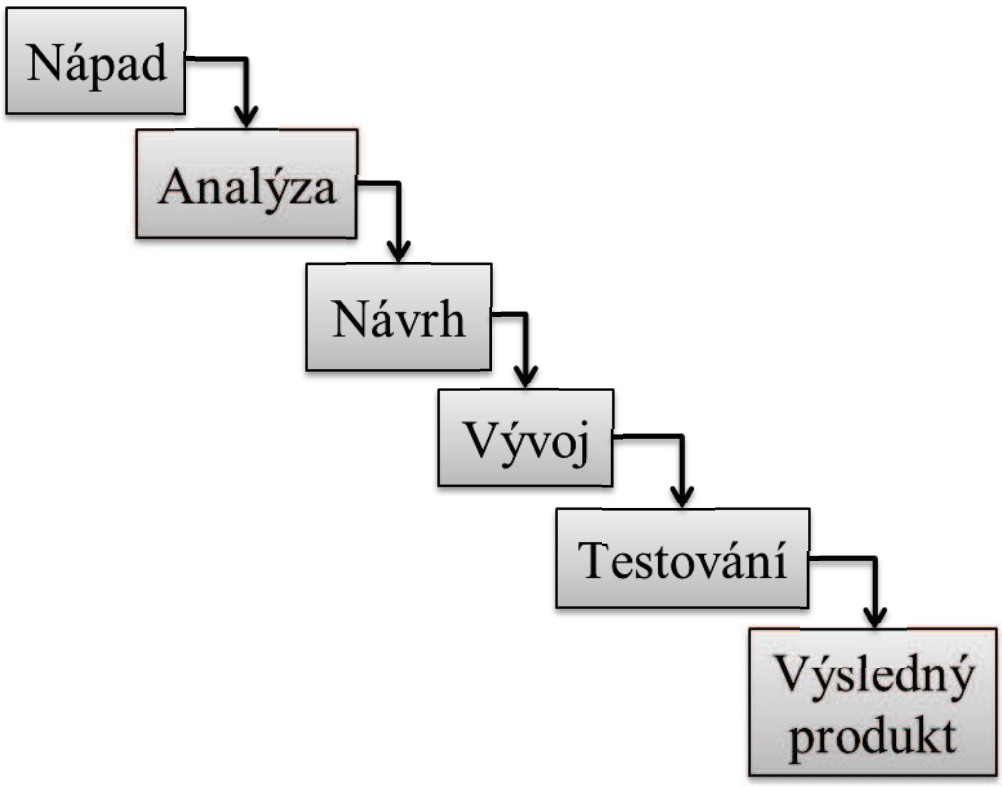
2.4 Modely vývoje softwaru[4]

Modely vývoje softwaru jsou rozděleny do dvou skupin. První skupina by se dala popsat jako chaotický vývoj, bez nějakých více promyšlených posloupností organizace práce. Nepřekvapí ani jejich pojmenování, které naznačuje tuto vlastnost. Do skupiny patří model „*velkého třesku*". Dal by se charakterizovat tím, že někdo přijde s revolučním nápadem, společnost investuje obrovské prostředky na jeho realizaci a vyjde z toho dokonalý produkt (v tom lepším případě). Tato metoda je charakteristická minimálním plánováním a téměř žádným rozvrhem práce. Jediné zaměření je na samotný výsledek produktu, aby plnil funkci, která byla oním revolučním nápadem, proč se začalo s vývojem. Softwarové aplikace vyvíjené touto metodou bývají velice inovativní. Často obsahují velký počet chyb, jelikož je u nich velmi obtížné systematické testování, nebo se netestují vůbec. Uživatelé by měli být na tento fakt upozorněni a dávat sami podněty k opravě kódu. Tím se dostávám k cílové skupině uživatelů používajících software tvořený touto metodou. Nikoho nepřekvapí, že skupinu budou tvořit odborníci v oblasti IT. Ti jsou ochotni akceptovat chyby, ba dokonce chyby hledat, za cenu velmi revolučního a inovativního produktu, který jim může značně zjednodušit práci.

Druhým typem modelu patřící do této skupiny je model „*programuj a opravuj*". Jak název napovídá, opět se nejedná o systematický postup při vývoji softwarové aplikace. Tento model je podobný předešlému s tím rozdílem, že zde již existuje hrubý návrh specifikace produktu. Po určení požadavků na funkci produktu se opakuje smyčka programování a opravování vzniklých chyb. Až se vývojový tým shodne, že je aplikace dostatečně připravena na uvedení na trh (může stále obsahovat drobné chyby) vydají ji i s vědomím těchto chyb. Takovéto aplikace jsou zanedlouho aktualizovány novou verzí a tento běh pokračuje, dokud se nevydá verze, která radikálněji mění funkci produktu. Jak možná tušíte, ani tento model nemá zahrnut ve svém vývoji důkladnější testování. Určí se datum vydání produktu a s blížícím se termínem, se snaží softwarový tester odhalit co nejvíce chyb u nejaktuálnější verze programu.

[4] Čerpáno z jednoho zdroje:
EELES, Peter a Peter CRIPPS. *Architektura softwaru*. Vyd. 1. Brno: Computer Press, 2011, 328 s. ISBN 978-80-251-3036-0.

Nyní se dostávám k druhé skupině modelů vývoje softwaru, které jsou charakteristické promyšlenými postupy, striktně daným časovým harmonogramem a správným plněním všech povinností. Nejznámějším modelem v této skupině je model „*vodopádu*“, který je názorně zobrazen na Obr. 2.

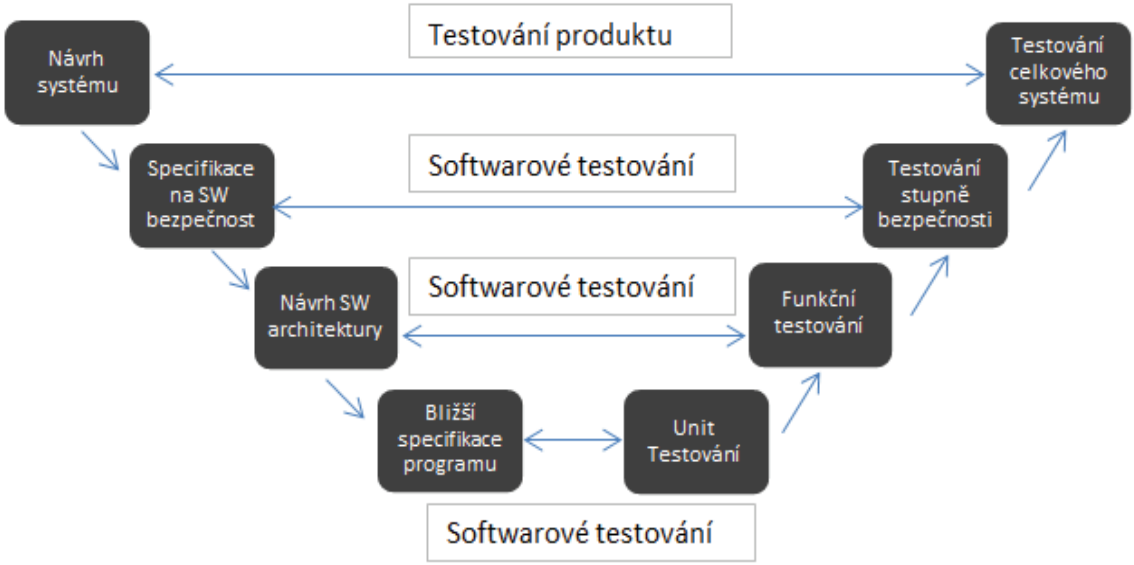


Obr. 2 Model vodopádu vývoje SW

Tento model je jedním z nejpoužívanějších vývojových schémat SW v běžné praxi, ale jak dále ukáži má i své velké nevýhody. Hlavní předností modelu je jeho jednoduchost a přesné popsání specifikace jednotlivých kroků vývoje produktu. V každé fázi vývoje je přesně dáno, jak by se mělo postupovat, kdy by měla být určitá fáze dokončena a tím je dán i termín vydání. Na konci každého kroku se hodnotí, jestli je splněn cíl pro právě dokončenou etapu vývoje a může se přejít do fáze následující. Po opuštění určité fáze se již nepřipouští návrat a je možno pracovat pouze v aktuální části vývoje. Z tohoto hlediska vyplívá asi největší úskalí modelu „*vodopádu*“. Jak jsem uvedl výše pravidlo nákladů na chyby, je vidět, že testování aplikace se nachází až na samotném konci modelu, pouze před vydáním produktu. Odhalení chyby až v tomto procesu vývoje chybu několikanásobně prodraží a může dokonce oddálit i samotné

vydání produktu, což může způsobit další finanční ztráty. Z tohoto hlediska je nemožnost vrátit se do předcházející fáze vývoje produktu při odhalení chyby dříve, než při konečném testování, značně nevýhodné a tím tuto metodu velice znehodnocuje. Existují určité typy modelu vodopádu, kde existuje možnost vrátit se o krok zpět a chybu napravit, což má ale za následek odsunutí data vydání (zde vítězí ekonomická stránka věci – vydat spolehlivý SW). Významným omezením je nepružná reakce na změny ve specifikaci při celém vývoji a psaní zdrojového kódu aplikace. V průběhu vývoje může zákazník měnit své požadavky. Ty mohou být zcela jiné, než při prvotním návrhu aplikace a tím výsledný produkt nebude přesně plnit funkci žádanou zákazníkem.

V současnosti je velmi často používaný model vývoje SW tzv. „*V – model*“. Je to rozšířený a upravený model „*vodopádu*“. Místo striktního lineárního postupu od shora dolů je model ohnut a vytváří tak typický tvar V, podle kterého je i tento model pojmenován. Blokové schéma modelu je zobrazeno na Obr. 3. V levé větvi jsou zobrazeny jednotlivé fáze vývoje SW, v pravé větvi jsou k těmto fázím přiřazeny fáze testování.



Obr. 3 V – model vývoje softwaru

V každém postupu ve struktuře o úroveň níž dochází k otestování určité části systému a zároveň probíhá verifikace daného kroku. Ve výše uvedeném modelu dochází k verifikaci celkem ve čtyřech fázích. Verifikace otestované části nám doloží, že daný subjekt vykazuje cílené chování a není třeba ho dále testovat. Důležité je, aby testování

probíhalo podle logického směru vývoje SW. Nejprve je nutné podrobit aplikaci unit testováním a až poté přejít k funkčnímu a chronologicky postupovat až k úplnému otestování a schválení produktu.

Poslední model, který v tomto přehledu uvedu je „*spirálový*“ model. Tento model je z hlediska použití nejefektivnější, využívá kladné vlastnosti modelů předešlých a zároveň je doplňuje o nové, které řeší chyby předchozích modelů. Základní myšlenka je velice prostá. Na začátku, kdy ještě nemáme úplnou představu o všech aspektech aplikace, nadefinujeme pouze několik málo důležitých funkcí, ty předáme dále k analýze uživatelům. Po vyhodnocení připomínek a jejich zapracování přecházíme na další úroveň vývoje.
Celý průchod jednou úrovní by se dal shrnout do šesti kroků:

1) Vymezení cílů, specifikace
2) Možná rizika při vývoji
3) Vyhodnocení jiných postupů
4) Psaní kódu a testování
5) Přípravy na další úroveň
6) Vyhodnocení postupu na další úroveň

Iterace jsou rozděleny na čtyři kvadranty – plánování, odhad rizik, vývoj a hodnocení. „*Spirálový*“ model je pro softwarového testera tím nejlepším vývojovým modelem. Dovoluje mu testovat produkt při každém průchodu úrovní, tím je zajištěno včasné odhalení chyby a snížení nákladů na její odstranění. Pravidelným testováním, je tester zapojen do týmu a vidí celý zrod aplikace. Při konečném testování ověří pouze správnost postupu.

2.5 Statická a dynamická typová kontrola softwaru

Statická typová kontrola zkoumá kód, který není spuštěn. Tester pouze prohlíží kód a snaží se v něm nalézt chybu pouze za pomoci překladače softwaru. Staticky lze zkoumat i dokumentaci a specifikaci, nepotřebujeme k tomu žádný funkční kód, pouze analyzujeme proveditelnost a časovou náročnost. Dynamická typová kontrola se provádí při spuštěném softwaru a je tedy nutná již spustitelná část zdrojového kódu. Nejčastěji se provádí zadáním vstupních hodnot a vyhodnocováním hodnot výstupních.

2.6 Testování černé a bílé skříňky

Při testování softwaru se objevují pojmy jako je *bílá (white box)* a *černá (black box*) skříňka. Testování černé skříňky je pouze analýza vstupů a výstupů bez znalosti vnitřního algoritmu. Takto software vidí uživatel a je to tedy testování uživatelské přívětivosti aplikace, její možné omezení funkčnosti při zadání špatných vstupních parametrů. Testování bílé skříňky je pro testera příjemnější, neboť se může podívat i na vnitřní strukturu kódu, použité algoritmy a lépe tak odhalovat chyby v aplikaci. Zde již SW aplikaci nevidí jako uživatel, což může být někdy nevýhodné z hlediska rozpoznání uživatelské chyby.

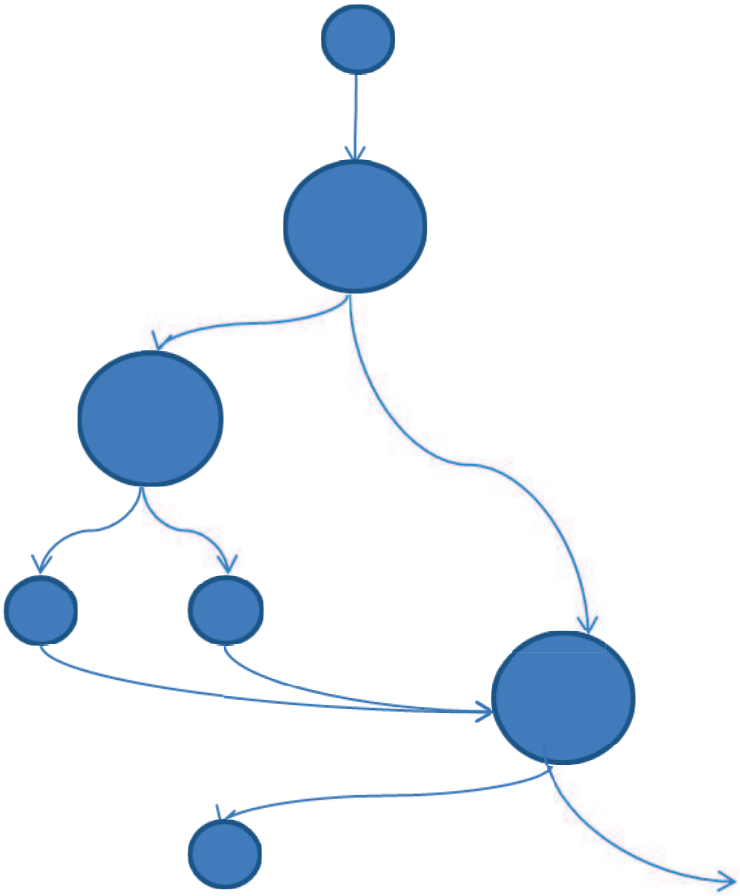
Některé zdroje uvádějí ještě třetí pohled na testovaný objekt. Používá se pojem *šedá (grey box)* skříňka. Tato šedá skříňka je kombinací dvou předchozích případů, kde se testují vstupy a výstupy s mírnou znalostí vnitřních algoritmů a struktury zdrojového kódu.

2.7 Pokrytí kódu[5]

Důležitým aspektem při testování softwaru je jeho pokrytí testy. Míra pokrytí se vyjadřuje procentuálně a dává nám jasný ukazatel, z jaké části je produkt otestován. Předmětem testování je také dokumentace k softwarové aplikaci i požadavky zákazníka. Pro testera je nejdůležitější známkou funkčnost programu, ale funkčnost ještě neznamená plné otestování a pokrytí aplikace. Mezi nejjednodušší metody patří *pokrytí řádků*. Test spočívá v kontrole řádek, tester označuje ty, které již byly zkontrolovány. Na řádcích se nachází přímo jednotlivé příkazy, objevuje se proto i termín *pokrytí příkazů*. Tímto testováním, mohu bez obtíží dosáhnout 100% pokrytí, ale to ještě neznamená, že je kód v pořádku. Hlavním úskalím tohoto typu testování je, že ke správnému vyhodnocení řádku stačí pouze provedení příkazu. Při vyhodnocení podmínky proto netestuji, jestli je podmínka vyhodnocena správně a její nesprávné vyhodnocení mi ovlivní další běh programu, ale stačí pouze libovolné vyhodnocení řádku. To může samozřejmě působit problémy, ale pro velkou jednoduchost a vysoké procento pokrytí se tato metoda hojně používá. Promyšlenější a pokročilejší metoda je založena na *pokrytí hran*, nebo se také uvádí termín *pokrytí rozhodnutí*. Metoda se nejlépe popisuje grafy, kde si příkaz představím jako uzel a přechod mezi jednotlivými

[5] Čerpáno ze dvou zdrojů:
RAZORCAT DEVELOPMENT GMBH 2009. *Tessy 2.9*: *User manual*. Berlín, 2009
Testování softwaru: Problémy s pokrytím kódu. In: *Testovanisoftwaru* [online]. 5.10. 2009 [cit. 2012-04-20]. Dostupné z: http://testovanisoftwaru.blogspot.com/

uzly (příkazy) tvoří hrany. Pokud se dostanu do stavu, že příkaz tvoří podmínka *if*, v grafu se mi to projeví tak, že z uzlu vedou dvě hrany (podmínka vyhodnocena jako true, nebo jako false) Jednoduchý příklad pokrytí hran znázorňuje Obr. 4, který zobrazuje kód, kde se nacházejí tři podmínky – velká kola a čtyři příkazy – malá kola. Takováto struktura je vhodná pro příkazy typu *if* (podmínka) – podmínka je splněna, dojde k postupu na další příkaz, nebo není splněna a dojde k vypsání chybové zprávy, nebo provedení příkazu při nesplnění podmínky.



Obr. 4 Graf pokrytí hran

Avšak ani toto pokrytí kódu není dokonalé. Při složitějších podmínkách je pokrytí hran nedostatečné. Na řadu přichází *pokrytí podmínek*, které se používá při složených podmínkách, kde by pouhé pokrytí hran nestačilo. Pokrytí podmínek musí splňovat pokrytí hran a navíc musí umět vyhodnotit všechny možné stavy podmínky, nejenom, jestli je splněná, nebo nesplněná. Při vynechání podmínky pokrytí hran bych se dostal opět pouze k vyhodnocení pravdivosti, nebo nepravdivosti podmínky. Pokrytí

podmínek se proto může jevit, jako vnořené pokrytí hran samo do sebe. Toto pokrytí už detekuje velký počet chyb, avšak to ještě není nejspolehlivější kontrola kódu. Nejpodrobnější testování kódu je provedeno *pokrytím cest*. Testování, které je provedeno touto metodou pokrytí kódu, již vykazuje výborné výsledky v počtu neodhalených chyb, které se blíží k 0% (ne všechny chyby jdou objevit testováním pokrytí kódu). Tento test spočívá ve vyhodnocování všech možných průchodů testovaným kódem. Pokrytí cest musí splňovat kritérium pokrytí podmínek a zároveň musí spojit veškeré vstupní a výstupní uzly s co možná nejmenším počtem průchodů. Tímto testováním ověřím veškeré možnosti běhu aplikace. Největší nevýhodou pokrytí cest je ve složitosti návrhu, který pro dlouhý zdrojový kód představuje obrovský nárůst počtu testů. Tento nárůst je exponenciální, a proto využití pokrytí cest v praxi je téměř nemožné, jelikož se zde pracuje s velice obsáhlými projekty, které mohou obsahovat i několik desítek tisíc řádku kódu. V důsledku této skutečnosti přechází vývojáři aplikací pro HW komponenty z klasického programovaní na modulové funkční programování, na které se dá již aplikovat testování pokrytí cest a testovací nástroj Tessy tuto možnost aplikuje v praxi.

2.8 Rozdíl mezi Unit a Komponent testem[6]

Jak název podkapitoly napovídá, budu se věnovat tématu různých pohledů na testy aplikací. Na první pohled nemusí být rozdíl mezi těmito přístupy testování tak patrný, ale po dočtení této kapitoly snad bude mít čtenář jasnější obraz a nebude tyto dva pojmy zaměňovat.

2.8.1 Unit test

Unit testování by se dalo chápat jako funkční „black box“ testování, kde mám k dispozici vstupní údaje a sleduji údaje výstupní. Na tomto místě je vhodné vysvětlení pojmu *unit* – co si pod tímto pojmem představit? Unit je možno chápat, jako jednotku, takový by byl přímý překlad z anglického jazyka, ale v oblasti programování a testování se pod tímto pojmem chápe i funkce, procedura, program. Hlavním smyslem unit testování je od sebe oddělit funkční bloky a ty nezávisle otestovat. Toto je ale dosti idealizovaný pohled, neboť i v tomto testování dochází ke vzájemným iteracím jednotlivých funkcí a algoritmů. Pokud přesto chci otestovat jedinou funkci, která

[6] Čerpáno z jednoho zdroje:
RAZORCAT DEVELOPMENT GMBH 2009. *Tessy 2.9*: *User manual*. Berlín, 2009

v sobě obsahuje i volání na jiné funkce či procedury, musím je nahradit tzv. stub function, nebo pomocnými objekty, které se snaží simulovat předpokládaný kontext funkce. Testování pomocí stub function je někdy označováno jako *testování hierarchie funkcí*, je to technicky velice obdobné a i pro tento druh testu se používá termín unit test. Hierarchické testování může být považováno za integrační testování jednotlivých funkcí v hierarchii. Obecně musí všechny funkce projít testy v závislosti na předešlé funkci, lépe řečeno na předešlé funkci v hierarchii. Pro toto uspořádání funkcí se zavádí termín *modul*, ale to je nepřesné použití tohoto termínu. Testování zdrojového kódu v jazyce C není automaticky připraveno pro modulové testování, jak by z termínu mohlo vyplývat. Není to objekt modulového testování, protože je definován pouze syntakticky. Modul pro modulové testování je definován sémanticky. Proto bych byl velice opatrný s použitím termínu modul pro hierarchii funkcí, přesto jsem se s tím v praxi často setkal.

2.8.2 Komponent test[7]

Jiná architektura programu než je hierarchie funkcí, kde se funkce nemusí vzájemně volat, ale přesto jsou propojeny, například zpracováváním stejných údajů, nebo sběrem dat za dosažení stejného cíle se může nazývat *komponenta*. Toto pojmenování je mnohem výstižnější než pojem modul, kterému bych se chtěl vyhnout ze stejného důvodu jako v předešlé kapitole. Jednoduchým a známým příkladem může být zásobník. Tento datový typ využívá funkce *push* a *pop* a ty poté nezávisle volá v procedurách.

V komponentě je alespoň jedna funkce, která je stimulem pro ostatní vnitřní funkce komponenty a uvádí je do pracovního režimu. Takováto funkce nemusí být pouze jedna, ve skutečnosti to bývá větší množství funkcí. V reálném světě je také běžnější, ovlivňování vnitřních součástí systému větším počtem okolních činitelů.

Otestování komponenty již není jen pouhý sled volání jednotlivých funkcí uvnitř systému, ale je to systematické volání funkcí komponenty ve správném časovém okamžiku. Jednotlivé funkce komponenty stimulují celý systém a já pouze sleduji odezvu na tyto stimuly. Opět do systému pouštím vstupní data, ale pouze do stimulujících funkcí a následně vyhodnocuji data výstupní. Komponenta může obsahovat i vnitřní funkce, které nemohu ovlivňovat z vnějšího prostředí. Tyto funkce

[7] Čerpáno z jednoho zdroje:
BUECHNER, Frank. HITEX GMBH. *Tessy Tutorial*: *Component / Integration Testing With Tessy*. Karlsruhe, 2009.

jsou ovlivňovány pouze ději uvnitř komponenty. Pokud takové funkce skutečně existují, nemají vliv na komponent testování, jelikož se na komponentu dívám jako na „black box“. Výsledkem komponent testu je pak sekvence příkazů vstupních hodnot volaných z určité komponenty, které ovlivňují funkce v ostatních součástech systému. Výsledek je brán s ohledem na počet možností příkazů k dalším jednotlivým funkcím ostatních komponent.

V předchozím testování jsem nebral v úvahu časové zpoždění signálu při průchodu komponentou, zdržení signálu z vnějšího okolí, nebo časovou prodlevu výstupního signálu z testované komponenty do následující části systému. Nicméně pro testování je důležité i dodržení časového rámce signálů a zpoždění může hrát vysokou roli, zvláště u přístrojů zajišťující bezpečnost, kde je přesné časování nutnou podmínkou správné funkce systému. Aby bylo možné otestovat správné chování simulovaných signálů, musím mít dobře zavedenou časovou základnu. Ta je obvykle pod kontrolou operačního systému, který pracuje v reálném čase. Operační systém má na starost i další důležitou funkci a tou je převod spojité časové veličiny na diskrétní hodnoty. Poté již můžeme funkce volat v určitém časovém intervalu (nejběžnější hodnota je 10ms, ale tento interval je možno měnit). Tyto časové intervaly jsou pod plnou kontrolou operačního systému. Abych věděl, kdy se má daný cyklus provést, zavádí se pojem „výzva k volání funkcí“. Výzvu představuje tzv „heartbeat“ funkce. Ta představuje časovou referenci, pro testování chování komponenty. Příkladem může být vstupní signál, který je volán každých 10ms a „heartbeat“ funkce, která kontroluje intenzitu osvětlení. Pokud jednotka dostane impuls na zvýšení osvětlení, „heartbeat“ funkce tuto informaci předá systému, který provede cyklus spuštění funkcí na zvýšení intenzity osvětlení a po 10ms dojde k navýšení intenzity.

2.9 Funkční bezpečnost

Programové vybavení není jediným bezpečnostním rizikem, které je nutné v automobilovém průmyslu řešit. Každý elektronický výrobek prochází životním cyklem vývoje od samotného návrhu výrobku až po jeho smrt – v dnešní době může být za smrt výrobku považována i jeho recyklace a další použití. Nejen výrobky, ale i technologické postupy podléhají přísné kontrole a dodržování norem. Tyto pravidla platí i pro vývoj SW, a proto bych je okrajově zmínil, jelikož testování zahrnuje i požadavek na funkční bezpečnost. Veškeré postupy a předpisy jsou uvedeny v evropských normách IEC/EN 61508 a IEC 61511, které platí i pro Českou republiku.

Tyto normy určují funkční bezpečnost technologických zařízení a postupů nejen v elektrotechnickém průmyslu, ale v celém odvětví výroby obecně.

2.9.1 SIL – Stupně bezpečnosti[8]

Podle normy IEC 61508 jsou definovány čtyři stupně integrity bezpečnosti SIL (Safety Integirty Level). Na určení SIL má značný vliv frekvence výskytu poruch, která je dána součinem časového intervalu vzniku chyb (počet řešení krizových situací) a poměrem chybových stavů k bezchybnému průběhu činnosti výrobku. Pro regulaci výskytu chyb jsou možné dvě varianty. Jednou je snižování počtu chybových stavů a druhým je zvýšení časového intervalu mezi vznikem chyb. Pojem, který se používá pro frekvence poruch, je *střední doba mezi poruchami*. Pokud je střední doba mezi poruchami jeden rok a nezpůsobí žádné vážné materiální škody, ani újmu na zdraví obsluhy, je možné plnit pouze standard SIL1. Při výskytu vážné chyby, i kdyby byla perioda dlouhá deset let, je nutné plnit vyšší standard SIL. Za vážnou poruchu se považuje ohrožení zdraví a života člověka a velké materiální škody. Předcházení vážným poruchám znamená zvýšení střední doby do poruchy, ať použitím modernějších technologií, nebo postupů. Pro systémy, které vyžadují vysoký stupeň bezpečnosti, se volí redundantní zálohování. Chod takového systému je většinou kontrolován dohlížecím systémem (watchdog). Následující přehled stupňů SIL slouží pro orientaci, jak je možné na určité úrovně nahlížet.

SIL 1 – nejnižší úroveň, používají se běžná řešení a postupy

SIL 2 – již musí být kvalitní technický návrh

SIL 3 – použití nejlepší dostupné technologie

SIL 4 – běžně se nevyužívá – nejvyšší stupeň

Označení SIL 1 až SIL 4 přiřazujeme jednotlivým částem systému, podle pravděpodobnosti výskytu chyb a možného dopadu rizik. Platí, že celkový systém má takový SIL, jako je nejnižší úroveň SIL části systému. S určením SIL souvisí ještě jeden pojem a tím je *bezpečný stav*. Bezpečný stav by se dal definovat jako stav, který vyústí k bezpečné akci. Pro každý systém znamená bezpečný stav něco jiného. Příkladem

[8] Čerpáno z jednoho zdroje:
SOUKENÍK, Martin. *Analýza funkční bezpečnosti parních turbín s příslušenstvím*. Zlín, 2010. Diplomová práce. Univerzita Tomáše Bati ve Zlíně.

uvedu světelnou signalizaci pro řízení dopravy na křižovatkách pozemních komunikací. Při vzniku poruchy je za bezpečný stav považována signalizace červeným světlem. Bezpečná akce je v tomto případě zastavení dopravy a tím zabránění vzniku dopravní nehody.

2.9.2 ASIL a norma ISO 26262[9]

Pro potřeby automobilového průmyslu byla zavedena norma ASIL jako automotive SIL. Spolu s tímto pojmem souvisí zavedení normy ISO 26262, která byla zveřejněna v červenci 2009. Tato norma popisuje současný stav v rozvoji příslušných bezpečnostních funkcí vozidel. ISO 26262 vychází z normy IEC 61508, která je výchozí normou pro elektrické a elektronické systémy (E/E). Norma ISO 26262 je základní bod pro veškeré bezpečnostní aktivity v sektoru automotive, analyzuje a vyhodnocuje rizika, která mohou nastat. Hlavní podnět ke vzniku takovéto normy dali němečtí výrobci automobilů, jelikož podle německého práva jsou odpovědni za škodu způsobenou nesprávnou funkcí výrobku, tedy i automobilu.

Někdo by mohl namítat, že zavádění dalších norem, je zbytečné a jen to prodlužuje uvedení výrobků na trh. Rostoucí složitost celé oblasti automobilového průmyslu nutí výrobce řešit problémy typu kompatibility a zpoždění signálu v elektronickém systému. Cílem normy ISO 26262 je sjednotit postupy a testování úrovně bezpečnosti pro všechny automobilové E/E systémy.

Norma ISO 26262 používá k dosažení stanoveného cíle systém kroků k řízení funkční bezpečnosti, vývoje produktu jak na HW, tak SW úrovni. Norma dále stanovuje bezpečnostní předpisy a doporučení v celém procesu vývoje výrobku od myšlenky až k vyřazení z provozu. V neposlední řadě norma nařizuje přijatelné míry rizika při výrobě a definuje celý proces testování. Celý proces by se dal shrnout do několika kroků:

1) Popis životního cyklu (definice vlastností výrobku, vývoj, výroba, provoz, servis, vyřazení z provozu)

[9] Čerpáno ze dvou zdrojů:
SCHAFFNER, Johanna. *Gefahrenanalyse und Sicherheitskonzept nach ISO 26262 für Fahrerassistenzsysteme*. Lindau, 2011.
What is the ISO 26262 Functional Safety Standard?. NATIONAL INSTRUMENTS. *National Instruments* [online]. 23.2.2012 [cit. 2012-04-21]. Dostupné z: http://zone.ni.com/devzone/cda/tut/p/id/13647

2) Definování rizik pro specifický automobilový průmysl a stanovení třídy rizik (ASIL – úrovně integrity bezpečnosti v automotive)
3) Využití ASIL pro určení nezbytných bezpečnostních požadavků pro dosažení přijatelného rizika
4) Stanovení požadavků pro ověření a potvrzení dosaženého stupně bezpečnosti

Všechny tyto kroky musí být náležitě plněny při sériové výrobě automobilů a jejich dodržování je kontrolováno bezpečnostními audity. ASIL je klíčovým prvkem pro plnění normy ISO 26262. Na počátku vývoje je stanovena hodnota ASIL vzhledem k možným rizikům, které mohou při výrobě a následném plnění funkce výrobku vzniknout. Základní otázka je – pokud vznikne porucha na daném výrobku, co to znamená pro celý systém (automobil) a jeho uživatele (řidič) a ostatní účastníky silničního provozu. Odhad rizika poruchy je založen na kombinaci pravděpodobnosti vzniku poruchy, ovladatelnosti systému uživatelem a možném výsledku při vzniku kritické chyby. ASIL se nezabývá technologiemi, které jsou použity v systému, ale pouze škodami, které by utrpěl uživatel a okolí použitého výrobku. Stupně ASIL jsou rozděleny do čtyř skupin, podobně, jako je tomu u SIL. Jednotlivé skupiny jsou označeny A, B, C nebo D, kde D je označení pro nejdůležitější bezpečnostní procesy a nejpřísnější testovací předpisy. Norma ISO 26262 stanovuje minimální požadavky na testování, které musí jednotlivé stupně ASIL podstoupit. Tím je definováno chování systému pro zajištění potřebného stupně bezpečnosti. Jako příklad můžeme uvést ztrátu funkce stěračů. Analýza bezpečnosti musí určit, jaký bude mít vliv nefunkčnost stěračů na viditelnost řidiče a jak by to mohlo ovlivnit ostatní účastníky silničního provozu. Musí být popsán návod, jakou metodou určit stupeň bezpečnosti a uvést bezpečnostní postupy při výskytu této chyby. Současné automobily jsou vyráběny s vysokým stupněm bezpečnosti a norma ISO 26262 slouží pro standardizaci postupů v celém automobilovém odvětví.

Hardwarové komponenty ve vazbě na ASIL mají ukázat dva hlavní cíle: ukázat, jak část zapadá do celkového systému a posoudit druh poruchy. Pokud je HW část jednoduchá dá se posoudit jako celek, u složitějších struktur je nutno systém rozložit na menší části a až poté jednotlivé komponenty posuzovat pomocí ASILu a provádět testy. Testování spočívá v určení provozních podmínek s ohledem na prostředí.

Výsledky testů jsou analyzovány numerickými metodami, které jsou zpracovány do podrobné dokumentace. Tento dokument musí obsahovat data, při jakých podmínkách bylo testování realizováno, přesný průběh testů a zadané vstupní parametry.

Testování softwarových komponent představuje jednodušší činnost, pokud probíhá v celé fázi vývoje výrobku. V ASIL jsou definovány funkční požadavky na jednotlivé komponenty, které musí být splněny. Důležitou vlastností SW komponent je předvídatelné chování softwaru při přetížení a v krizových situacích. Mezi nejvýznamnější SW komponenty patří operační systémy, knihovny funkcí, databáze, ovladače. Aby SW komponenty splnily požadavky na určitou úroveň ASIL, jsou zkoušeny za normálních provozních podmínek, kde se vstupními parametry vložíme záměrně chybu a sledujeme reakci na tento abnormální stav. Důležité je, aby systém při tomto abnormálním stavu vykazoval stabilitu a nebyla přerušena jeho funkce.

Pro testování SW aplikací je neméně důležité vědět, jakou úroveň důvěry má daný testovací software. Tento parametr bývá označen TCL (Tool Confidence Level) a je stanoven čtyřmi úrovněmi. TCL1 TCL2, TCL3, nebo TCL4, kde TCL1 je aplikace s nejvyšší úrovní důvěry a TCL4 s nejnižší. Aby bylo možné v rámci normy ISO 26262 uplatnit TCL, musí být splněno mnoho požadavků. Musí být vymezen ASIL, testovací nástroj musí mít uživatelskou příručku, jedinečné identifikační číslo a verzi, musí být popsán instalační postup a další důležité funkce.

TCL se skládá ze tří částí. První část je plán kvality, který se vytváří v první fázi vývoje výrobku, kdy se začínají tvořit bezpečnostní limity. V této části se definují použité SW aplikace pro testování a druhou věcí je definice požadované úrovně bezpečnosti. Po dokončení první fáze musí být známy položky, jako jsou verze a identifikační číslo softwarového nástroje, pomocí kterého se bude provádět testování. Mezi další informace patří prostředí, kde je možné tento nástroj použít, popis nástroje, uživatelský manuál a definován stupeň ASIL.

Druhou částí TCL je analýza klasifikace SW nástroje. Zde se určují hladiny spolehlivosti. Hladina spolehlivosti obsahuje dvě hlavní složky. První z nich je dopad nástroje (Tool Impact - TI) na bezpečnostní požadavek. Druhým je nástroj detekce chyb (Tool Error Detection - TD). S ohledem na volbu těchto dvou parametrů volíme TCL. U položky TI je voleno mezi TI0 a TI1. Pokud neexistuje možnost, že by porucha SW nástroje mohla porušit stupeň bezpečnosti, zadá se TI0, v ostatních případech TI1.

Nástroj detekce chyb je klasifikován od TD1 do TD4. Vysoká míra důvěry je označena jako TD1, střední TD2, nízká TD3. Označení TD4 se používá pro systémy, které odhalí chybu pouze náhodně.

Hodnoty TCL		
	TI0	TI1
TD1	1	1
TD2	1	2
TD3	1	3
TD4	1	4

Tab. 1 Výběr TCL s ohledem na parametry

V Tab. 1 je vidět vliv jednotlivých parametrů na konečnou volbu TCL. Pokud je parametr TI0, nemusí se již brát ohled na TD. Při TI1 se vyhodnocuje i parametr TD a platí, že čím vyšší TD, tím horší celkový výsledek TCL.

Třetí část TCL obsahuje výsledky a podrobnou dokumentaci, že veškeré testy byly dokončeny a požadavky na testování splněny. Pokud se při testování vyskytly chybné výstupy, měly by zde být uvedeny možné příčiny. Chybné výstupy by měly být dále analyzovány a podrobněji zkoumány.

U starších systémů, které nebyly vyrobeny dle normy ISO 26262, ale přesto vykazují spolehlivé chování, se používá označení „*prokazatelně používané*“. Tyto systémy jsou vyrobeny dříve, než byla zmíněná norma ustanovena a přesto jejich funkčnost vykazuje vysokou míru bezpečnosti, jako by byly touto normou kontrolovány. Nemá proto smysl osvědčené výrobky a postupy podrobovat novému zkoumání, pokud byly ověřeny v průběhu několika let na milionech kusů výrobků. Často se toto označení používá u výrobců automobilů, kteří již dříve dbali na vysokou spolehlivost a bezpečnost svých výrobků. Mohlo by zde dojít k omylu, že tedy není norma ISO 26262 opodstatněná a potřeba. Jak jsem zmiňoval výše, nové výrobky a postupy jsou stále složitější, a proto je potřeba při zavádění nových technologií dbát na vyšší bezpečnost. Norma je proto zaváděna výhradně na současné postupy. Druhým přínosem je sjednocení postupů a standardů. Dosavadní výsledky ukazují, že se norma ISO 26262 dobře přizpůsobuje současné bezpečnostní politice v průmyslovém odvětví. U řady společností se již projevují výhody hodnocení rizik. Je důležité, aby společnosti,

které chtějí implementovat normu ISO 26262, pochopili, že je nutné stanovit bezpečnostní požadavky již během vývoje a tyto požadavky také důkladně testovali.

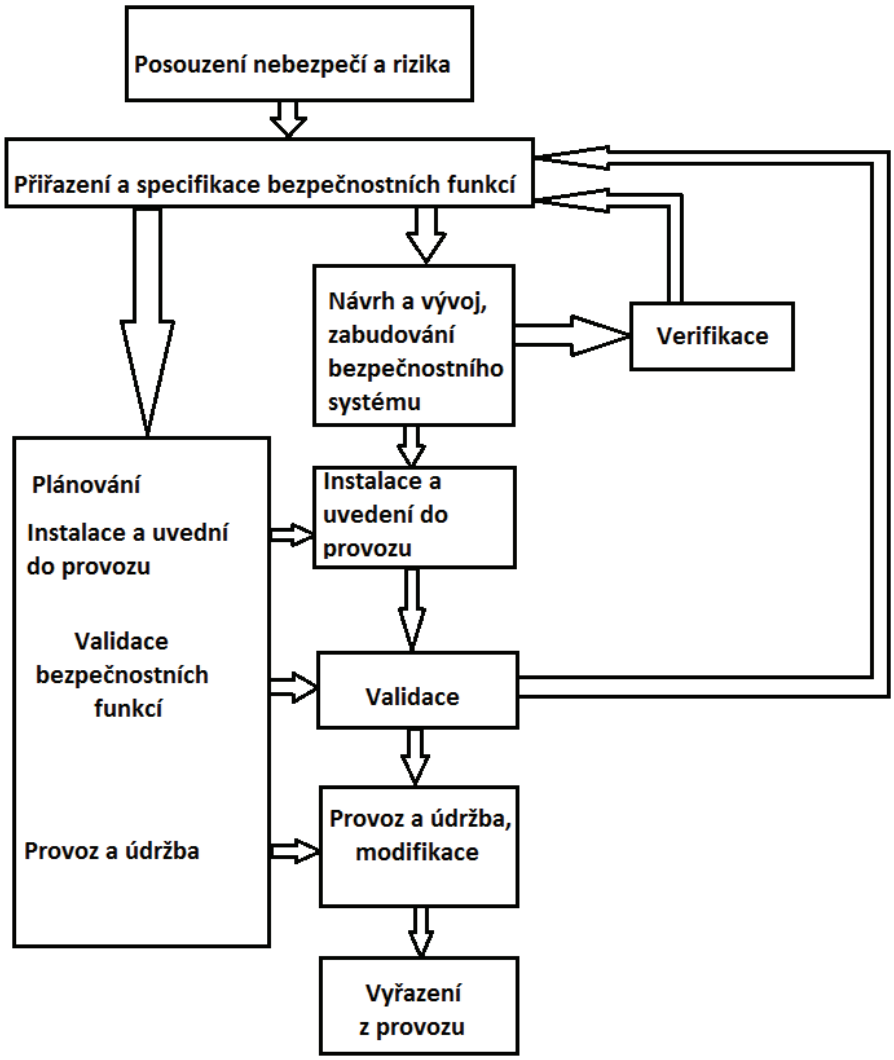
Velkou výhodou při dodržování norem je použití již otestovaných procesů a výrobků. Důležité je prokázání, že daný výrobek, nebo postup opravdu prošli testy a dané normy splňují. Tímto způsobem mohou společnosti ušetřit obrovské finanční prostředky a značně urychlit vývoj a výrobu.

2.9.3 Životní cyklus bezpečnosti

Výrobky, výrobní procesy, všechny tyto prvky mají svůj životní cyklus. S životním cyklem zmíněných prvků souvisí i životní cyklus bezpečnosti, který je doprovází. S rostoucím stářím produktů se zvyšuje jejich chybovost, ale tato skutečnost je kompenzována vyšším know-how o daném výrobku či technologickém postupu, a proto je možno tuto chybovost snižovat. Dalším důležitým faktorem zvyšování bezpečnosti je kombinace různých bezpečnostních systémů, které pracují na různých principech – nejen elektrických. Jednotlivé fáze životního cyklu systému ukazuje zjednodušený blokový diagram na Obr. 5.

Blokové schéma názorně zobrazuje, že se tento cyklus netýká pouze výrobce zařízení, ale i provozovatele, který má povinnost se o zařízení starat pravidelnou údržbou.

Na tomto místě je vhodné zmínit, že veškeré uvedené informace a postupy se vztahují nejen na hmotné výrobky, ale i na softwarové aplikace. To je také důvod, proč tu tuto problematiku zmiňuji, jelikož při testování SW je důležité na toto brát ohled a uvědomit si, v jakém stupni bezpečnosti se tester pohubuje a jaký vliv má testování na koncový produkt. Určení nebezpečí a rizik, ale zdaleka nezávisí pouze na jednom člověku. Většinou je k této činnosti vybrán celý tým lidí, kteří se touto problematikou zabývají a mají zkušenosti z praxe. Tento tým je schopen pomocí HAZOP (Hazard and operability study) určit nebezpečí procesu výroby. Studie HAZOP je technika, kde se určuje nebezpečí a provozuschopnost průmyslového procesu. Formální potup je popsán v normě ČSN IEC 61822.



Obr. 5 Blokové schéma životního cyklu bezpečnostního systému

2.9.4 Stupně bezpečnosti SIL u SW aplikací[10]

Výše zmíněné postupy se aplikují pro průmyslové procesy, kterými se ale v poslední době stává také software, takže je nutno tyto normy aplikovat i do této oblasti působnosti. Tento trend vývoje je dán velkou implementací programovatelných součástek na místa, kde to bylo dříve nemyslitelné. Kategorie SIL 1, 2, 3 jsou sjednoceny a obsahují stejné požadavky na softwarové aplikace.

[10] Čerpáno z jednoho zdroje:
SOUKENÍK, Martin. *Analýza funkční bezpečnosti parních turbín s příslušenstvím*. Zlín, 2010. Diplomová práce. Univerzita Tomáše Bati ve Zlíně.

Norma rozděluje software do tří skupin:

1. Aplikační software
2. Vývojový software, obslužný software
3. Firmware – software pro vestavěné systémy

Další definice určují druhy vývojových jazyků:

1. Pevný programovací jazyk (FPL – Fixed Program Language)
2. Jazyk s plnou variabilitou (FVL – Full Variability Language)
3. Jazyk s omezenou variabilitou (LVL – Limited Variability Language)

FPL se používá v nejjednodušších aplikacích a zařízení, jelikož uživatel může měnit pouze parametry, které ovlivňují provoz zařízení – měřící rozsahy, atd. Uživatel si nemůže přidávat vlastní funkce, měnit smysl stávajících funkcí. Všechny funkce jsou pevné a nadefinované výrobcem, který také definuje nastavitelné parametry v přiložené dokumentaci. U tohoto typu přístrojů se nepředpokládají chyby SW, jelikož podstupují velice komplexní testy.

FVL jsou klasické programovací jazyky, jako jsou C, C++ a další. Typické vlastnosti těchto jazyků jsou real-time prostředí a využití operačním systémem. Programování pomocí FVL je náročnější a je potřeba mít programátorské dovednosti, umět syntaxi použitého jazyka, a proto aplikace psány tímto druhem jazyků vyžadují odborníky a specialisty na toto odvětví.

LVL jazyky se nejvíce používají v PLC systémech. Programování je zde realizováno pomocí nadefinovaných funkcí, které prošli testováním a je možno je pro danou aplikaci využít. Záleží na samotném uživateli, jak jednotlivé funkce využije. Nejpoužívanější nástroje jsou Ladder Diagram (LD), Function Block Diagram (FBD), Sequential Function Chart (SFC), Instruction List (IL), Structured Text (ST). Více o těchto nástrojích se můžete dovědět z literatury.[11]

Norma, která řeší problematiku FPL a LVL je ČSN EN 61511. Odlišný přístup k FVL musí být řešen jinou normou a tou je ČSN EN 61508.

[11] KOVÁŘ, Josef, Zuzana PROKOPOVÁ a Ladislav ŠMEJKAL. *Programování PLC*. Zlín, 2010. Dostupné z: http://www.spszl.cz/soubory/plc/programovani_plc.pdf

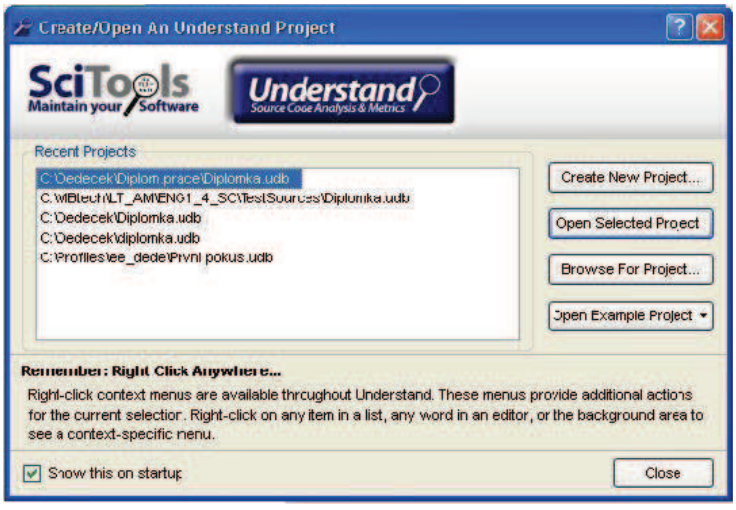
Důležitá podmínka, která určuje bezchybnost systému, je nutnost aplikaci otestovat nezávislým subjektem. Pro testování se používá takzvaný V - model, který je standardizován a obecně používaný model. V diagramu je ukázáno, jak jsou jednotlivé fáze vývoje provázány s jeho testováním. Tímto způsobem je možno vytvářet aplikace, které mají téměř nulovou chybovost a pokud nedojde k fyzickému poškození hardwaru, dokáží plnit požadovanou funkci po celou dobu života výrobku. Pomocí V - diagramu dokážeme snížit náklady na pozdější reklamace a opravy výrobků vzhledem k jeho SW částem. Uspořené náklady se dají využít na vývoj a zlevnění jednotlivých součástek. V - model jsem podrobněji popisoval v kapitole 2.4 Modely vývoje SW.

3 Základy programu Understand

Pro velice obsáhlé projekty je výhodné používat program Understand. Část projektu, který jsem testoval, obsahoval pouze několik souborů, ale i vzhledem k této skutečnosti jsem ocenil výhody tohoto SW. Musel jsem upravovat hlavičkové soubory a program Understand mi tuto práci velmi zjednodušil. Cílem této kapitoly je seznámit čtenáře se základním použitím programu, nebudu zde popisovat veškeré dostupné funkce programu, ale pouze ty části, které jsem využil v rámci diplomové práce. Následující podkapitola se zabývá vytvořením projektu, popis je doplněn obrázky pro názornou představu. Mezi funkcemi programu jsem vybral zobrazení větvení procedur. Tuto funkci ocení začínající tester při určování počtu testovacích kroků. Více o tomto produktu je možno se dovědět z literatury.[12]

3.1 Založení projektu

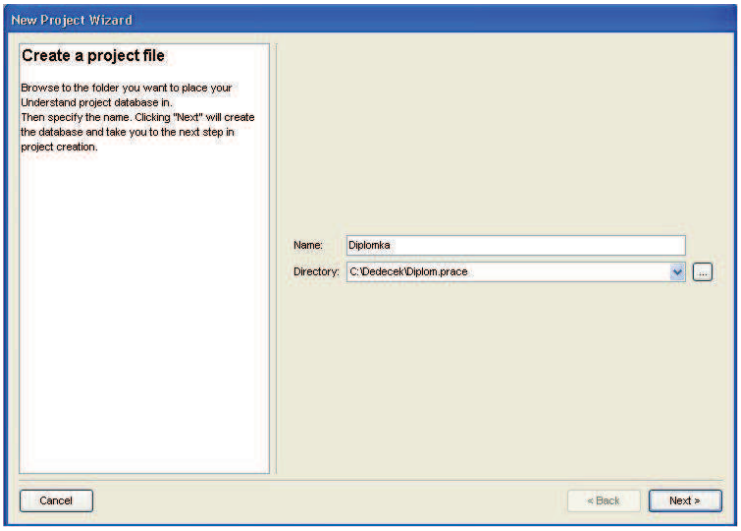
Podobně, jako v mnoha jiných SW aplikací patří mezi první kroky založení projektu. Po otevření aplikace se naskytne pohled, jako je na Obr. 6. Zde mohu vybrat již existující projekt a pokračovat v práci, nebo vytvořit projekt nový. Pokud existuje více projektů, mohu mezi nimi libovolně přepínat a otevřít až požadovaný projekt. Na Obr. 6 je vidět, že je vytvořen projekt *Diplomka*, ale přesto ukáži vytvoření nového projektu.



Obr. 6 Založení projektu v programu Understand

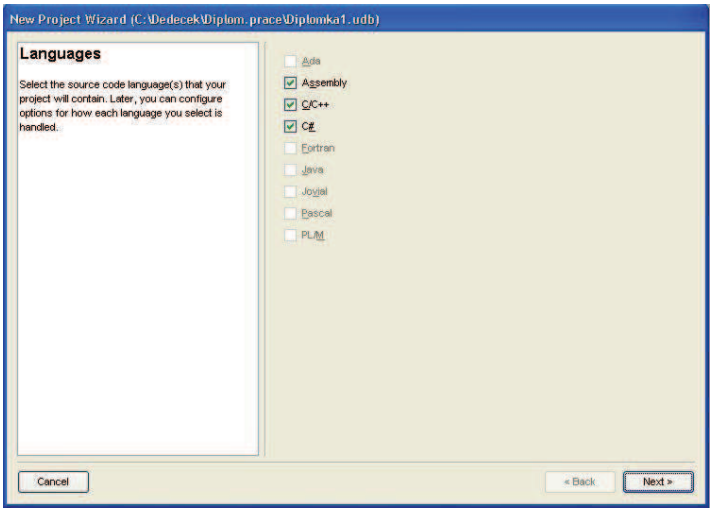
[12] SCIENTIFIC TOOLWORKS, Inc. *Understand*: *User Guide and Reference Manual*. 2.5. St George, 2010. Dostupné z: http://www.scitools.com/documents/manuals/pdf/understand.pdf

Po zadání příkazu vytvoření nového projektu (Create New Project) je možnost zadat název nového projektu a umístění v adresářové struktuře, jak ukazuje Obr. 7.



Obr. 7 Nový projekt v programu Understand

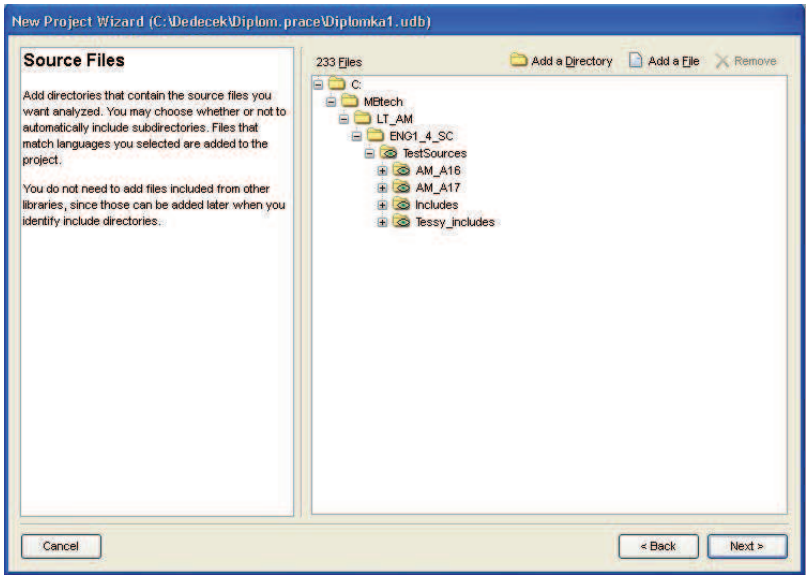
Dalším krokem je vybrání programovacích jazyků zdrojového kódu, se kterými se bude v daném projektu pracovat. Je proto nutné tuto informaci v této fázi zakládání projektu již vědět. Na doporučení konzultantů jsem zvolil všechny licencované jazyky, které jsou ve firemní verzi Understandu k dispozici. Tato skutečnost je zobrazena na Obr. 8.



Obr. 8 Výběr programovacích jazyků zdrojového kódu

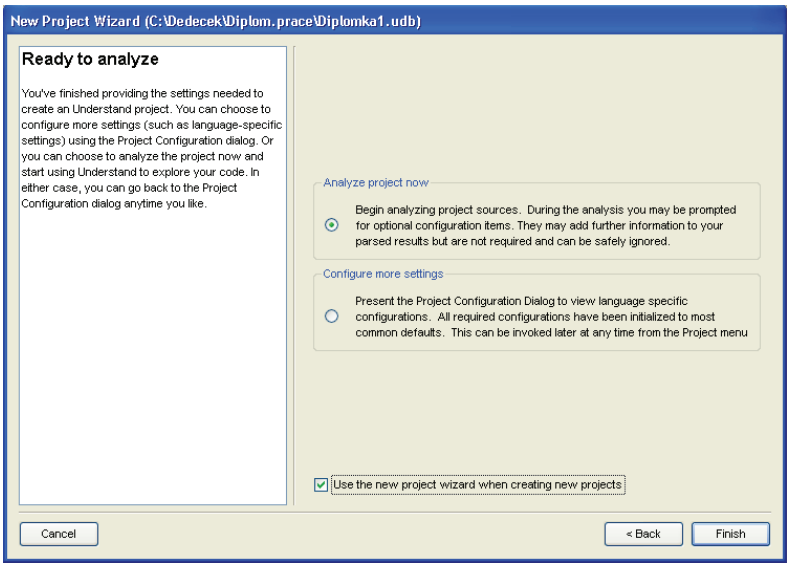
Pokud bych pracoval s Visual Studiem, je možno projekt ve Visual Studiu přímo propojit s programem Understand. Nastane-li změna v jednom z programů, automaticky se to projeví v programu druhém. Mezi oběma programy funguje vzájemná

synchronizace. Jelikož jsem nepracoval ve Visual Studiu, tuto položku jsem přeskočil a do Understand projektu jsem vkládal zdrojové kódy, které mi byly dodány společností MBtech Bohemia s.r.o. Všechny soubory, které jsem chtěl mít v projektu, jsem vybral na následující stránce. Zde je možno vkládat jednotlivé soubory, nebo celé adresáře (což je neocenitelné při zakládání projektu, který obsahuje velké množství souborů). Výběr je ukázán na Obr. 9.



Obr. 9 Výběr souborů projektu

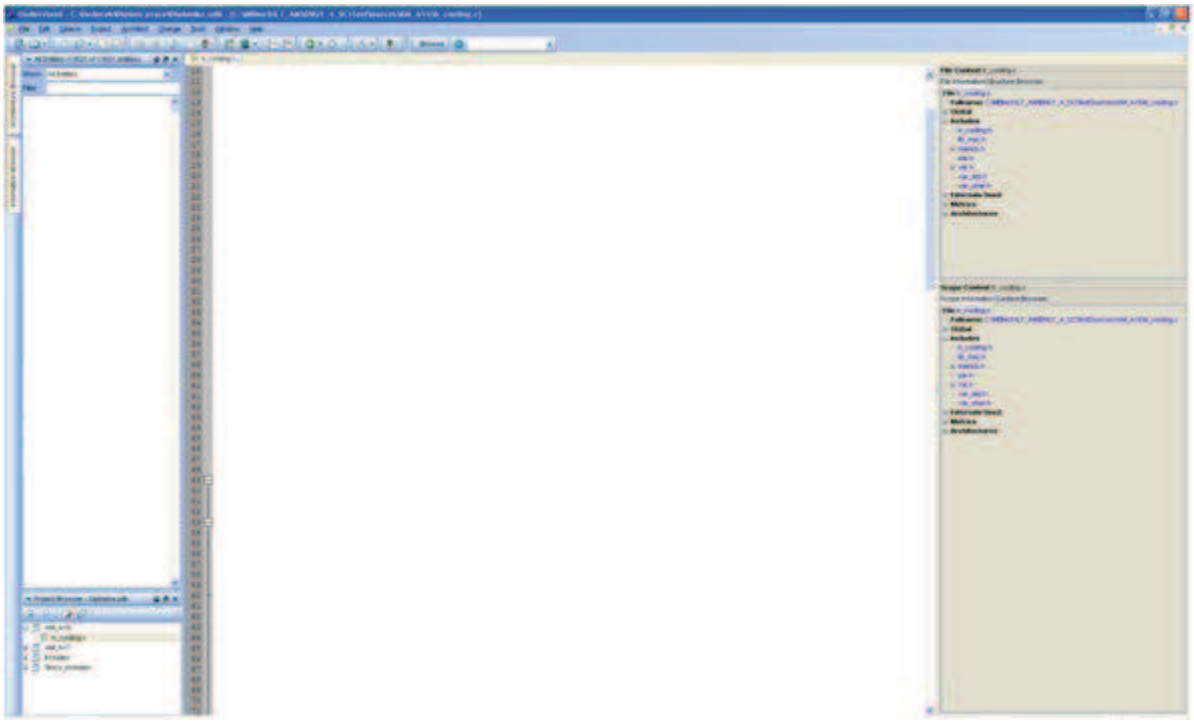
Posledním krokem před úspěšným vytvořením databáze je analýza vybraných souborů, kde se o nich dají získat další užitečné informace.



Obr. 10 Dokončení vytváření projektu

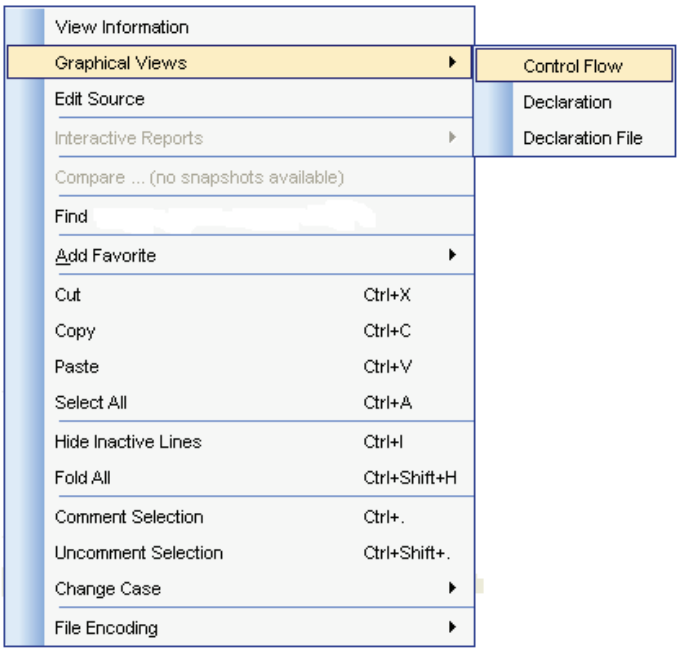
Pokud bych chtěl upřesnit další nastavení, jako je například bližší konfigurace programovacího jazyka, zvolil bych druhou možnost configure more settings. Pro potřebu mého projektu jsem zvolil možnost první - analyzování projektu.

Po dokončení projektu se mi naskytl pohled, jaký je zobrazen na Obr. 11. Struktura programu je velmi podobná většině aplikací, na které jsou vývojáři a testeři zvyklí. V levé části je vidět stromová struktura projektu. V této části pracovní plochy se zobrazují další informace, jako jsou nejčastěji použité proměnné, funkce prohledávání souborů, nastavování různých filtrů a mnoho dalších funkcí. Ve středové části je možno editovat zdrojový kód. Ten se otevře pouhým poklepáním na požadovaný soubor ve stromové struktuře. Po každé úpravě zdrojového kódu je nutné provést uložení souboru, aby se provedené změny projevily i pro SW Tessy. Uložení se provede poklepáním na ikonu rychlého uložení v aplikačním řádku. V pravé části obrazovky jsou bližší informace o právě otevřeném souboru. Pokud je otevřen modulární C soubor, jsou to hlavičkové soubory, které jsou potřeba ke správnému překladu souboru, úplná cesta k souboru na pevném disku, globální proměnné a jiné vlastnosti souboru zdrojového kódu. Obr. 11 jsem z důvodu zachování obchodního tajemství anonymizoval a slouží pouze pro ilustrační představu.



Obr. 11 Prostředí Understand

Program Understand disponuje velmi užitečnou funkcí a tou je zobrazení struktury funkce ve zdrojovém kódu. Strukturou funkce myslím větvení podmínek a příkazů v přehledném schématu. Pomocí tohoto zobrazení si tester udělá lepší představu o počtu testovacích kroků a logickém průchodu proměnných danou funkcí. Struktura se hodí pro testování pokrytí kódu, aby nebyla opomenuta žádná větev funkce. Přesto, že jsem neprováděl testy na pokrytí kódu, pomohlo mi toto schéma lépe pochopit testované funkce. Struktura požadované funkce se zobrazí jejím označením pomocí počítačové myši a poté vyvoláním menu přes pravé tlačítko myši. Zobrazí se menu, jaké je vidět na Obr. 12 po zadání příkazu control flow se vytvoří požadovaná struktura funkce.



Obr. 12 Zobrazení struktury funkce

Příklad zobrazené funkce pomocí control flow je k nahlédnutí v příloze 1. Funkce obsahuje upravené názvy proměnných, slouží pouze jako demonstrativní ukázka. Druhou velmi užitečnou funkcí je zobrazení informace o daném označeném příkazu. Pomocí této funkce se snadno a rychle dohledají informace o deklaracích, definicích a použití příkazu v celé adresářové struktuře projektu. Při hledání definic v hlavičkových souborech to je neocenitelná výhoda.

Na konci této kapitoly bych chtěl zmínit, že použití programu Understand není pro testování nezbytně nutné. Výhody, které nabízí, jsou však tak velké, že použití tohoto, nebo obdobného programu velice doporučuji. Myslím, že při řešení projektu, který obsahuje několik desítek souborů, se již bez podobného programu neobejdete.

4 Využití grafického rozhraní Tessy 2.9 pro testování softwarových aplikací

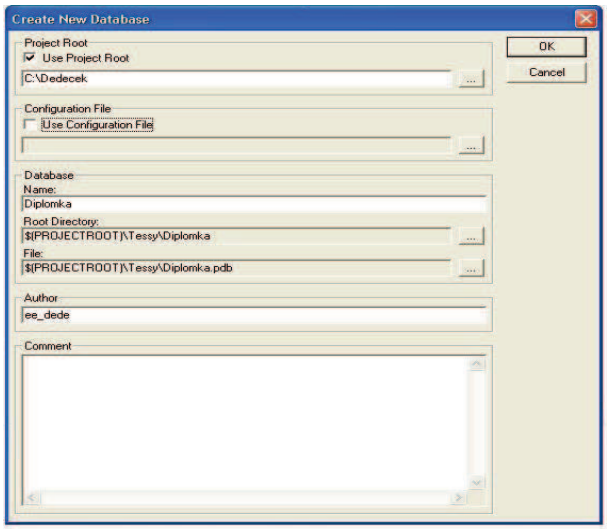
V současnosti si žádná softwarová společnost nedovolí vydat produkt, aniž by nebyl řádně otestován. Je to dáno tím, že se softwarové aplikace používají nejen v oblasti osobních počítačů, ale i v řídicích systémech. Ještě před několika lety nebylo vůbec pomyšlení na to, že by mechanické procesy řídil sám procesor, který bude plně automatizovaný za pomoci SW vybavení. Tyto aplikace musí fungovat bez jediné chyby, protože řídicí jednotky jsou součástí automatizovaných linek, automobilů, CNC strojů a jiných zařízení, kde nehrozí pouze materiální ztráty, ale je zde ohroženo zdraví a život člověka. Proto je kladen zvlášť vysoký důraz na testování, které ovšem s tímto rozmachem automatizace bere potřebný čas na vývoj samotného řídicího SW a přístrojů, kde se aplikační SW používá. Je nutný automatizovaný, ale spolehlivý proces testování softwarových aplikací. Program Tessy od společnosti Razorcat, je jedním ze zástupců automatizovaného modulového testování pro aplikace vyvíjené v programovacím jazyce C. Tento software projde všechny testovací fáze a zajistí veškerou organizaci zkoušek. Součástí je také test management, který činí testování přehlednější. Program Tessy je vytvořen jako integrované grafické uživatelské rozhraní. Pro každou testovací činnost existuje speciální nástroj, který v mnohém tuto činnost usnadní a zautomatizuje. Nástroje jsou implementovány přímo v Tessy, není tedy potřeba žádná dodatečná instalace. Testování pomocí programu Tessy by se dalo shrnout do čtyř kroků:

1) Stanovení testovací úlohy (Test case determination)
2) Stanovení testovaných a očekávaných hodnot (Test data and expected value determination)
3) Spuštění testu (Test execution)
4) Vyhodnocení testu a dokumentace (Test evaluation and test documentation)

Každý testovaný objekt prochází samostatně těmito čtyřmi fázemi testování. Než se pustím do praktického testování softwarových aplikací, představím program Tessy obecně. Prvními kroky jako jsou založení databáze, projektu a modulů, prochází všechny testy. Všem těmto činnostem se věnuji v následující kapitole.

4.1 Založení databáze v programu Tessy 2.9

Pro dodržení správného postupu a vzájemné kompatibility, je nejvýhodnější, když databázi zakládá pouze jeden softwarový tester, většinou to bývá senior tester – neboli vedoucí testovacího týmu. Tento tester obdrží veškerou dokumentaci a kompletní kód, který je potřebný otestovat. Dalším důležitým krokem je nastavení jména projektu. Jméno může být zadané již ve specifikaci, pokud tomu tak není, je potřebné, aby i tuto činnost provedl senior tester a uvedl tuto informaci v závěrečné dokumentaci při odevzdávání výsledku testů. Při vytváření databáze již musí testovací tým vědět, jaký kompilátor bude potřeba pro překlad zdrojového kódu, jelikož se zde udává možnost použít configuration file. Configuration file se používá pro konkrétní nastavení bez možnosti měnit prostředí v průběhu testu. SW testeři ve společnosti MBtech Bohemia s.r.o. preferují ruční nastavení, a proto jsem i já zvolil tuto možnost při vytváření databáze a configuration file jsem nepoužil.

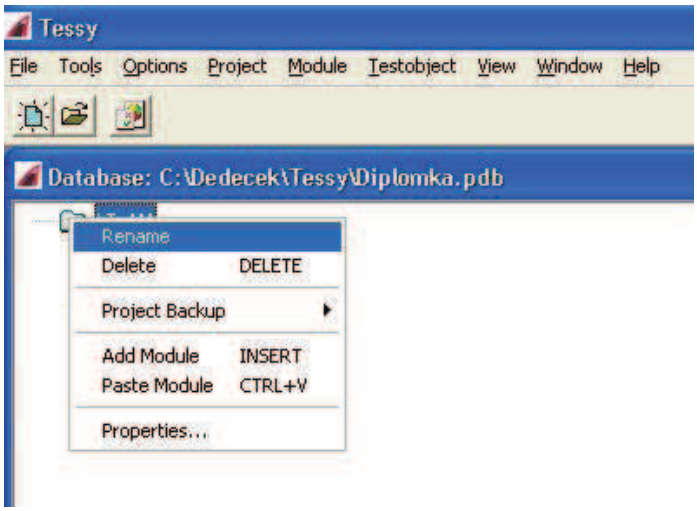


Obr. 13 Vytvoření databáze

4.2 Vytvoření projektu a modulů

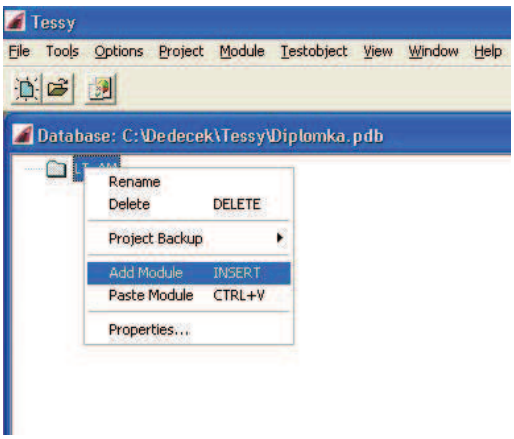
Dalším krokem je nastavení project root. Je to adresář, kde jsou uloženy soubory připravené k otestování, hlavičkové soubory, dokumentace k testům, specifikace produktu a další soubory nezbytné pro spuštění a otestování projektu. Project root je také výchozí místo, kam se budou ukládat vytvořené dokumentace k dokončeným testům. Adresář project root pouze usnadňuje organizaci celého testování, jak později ukáži, adresář pro ukládání výsledků testů je možno měnit. Po dokončení databáze se automaticky v dalším kroku vytvoří projekt, který se jmenuje jako právě vytvořená databáze. Pokud se testuje pouze jeden projekt, jméno je možné ponechat, ale většinou

se jména upravují, aby tester věděl, který projekt právě testuje. Jméno projektu může být stejné, jako je jméno modulu, který se vytváří o úroveň níže. Tento popis se používá převážně pro modulární typy projektů, které obsahují pouze jeden modul. Při testování více modulů jsou jejich jména uvedena ve specifikaci SW. Přejmenování projektu se provede zobrazením menu a prvním příkazem rename, jak je vidět na Obr. 14.



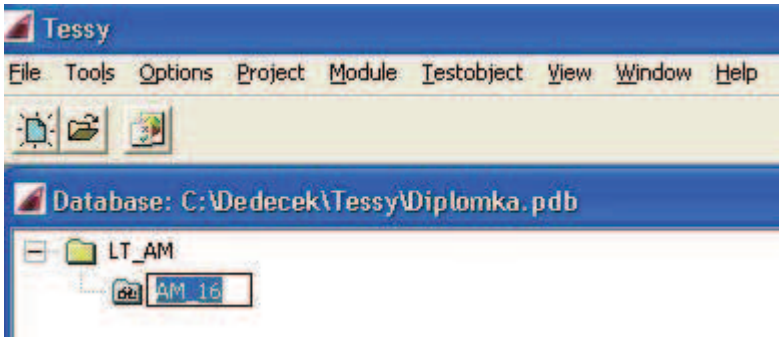
Obr. 14 Přejmenování projektu

Poté se může přistoupit k vytvoření modulů projektu. Důležité je mít označen projekt, kam je požadovaný modul potřeba umístit. Pokud se vytváří nový modul, který nebyl ještě nikde zpracován, přidání se provede pomocí menu a příkazu add module, nebo klávesou insert. Tímto způsobem jsem vytvořil nový modul, který jsem musel nakonfigurovat a vložit testovaný C – soubor. Pokud bych chtěl otevřít již existující modul, který je uložen na pevném disku počítačové stanice, přidání bych provedl pomocí standardního menu na hlavní liště, kde pod položkou Module použiji příkaz restore a pouze bych vybral požadovaný modul v adresářové struktuře stanice.



Obr. 15 Přidání nového modulu

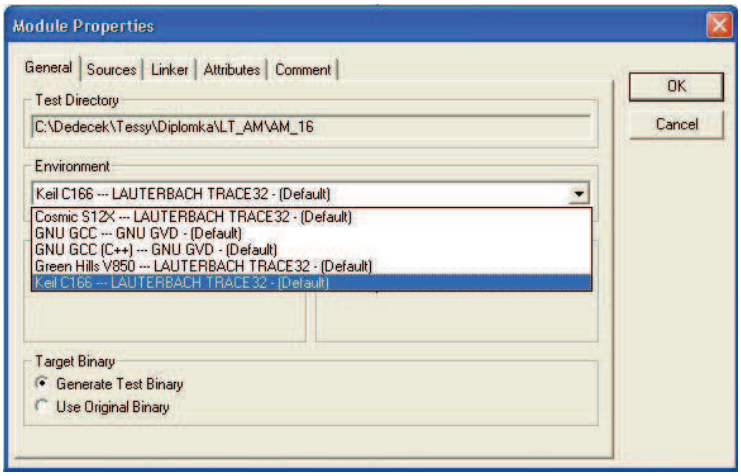
Přidání nového modulu je zobrazeno na Obr. 15. Po vytvoření nového modulu jsem jen provedl přejmenování. Z Obr. 16 je možné zjistit veškeré informace o pojmenování vytvořených struktur. Databáze nese jméno *Diplomka*, projekt je pojmenován LT_AM a vytvořený nový modul AM_16.



Obr. 16 Přejmenování modulu

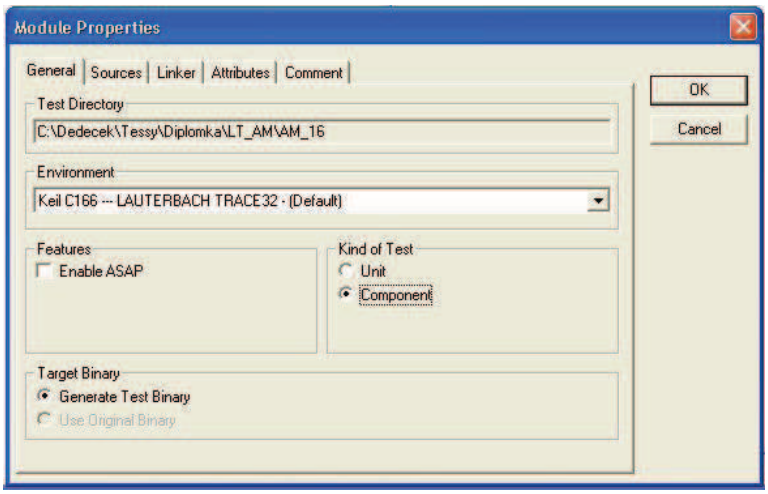
4.3 Nastavení parametrů modulu

Dalším krokem je nastavení parametrů modulu, které se provede vyvoláním menu na požadovaném modulu a výběrem příkazu properties (vlastnosti). Pro moje testování je důležitá položka kind of test (druh testu) hned v první záložce General, kde je nutno označit component (komponent – funkční testování). Nastavení druhu testování nelze již v pozdější fázi měnit, a proto musím od počátku testování vědět, jakým druhem testu se budu zabývat. V této záložce se nastavuje i typ kompilátoru (General/Enviroment), který nastavuji na Obr. 17. Pokud bych při vytváření databáze použil configuration file, v nabídce pro výběr kompilátoru by byl uveden pouze ten, který je definován v tomto souboru.



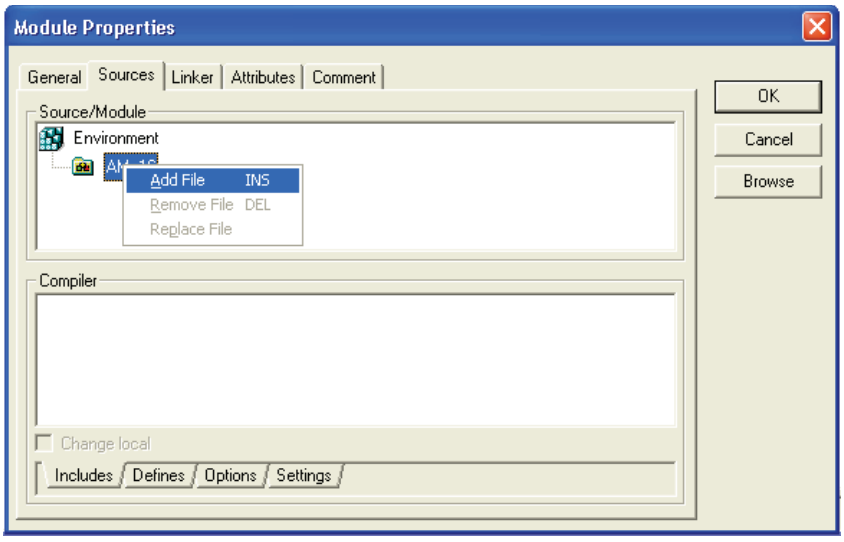
Obr. 17 Výběr kompilátoru

Obr. 18 ukazuje správné nastavení záložky General. Pokud v nabídce environment není požadovaný kompilátor, musí se povolit v TEE (Tessy Environment Editor). Podmínkou je instalace požadovaného kompilátoru.



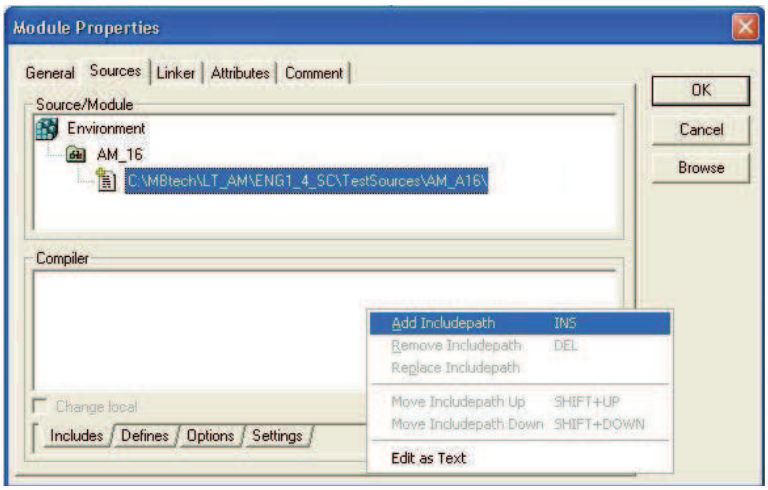
Obr. 18 Výběr komponent testu

Obr. 19 zobrazuje přidání C - souboru, který je nutné otestovat. Toto přidání se provede v záložce Sources (zdroje). Musí být označen modul, kterému se požadovaný soubor přiřadí. Poté se jen klávesou Insert, nebo vyvoláním menu vloží testovaný soubor. V adresářové struktuře project root, nebo v jiném umístění na pevném disku pracovní stanice stačí vyhledat cestu k požadovanému souboru a přiložit ho.



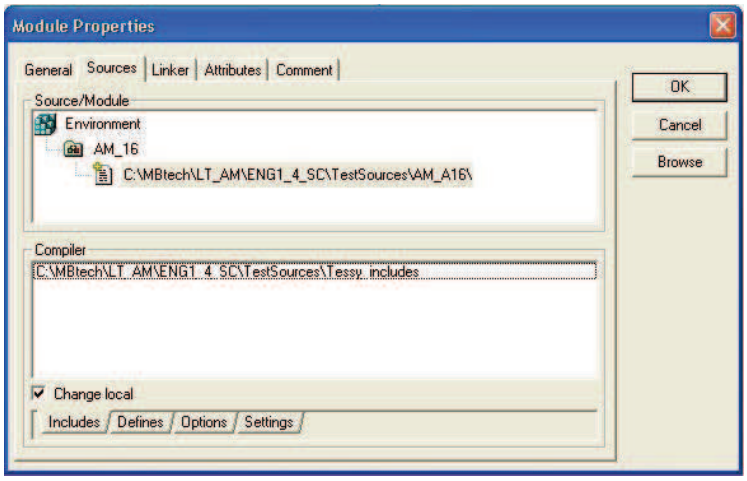
Obr. 19 Otevření testovaného C-filu

Následující krok je velice důležitý s ohledem na spuštění testovaného C – souboru (typicky přiřazení H – souborů, ale mohou to být i jiné soubory, bez kterých není možné otevření požadovaného souboru). Nutné je mít přímo označen C – soubor a až po té přiřazovat H – soubory. Správné označení a přidání H - souborů ukazuje Obr. 20. Častou chybou bývá přiřazení H – souborů o úroveň výše, a tím přiřazení hlavičkových souborů modulu, místo C - souboru. S tímto přiřazením je hlášena chyba nepřítomnosti H – souboru při pokusu o následnou kompilaci. Přiřazují se celé adresáře s H – soubory. Pokud je adresář dále větven na podadresáře, je nutno přiřadit i tyto podadresáře, Tessy 2.9 neumí tyto podadresáře prohledat a opět by nebylo možné soubor zkompilovat.



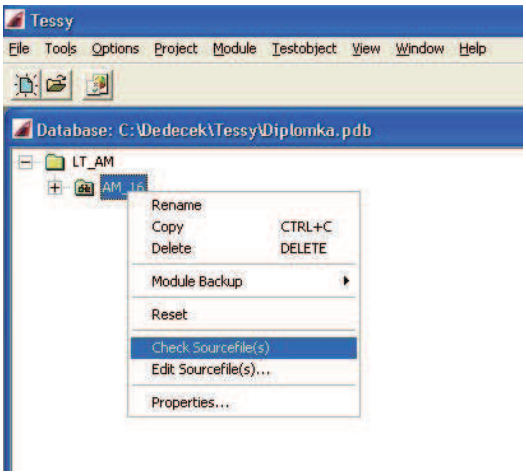
Obr. 20 Přiřazení H - souboru

Na Obr. 21 je vidět správné přiřazení hlavičkových souborů testovanému souboru. Pokud by adresář Tessy_includes obsahoval další podadresář, byla by vidět cesta k tomuto podadresáři.



Obr. 21 Správné přidání H – souboru

Další nastavení již nejsou nezbytná pro správné spuštění testovaného souboru. V praxi je možno se setkat s nastavením v define – jsou pevně daná od zákazníka a většinou charakterizují projekt, nelze je měnit.



Obr. 22 Kontrola souboru

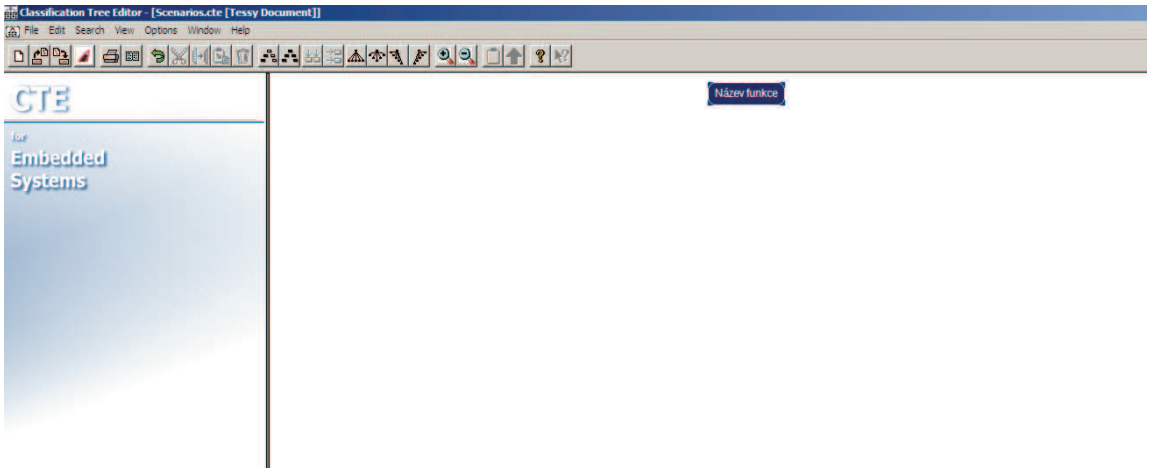
Po zvládnutí všech uvedených kroků, už stačí pouze zkontrolovat přiložené soubory. Pokud neobsahují chyby a jsou přeložitelné, nechybí žádné hlavičkové soubory, překlad proběhne bez problémů. Nyní již mohu otevřít modul a pokračovat s nastavováním testcase v CTE, kterému se věnuji v následující podkapitole. Pojem testcase by se dal charakterizovat jako testovací krok.

4.4 Využití CTE pro unit testování

Pro testování pokrytí kódu, tedy unit testování se používá aplikace CTE (Classification Tree Editor). Tento editor je plně integrován v Tessy 2.9, ale i v nižších verzích, se kterými jsem pracoval (Tessy 2.6). Editor slouží k vytváření struktur testování pro jednotlivé funkce. Pro potřeby funkčního testování jsem tento editor nepoužíval. Při testování pokrytí kódu jsem se s ním setkal, a jelikož společnost MBtech Bohemia s.r.o. provádí unit testování je práce s CTE neodmyslitelnou činností. Myslím, že by se tento editor dal využít i pro funkční testování, ale jelikož není u funkčního testování důležité testovat pokrytí kódu, přišlo mi jeho použití zbytečné. Konzultoval jsem tuto věc i se zaměstnanci, kteří mají s testováním SW větší zkušenosti a tuto mojí domněnku mi potvrdili. Přesto bych zde rád ukázal, jak se s daným editorem pracuje, jelikož to je velmi mocný nástroj programu Tessy a pokud se s tímto SW někdy dostanete do styku, určitě CTE využijete.

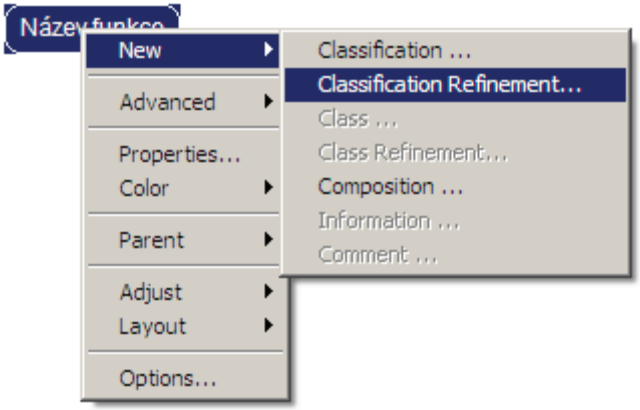
4.4.1 Spuštění CTE a vytvoření testcase

CTE se musí spouštět pro každou jednotlivou funkci zvlášť, jelikož strukturou modulového kódu jsou funkce odděleny. Spuštění se provede po vyvolání menu na funkci a zadáním příkazu Define Testcase, nebo pomocí ikony  . Po spuštění se zobrazí úvodní obrazovka editoru, jako je vidět na Obr. 23.



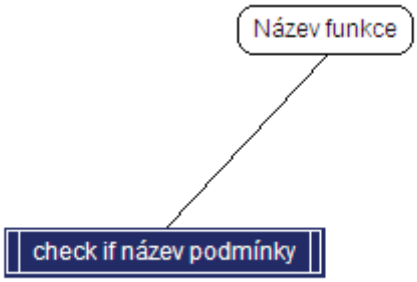
Obr. 23 Obrázek CTE po spuštění

Po spuštění CTE je vyplněný název funkce a mohou se přidávat jednotlivé části do struktury. Jako první se vytvoří classification refinement podle podmínek v kódu. Počet classification refinement v první struktuře je dán složitostí podmínky a také stylem a zkušenostmi testera. Záleží pouze na něm, jestli bude pro každou podmínku volit nový classification refinement v první vrstvě struktury, nebo bude tvořit strom, který je podobný control flow z Understandu.



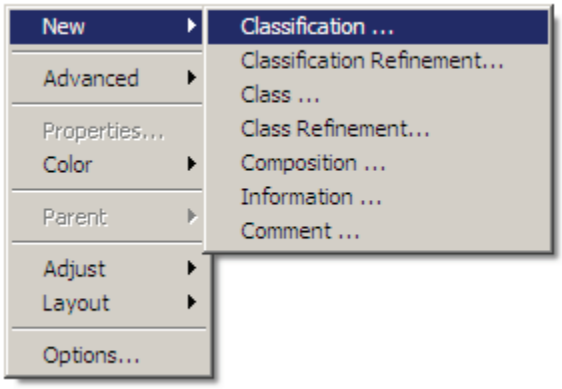
Obr. 24 Vytvoření classification refinement

Jméno struktury se zadává podle dané funkce. Pokud se jedná o jednoduchou funkci tipu inicializace, vrácení hodnoty (tyto funkce mají pokrytí kódu 100% pouhým spuštěním) popisují se tyto funkce jako *init*, *return*, *void*. Pokud to je již složitější funkce – příkladem jsou podmínky *if*, popisují se jako *check if* a podmínka testování. *Switch* se definuje jako *Switch (název switche)*. Názvy a popis si volí tester sám a je pouze na jeho uvážení, jak vše popíše. Důležitý je systematický postup a dodržování popisu v celém projektu. Pro přehlednost popisu je vhodné volit kratší popisy, ale jak jsem se sám přesvědčil, v některých případech to je velmi složité (pokud podmínka if obsahuje tři a více testovaných podmínek).



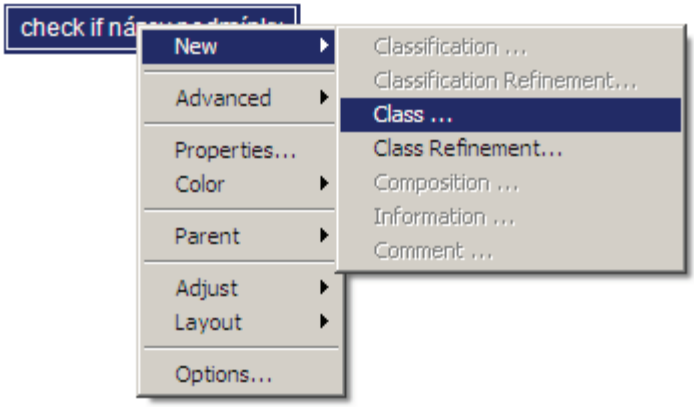
Obr. 25 Vytvořený clasification refinement

V classification refinement se postoupí o úroveň níže a vytvoří se classification, která má stejné jméno. Classification refinement slouží pouze jako přechod mezi úrovněmi. Pro přechod na úroveň výše je vhodné používat ikonu  .



Obr. 26 Vytvoření classification

Po vytvoření classification se přiřadí určitý počet class, aby se splnily veškeré stavy funkce. Důležité je mít označen příslušný classification pro který se vytváří class. Pokud tomu tak není, nevytvoří se mezi těmito prvky vazba.



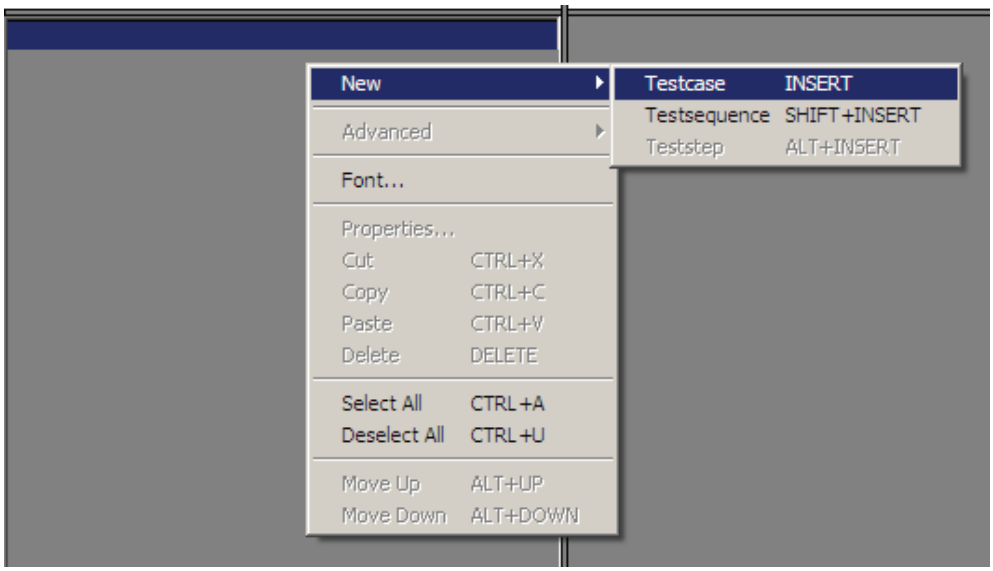
Obr. 27 Vytvoření class

Počet class záleží na typu funkce. Funkce jako inicializace, nebo return, bude mít pouze jeden class. U podmínky if budou dva class, jeden pro splnění podmínky, druhý pro nesplnění podmínky. Pokud má funkce formu switch, počet class bude záležet na počtu jednotlivých stavů switch. Zde je častá chyba nezapočítání default stavu. Občas tento stav chybí i v samotném kódu a to je již bráno jako chyba, kterou musí SW tester uvést v dokumentu o nalezených chybách (Error list, Error report).

Obr. 28 Vytvořený strom pro podmínku if

Po dotvoření grafického stromu a propojení vazeb následuje vytvoření testcase – testovacích kroků. Název testcase se volí stejný, jako je název funkce. Počet kroků je dán počtem větví funkce, ale existují případy, kdy se dá i velmi složitá funkce vyřešit jedním testcase. Tento případ může nastat, pokud se ve funkci objevuje cyklus for a všechny následující podmínky jsou uvnitř cyklu. Vhodným nastavováním parametrů se

dá docílit otestování všech stavů podmínek v jednotlivých průchodech cyklu. U jednodušších funkcí je odhadnutí počtu testcase velmi jednoduché. Inicializační funkce bude mít jeden, podmínka if bude mít dva testovací kroky. Pokud je funkce složitější, obsahuje například dvě vnořené if podmínky, musí být počet testcase takový, aby obsáhl všechny možnosti funkce, a tím bylo docíleno 100% pokrytí kódu (u tohoto případu stačí tři testovací kroky). SW tester by měl odhadnout nejmenší počet kroků, ale v některých případech se na počet smyček přijde až při samotném testování. Pokud dojde k situaci, že je nedostatečný počet testcase, je možné se vrátit do CTE a potřebný počet vytvořit (velkým pomocníkem je v tomto ohledu program Understand a jeho možnost zobrazení control flow funkce, jak jsem popisoval v kapitole tři).



Obr. 29 Vytvoření testcase

Každému vytvořenému testcase se přiřadí class, který je potřeba otestovat, aby byl v určité smyčce splněn. Každá class musí mít alespoň jedno připojení na testcase. Pokud nemá žádné přiřazení, následné vyhodnocení testu by zaznamenalo chybovou zprávu. Class, který nemá být v daném Testcase vyhodnocen se nepropojuje.

Obr. 30 Propojení class a testcase

Na Obr. 30 je možno vidět označené testcase, první class se provede v prvním kroku a druhý class v kroku druhém. Typické propojení podmínky if pro hodnoty TRUE a FALSE. Chování Testcase je vhodné popsat v description. Pokud obsahuje

testovaná funkce komentáře, je možné tyto popisky použít. Po okomentování a popsání jednotlivých testcase je nutné CTE uložit pomocí ikony  a následně vložit do Tessy ikonou  . Obě tyto ikony se nachází v panelu pro rychlý přístup spouštění funkcí.

4.5 Nastavení hodnot do proměnných

Po definování testovacích smyček už zbývá poslední krok před spuštěním testu a tím je zadávání hodnot. U unit testování nezáleží na přesných hodnotách vstupních a výstupních veličin. Hodnoty proměnných se nastavují tak, aby se docílilo průchodu všech větví funkce a pokrytí kódu dosáhlo hodnot 100%. Tyto hodnoty proměnných se mohou od použití funkce v praxi značně lišit. Pro funkční testování, kterým se tato diplomová práce zabývá, je podstatná specifikace od výrobce. Funkčnost zařízení by se také dalo definovat jako chování funkce při běžných podmínkách provozu při příchodu vstupních stimulů. Při komponent testu se již vyhodnocují konkrétní vstupní údaje, které už nejsou pouze smyšlené hodnoty, ale reálný ukazatel zařízení. Pokud funkce správně vyhodnotí příchozí proměnné a na výstupu se objeví požadované hodnoty, dal by se produkt označit jako otestovaný a vyhovující. Vstupní a výstupní hodnoty jsou zadány ve zmíněné specifikaci dodané zákazníkem. Tento test je vytvořen pomocí SCE (scenario editor). Práci v SCE a testování vstupních proměnných se věnuji v následující kapitole, kde popisuji průběh testování konkrétních dat a souborů.

5 Testování zdrojových kódů

V této kapitole bude popsána moje práce na testování reálných zdrojových kódů, které mi poskytla společnost MBtech Bohemia s.r.o. Zdrojové kódy musely být podrobeny cenzuře, jelikož nejsem zaměstnancem společnosti a hrozí zde nebezpečí úniku citlivých informací o zákaznících. Kódy byly upraveny Ing. Markem Bogou Ph.D a Ing Tomášem Síčem.

Na úvod je dobré vysvětlit, jaký je postup od zadání testovací úlohy až po dodání výsledků testů zákazníkovi. Požadavek na testování přichází od zákazníka, který potřebuje otestovat určitý software. Základní otázka je, proč vůbec potřebuje software otestovat? Dnes je kladen vysoký požadavek na bezpečné a funkční aplikace, jejichž bezpečnost musí být podložena příslušnými testy. Po provedení testů dostane otestovaný produkt doklad o vyhovujících vlastnostech, aby při případné reklamaci nemohla být firma napadena. Pokud by neotestovaný výrobek způsobil majiteli škodu, výrobní společnosti by mohli být pokutovány, což by mohlo mít negativní vliv na jejich další fungování. Testování výrobků je důležité i z marketingového hlediska. Pokud má firma vysoký počet reklamací, odradí si tím zákazníky a její výrobky se nebudou prodávat. Další hledisko kromě vysokých pokut, je bezpečnost. Riziko ohrožení zdraví, nebo dokonce ztráty lidského života je nejdůležitější z hlediska návrhu a funkce zařízení. Žádný výrobek nesmí běžným provozem ohrozit život a zdraví obsluhy. Pokud dojde k poruše, testy musí prokázat, že ohrožení zdraví a života je sníženo na minimum. Významným hlediskem je i plnění norem a předpisů, které jsou požadovány legislativou dané země, nebo společnosti. Výrobek se i lépe dostává na trh, pokud má v dokumentaci uvedeno, že prošel uznávanými testy. Všechny tyto uvedené argumenty nutí společnosti dávat své výrobky k testování.

Zadavatel, který chce výrobky otestovat, musí testovací společnosti dodat veškerou dokumentaci, plný zdrojový kód se všemi přidruženými soubory (častá je absence hlavičkových souborů) a také potřebné kompilátory, aby se mohl kód otestovat přímo pro procesor, na kterém bude pracovat. Společnost testující SW většinou pouze provádí testy. Nevyvíjí vlastní SW, a proto nevlastní licence na specializované SW produkty. Tyto licence musí dodat zákazník s požadovaným testem.

5.1 Předprojektová příprava

Před založením databáze a začátku práce na testování je důležité vědět, jak se pracuje na projektu v rámci reálné společnosti. Rozhodl jsem se stručně popsat postup, který jsem absolvoval před samotným testováním, které je stěžejním tématem diplomové práce. Nejprve jsem spolupracoval s oddělením IT, kde jsem dostal vlastní přihlašovací údaje k pracovní stanici, osobní email společnosti a přístup k centrálnímu serveru SVN. Ve společnosti MBtech Bohemia s.r.o. je kladen vysoký důraz na bezpečnost, a tomu odpovídá i volba hesel a jejich častá změna. Myslím, že podobný přístup funguje ve většině firem, kde si cenní své informace, a proto jsem rád, že nebudu překvapen, pokud se s tímto přístupem v budoucnu setkám.

5.1.1 SVN server

SVN server by se dal popsat jako server, který mapuje verze projektů a hlídá změny v uložených souborech. Bylo mi vysvětleno, kam se projekty nahrávají a kde si je mohu stahovat. Je to velmi cenné, pokud na projektu pracuje více lidí, je zde tedy možnost práce týmu na jednom velkém projektu. Mohou se dělat úpravy souborů, aniž by hrozilo přepsání, nebo ztráta důležitých dat projektu. Po změně souborů stačí pouze nahrát jejich aktualizované verze a každý člen týmu ihned vidí, které soubory byly upraveny. Mezi další výhody SVN serveru patří snadné zálohování a s tím související historie práce na projektu. To je velice výhodné při konečném hodnocení práce. Pomocí historie změn je okamžitě vidět na kterém souboru se pracovalo nejdéle, nebo který činil problémy a do budoucna je možno učinit opatření a tím zvýšit efektivitu práce.

Nebudu zde popisovat, co vše SVN server dokáže, jen bych se chtěl zmínit, jaký je přístup k aktualizování dat. Aby nedocházelo k chybnému ukládání dat na SVN server, existuje takzvaná paralelní modifikace, která využívá „merge“ a „lock“ stavy. Stav „merge“ dovoluje editovat jeden soubor více uživateli a možný konflikt úprav se řeší až při nahrání souboru na server tzv. „commit“. Pokud chce vývojář pracovat na souboru pouze sám, použije stav „lock“. Tímto se vyhne případným konfliktům s jinými vývojáři při aktualizaci souboru, jelikož ostatní pracovníci mají soubor zamknutý. Po dokončení práce odemkne soubor a příkazem „commit“ aktualizuje data. Se souborem muže nyní pracovat jiný člen týmu a uvidí změny, které byly vytvořeny při zamknutí.

Pro případnou spolupráci na dalších projektech jsem rád, že mi byla ukázána práce s SVN serverem. V rámci mého testování pro potřeby diplomové práce, jsem

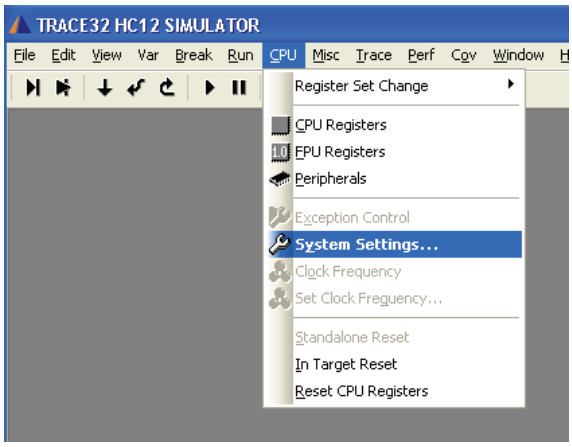
pracoval samostatně, a proto jsem využíval soubory pouze na lokálním disku mé pracovní stanice ve společnosti MBtech Bohemia s.r.o. v pražské pobočce. SVN server jsem využíval pouze jako zálohu originálních a neupravených souborů.

5.2 Příprava testovací stanice

Software Tessy 2.9 a program Understand byl již nainstalován na mé pracovní stanici. Kompilátor KEIL, který byl potřeba pro ladění tří modulů, které jsem dostal k otestování, byl volně ke stažení pouze s omezenou funkčností. Z výše uvedeného důvodu neměla společnost MBtech Bohemia s.r.o. potřebnou licenci na kompilátor KEIL v plné verzi. Tuto situaci jsme vyřešili pomocí časově omezené licence pro studijní účely. Tím byly vyřešeny veškeré administrativní procesy a testování modulů mohlo začít.

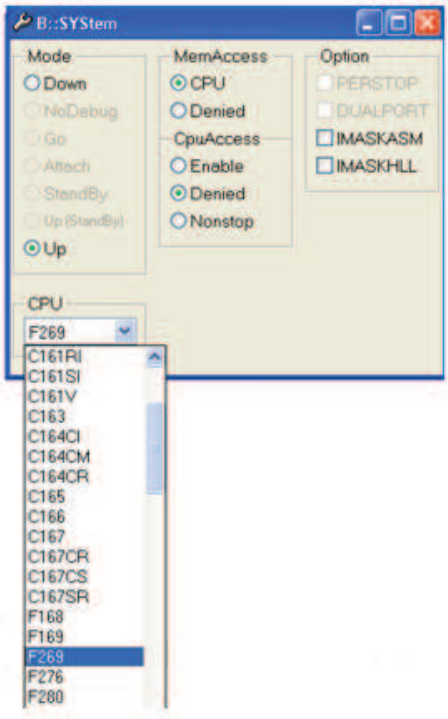
5.3 Nastavení procesoru v simulátoru Trace32

Aby bylo možné funkčně otestovat zdrojové kódy implementované do řídicí jednotky, je potřeba mít zapnutý program Trace32, kde se nastavuje procesor, který je uveden ve specifikaci testování. Program Trace32 slouží k simulování procesorů bez jakéhokoliv hardwaru. Práce procesoru je simulována softwarem. To z tohoto programu činí velmi mocný ladicí nástroj, pomocí kterého lze číst a zapisovat do paměti, číst a zapisovat do registrů procesoru, vytvářet HW breakpointy. Těmito funkcemi nahrazuje ladící systémy, které jsou implementovány přímo v procesoru, jako je BDM rozhraní od společnosti Motorola, JTAG pro ARM7 a JTEG pro PowerPC. Vzhledem k tomu, že není potřeba fyzický HW, je možno pomocí Trace32 požadovaný HW nejprve vyzkoušet simulací a až poté zakoupit vhodný procesor. Výběr typu procesoru je vidět na následujících obrázcích.



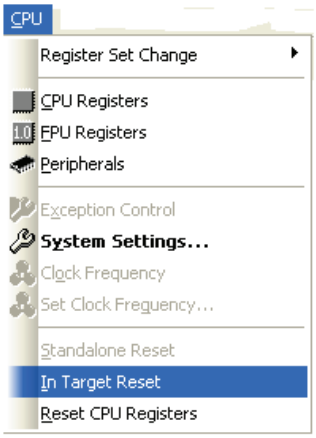
Obr. 31 Výběr nastavení procesoru

V menu CPU v položce system settings jsem nastavil definovaný typ procesoru. Nastavení, je vidět na Obr. 32. Testovaný modul využíval procesor ST10F269. Na Obr. 32 je vidět výběr procesorů, pro které se používá pouze několik posledních znaků. Označení záleží na typu a zavedených standardech.



Obr. 32 Výběr procesoru

Po provedení veškerých nastavení je vhodné resetovat registry a inicializovat nastavení zvoleného typu procesoru. Oba tyto příkazy jsou v menu CPU. 0 zobrazuje umístění těchto příkazů.



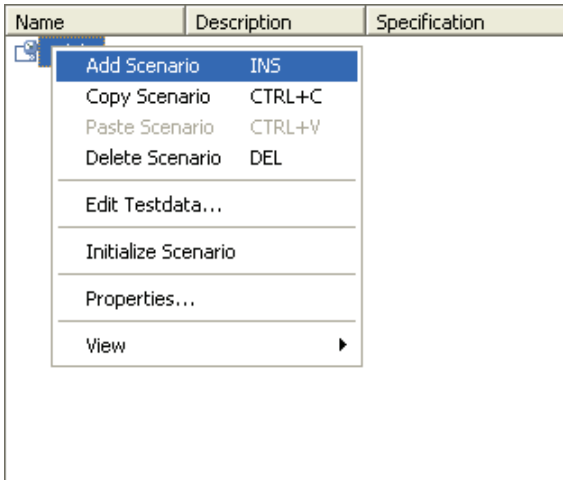
Obr. 33 Připravení procesoru

5.4 Testování modulů AM_14, AM_16, AM_17

Po nastavení kompilátoru a správné inicializaci procesoru jsem spustil program Tessy 2.9. Pro ověření mého konceptu testovací specifikace jsem dostal k reálnému otestování tři moduly. V následující části popíši přesný postup testů i s vyhodnocením výsledků. Prvním krokem je vytvoření databáze, popis uvádím v kapitole 4, a proto ho zde nebudu znovu opakovat. Po vytvoření databáze a spustitelného modulu, tj. přeložení přiložených souborů bez chyb, jsem začal pracovat na testovacích krocích.

5.4.1 Vytvoření testovacích kroků

Každému testovanému sloupci v tabulce, kterou získám ze specifikace testování SW (zmíněné tabulky jsou k nahlédnutí v příloze), jsem vytvořil jedno scenario podle návodu, který uvedu na následujících stranách. Na Obr. 34 uvádím přidání nového scenaria. Tento výběr je k dispozici v pravé části pracovní plochy v SW Tessy. Nejjednodušší vytvoření je přes klávesovou zkratku insert.



Obr. 34 Vytvoření scenaria

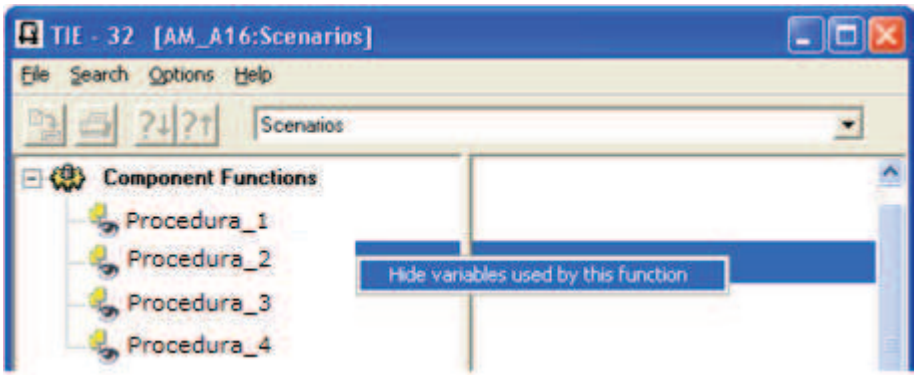
Ihned po vytvoření testovacího kroku je důležité pojmenovat scenario podle testovacích specifikací, které jsem provedl podle dokumentace. Toto pojmenování je důležité pro orientaci, jelikož testů bylo velké množství a pokud bych si testy pojmenoval pouze jako Test1, Test2…po chvíli bych nevěděl, co jsem již testoval a co ještě musím otestovat. Jméno jsem volil podle testované procedury. V description (popis) jsem určil pořadí sloupce v tabulce a ve specifikaci jsem vypsal požadavky, na které bylo testování zaměřeno. Příklad zvoleného popisu testovaných funkcí je ukázán na Obr. 35. Z důvodu obchodního tajemství a autorských práv jsem byl nucen popisky

v Obr. 35 upravit, abych neporušil výše zmíněné předpisy. Pokud se objeví chyba v testovaném scenariu, okamžitě vím, v které tabulce a v jakém sloupci chyba nastala a můžu řešit její odstranění. Odpadá dlouhé dohledávání chyb při špatném popisu testovaných procedur.

Name	Description	Specification
1(1) Test procedury_1	Sloupec 1	Specifikace procedury_1
2(1) Test procedury_1	Sloupec 2	Specifikace procedury_1
3(1) Test procedury_1	Sloupec 3	Specifikace procedury_1

Obr. 35 Pojmenování a popis testů

Po vytvoření jednotlivých testů pro sloupce tabulky – testoval jsem vždy jednu celou tabulku najednou – následuje otevření TIE a upravení vstupů a výstupů proměnných podle specifikace z tabulky. Jestliže jsou proměnné u procedury zašedlé – nelze s nimi provádět žádnou operaci a musí se aktivovat. Zde se používají výrazy „schování“ a „ukázání“ proměnných procedury. Stačí pouze pravým tlačítkem myši stisknout název procedury a podle aktuálního stavu, se rozbalí menu s příkazem, jak ukazuje Obr. 36 a Obr. 37.



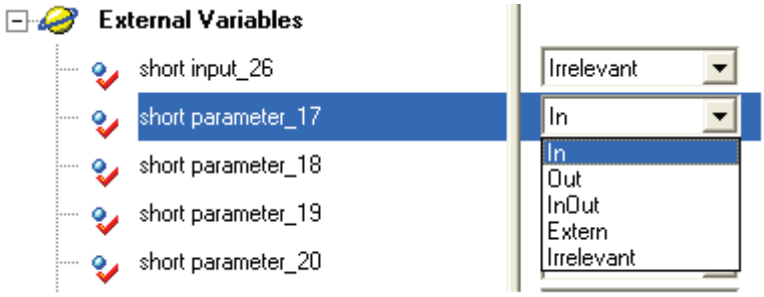
Obr. 36 Skrytí proměnných funkce

Rychlá kontrola viditelnosti proměnných je ikona oka u každé procedury. Pokud jsou proměnné neaktivní, oko je překryté červenou čarou.

Obr. 37 Odkrytí proměnných funkce

Po zviditelnění proměnných jsem přistoupil k nastavení propustnosti dat. Určoval jsem, které proměnné se budou chovat jako vstupní a které jako výstupní.

Na Obr. 38 ukazuji možnosti nastavení parametrů proměnné. Na výběr jsem měl z pěti možností: vstup, výstup, vstupní i výstupní, externí a bezvýznamný. Zde jsem podle tabulky, která se nachází v dokumentu specifikace testování SW modulu AM_16, nastavoval požadované parametry proměnných.

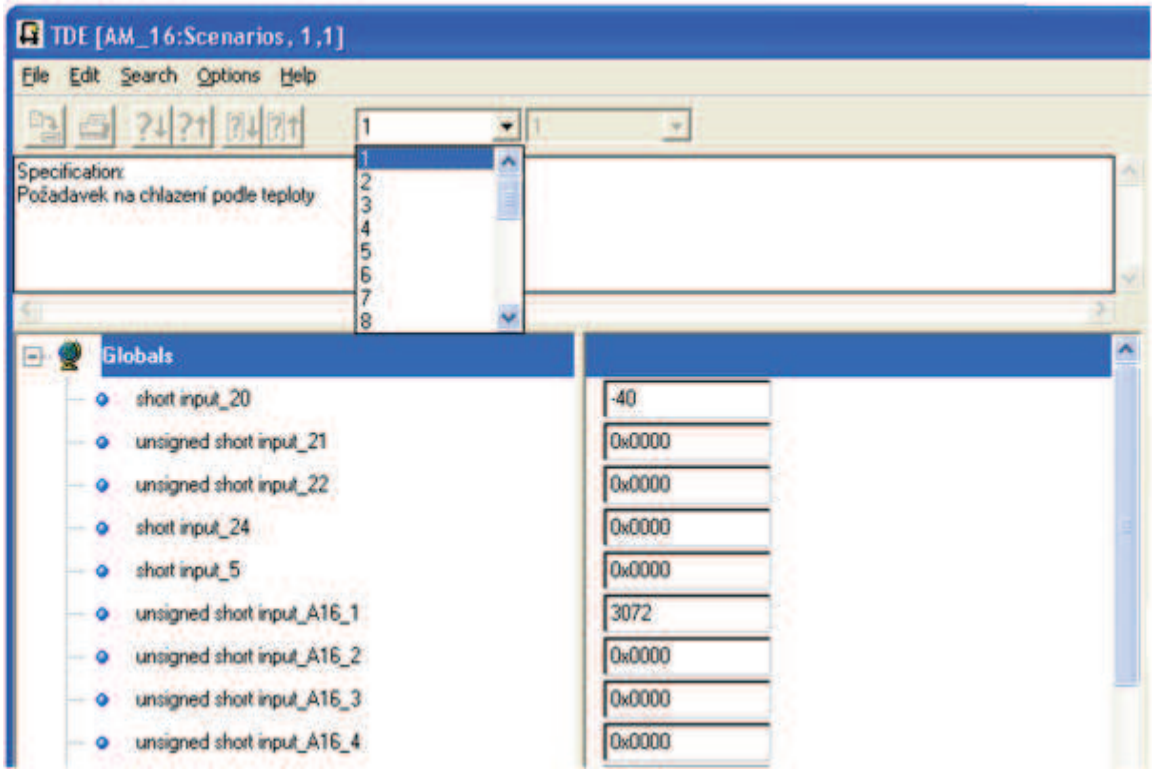


Obr. 38 Možnosti nastavení parametrů proměnné

Při vkládání parametrů proměnných jsem objevil velice důležitou vlastnost Tessy, která negativně ovlivnila celý průběh testování. Mám na mysli nemožnost pracovat s proměnnými, které jsou definovány jako pointer na pole funkcí. Jelikož jsou pointery velmi používané při programování v jazyku C, setkal jsem se s tímto problémem na více místech. Kdekoliv se pointer na pole funkcí vyskytl, program Tessy nemohl zobrazit proměnnou, nedalo se s ní pracovat a ani ji řádně otestovat. Problém s pointerem na pole funkcí, není jediný, který SW Tessy obsahuje. Další potíže má se zobrazením pointeru na pointer a pointer na funkci. Tuto skutečnost jsem na několika místech vyřešil úpravou kódu, což je ale nestandartní postup. Softwarový tester má na chyby v SW pouze upozornit a neopravovat je. Úpravy nebyly nijak rozsáhlé, na funkčnost kódu neměli vliv, sloužili pouze pro spuštění testů. Například v Tab. 5, kterou můžete vidět v příloze tři, jsem nemohl otestovat proměnné input_18 a input_19. Při bližším zkoumání funkce těchto proměnných jsem zjistil, že zastupují logické hradlo OR. Při výskytu jednoho bitu v logické 1 je dovolen další postup ve funkci. Pokud tomu tak není, funkce se nevykoná. Při zjištění této informace jsem i takto upravil testy. Všechny tyto proměnné byly typu boolean, výsledné hodnoty vracely logickou hodnotu jedna, nebo logickou hodnotu nula. Tabulky, které jsem měl k dispozici ve specifikaci testování SW modulů, uvádím v přílohách dvě až čtyři. Z důvodů utajení interních informací a autorských práv nemohu uvést celou specifikaci, proto uvádím pouze tabulky s upravenými názvy proměnných.

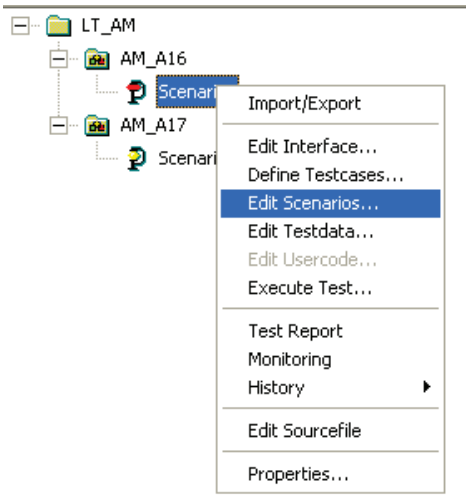
V dalším kroku testování jsem přistoupil k zadávání parametrů proměnných do TDE. Jak jsem uvedl výše, každý sloupec má své testovací scenario. Editor jsem

pustil „dvojklikem“ na scenario, které jsem upravoval. Okamžitě se spustí TDE, kde jsem zadával hodnoty z tabulky pro všechny proměnné. Pokud některá proměnná nebyla zobrazena, nejprve jsem překontroloval TIE, jestli je požadovaná proměnná povolena. Jestliže se proměnná nevyskytovala ani v TIE a všechny funkce modulu byly povoleny, byla proměnná typu pointer na pole funkcí. Jak jsem již zmiňoval výše, tuto proměnnou nebylo možno nastavit.



Obr. 39 Zadávání hodnot do TDE

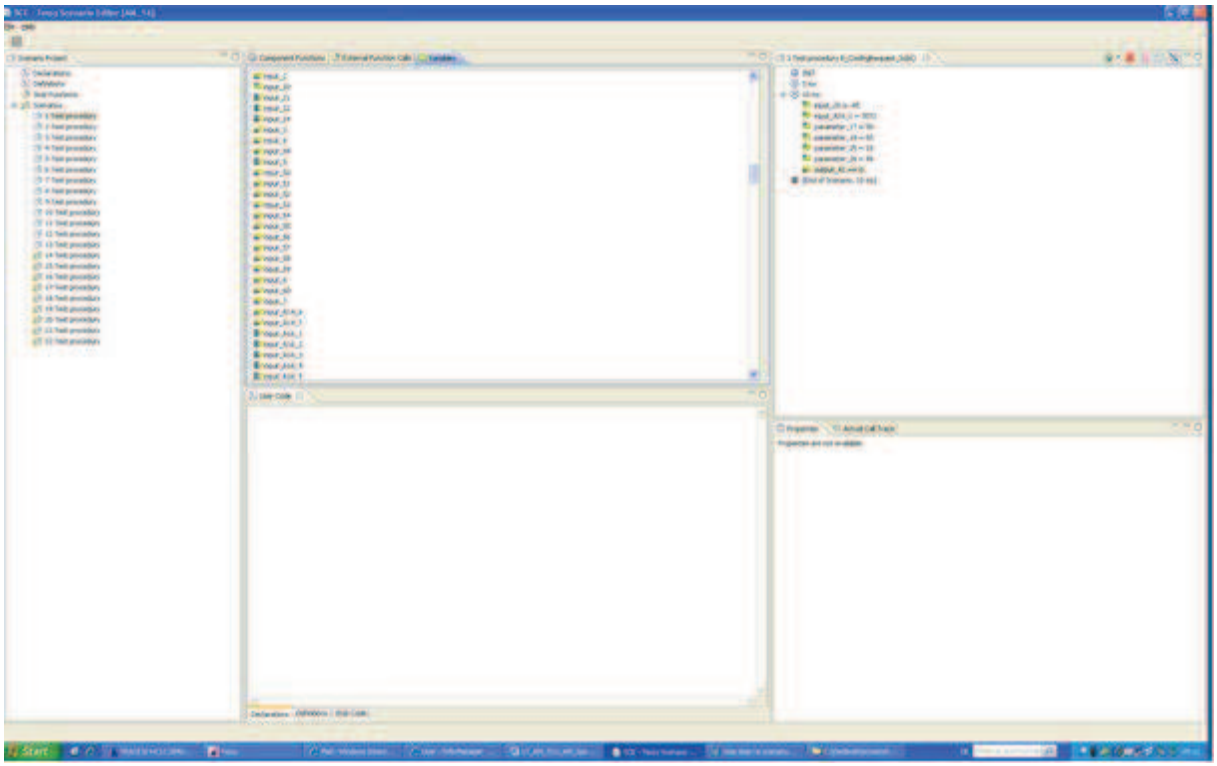
Po nastavení veškerých zadaných parametrů proměnných v TDE jsem uložil vykonané změny. Ukončení práce bez uložení by se neprojevilo v testu. Mezi testovacími kroky lze jednoduše přepínat v TDE a nemusí se proto spouštět pro každý krok zvlášť. Ukázka zadaných proměnných a přepínání mezi kroky je názorně předvedena na Obr. 39. Následně jsem již přistoupil k ověřovací fázi v SCE. 0 ukazuje menu, přes které jsem k tomuto nastavení přistupoval.



Obr. 40 Spuštění editace scénária

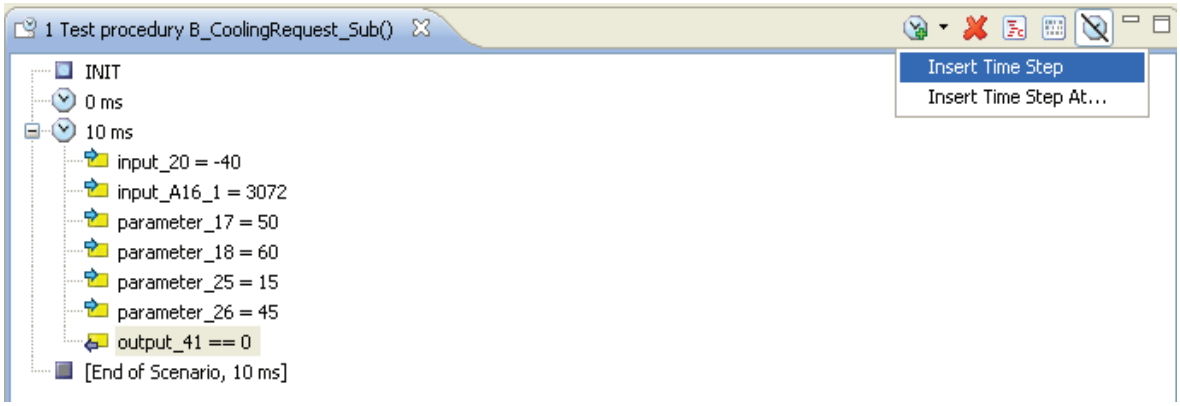
5.4.2 Scenario editor

Po spuštění SCE je pohled na 3 sekce, které jsou od sebe vzájemně odděleny. V levé části je struktura projektu, vytvořené testy a stub functions. Stub functions jsou funkce, u kterých může tester ovlivňovat jejich výstupní hodnoty. Tyto funkce využívají vnitřní proměnné Tessy. Tester má možnost pomocí těchto funkcí nastavit hodnoty, které potřebuje pro splnění, případně nesplnění podmínky, která s danou funkcí pracuje, ale není součástí právě testované funkce. Možnost ovlivňovat veškeré funkce, které jsou součástí produktu, má mít funkční testování. Z tohoto důvodu jsou stub functions nepoužité. Středová část zobrazuje funkce a procedury komponenty, v další záložce jsou externí funkce, třetí záložka řadí jednotlivé proměnné. V této části editoru je nutné nastavit takzvanou „heartbeat“ funkci. Je to funkce, kterou podrobujeme testu a která se volá v každém časovém intervalu testu. Pro každý z jednotlivých modulů AM_14, AM_16 a AM_17 jsem jednu z funkcí testoval více než ostatní, a proto jsem ji označil jako „heartbeat“. Je možné i požadovanou funkci vložit do každé časové smyčky, ale při velkém počtu testů, je to značně pracné. Tuto možnost vložení je vhodné využít pro testování dalších funkcí v jednom testu, jelikož označení „heartbeat“ funkce může mít pouze jediná. Pokud bych chtěl přesto otestovat i jinou funkci, než právě označenou a nechtěl bych ji vkládat do jednotlivých kroků testu, musel bych vytvořit nový modul, nové scenario a tam označit jako „heartbeat“ funkci tu, kterou chci podrobit testování. Označení „heartbeat“ se provede ikonou, která je ve stejné úrovni, jako přepínání záložek a nese označení „Set as Work Task“.



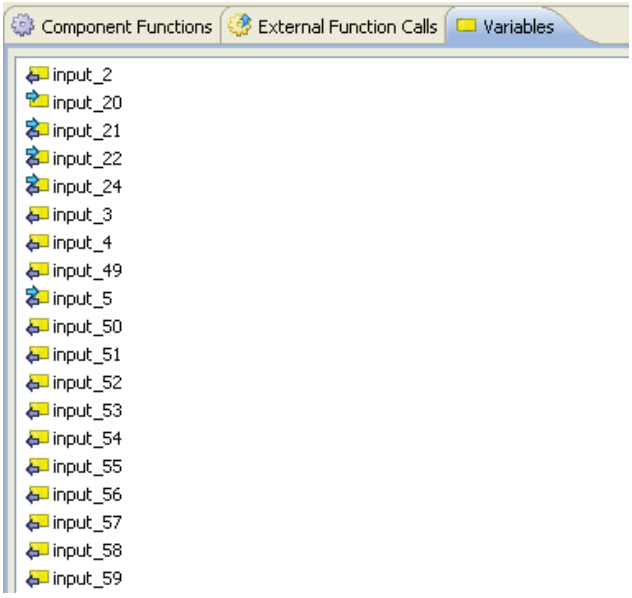
Obr. 41 Editor scenaria

Třetí sekce je z hlediska testování nejzásadnější, jelikož jsem zde přidával testovací časy, proměnné a nastavoval hodnoty jednotlivých proměnných. Z tohoto důvodu se na ni nejvíce zaměřím a vysvětlím postup jednotlivých činností. Nejdůležitější proceduru, kterou v této části editoru navrhuji, je vytváření časových kroků. Informaci o velikosti a průběhu časového kroku by měla být dána v testovací specifikaci pro každou funkci nebo model. V mém případě byl testovací krok 10 ms. Počet kroků jsem mohl vkládat ručně, nebo jsem mohl vložit až konečný čas, pokud jsem věděl, že se má některá smyčka vykonat několikrát za sebou. Poté se vytvořilo tolik časových oken, kolik se jich do uvedeného času stihlo provést. Můj postup vkládání proměnných a funkcí byl následující. Nejprve jsem do části INIT vložil inicializační proceduru, která nastavila parametry proměnných na výchozí nulovou hodnotu. Následně jsem vložil dvě časové smyčky, jelikož jsem chtěl nechat jeden časový rámec na inicializování proměnných a nastavení jejich hodnot na počáteční stav. Jelikož jsem měl nastavenou „heartbeat“ funkci na Proceduru_1 nebylo potřeba tuto funkci do časové smyčky vkládat.



Obr. 42 Vložení časového intervalu

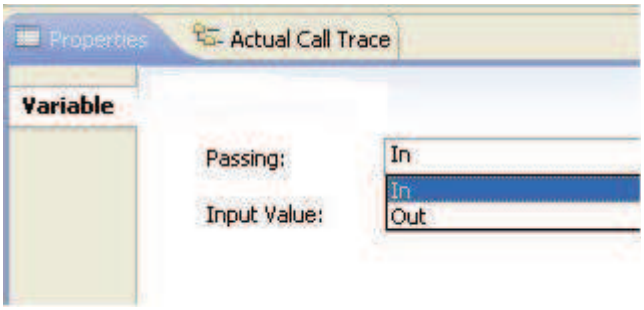
V tomto okamžiku jsem měl vše připraveno pro vkládání proměnných. Seznam proměnných, které jsem mohl vkládat, byl ve středové části pod záložkou „Variables“. U jména proměnné je šipkou znázorněno, jestli je vstupní, výstupní, nebo vstupní i výstupní. Toto nastavení zde nelze měnit, pokud bych zjistil, že požadovaná proměnná má jinou datovou propustnost, musel bych opustit SCE a nastavení provést v TIE, jak jsem popisoval výše. Část seznamu proměnných je vidět na Obr. 43.



Obr. 43 Seznam proměnných

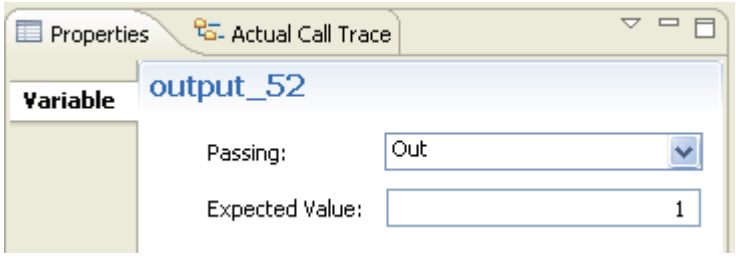
Přidání proměnné do časové smyčky jsem provedl pouhým chycením a přetáhnutím požadované veličiny. Nastavení dalších parametrů proměnných jsem provedl až v časových smyčkách. Z nastavitelných parametrů jsem konfiguroval

vstup/výstup, pokud byla proměnná nejednoznačně definována. Na tomto místě jsem také vkládal hodnoty z tabulek pro vstupní i výstupní proměnné.



Obr. 44 Volba parametrů proměnných

Po nastavení výstupu proměnné je hned vhodné zadat hodnotu, kterou jsem očekával podle údaje z tabulky. Tato možnost nastavení se zobrazila okamžitě po ukončení výběru průchodu proměnné.

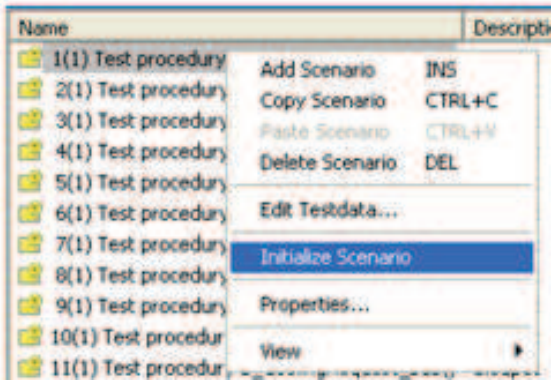


Obr. 45 Zadání hodnoty

Obr. 45 ukazuje, že výstupní proměnná output_52 očekává hodnotu 1. Zde je velmi důležité vědět, který sloupec tabulky se testuje, jelikož se na určité vstupní hodnoty očekávají určité výstupní hodnoty. Pokud bych zadal jinou hodnotu, než je očekávaná, test by se vyhodnotil jako chybný. Po nastavení všech požadovaných hodnot jsem uložil změny provedené v SCE a přešel k další části testování a tím je vyhodnocení testu.

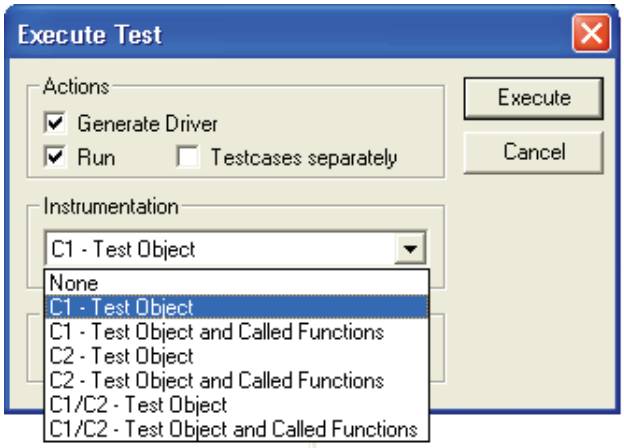
5.4.3 Spuštění a vyhodnocení testu

Závěrečná fáze testování je plně automatizovaná programem Tessy. Prvním krokem je inicializace scenaria, které se provede podle Obr. 46. Při vlastním testování se označí scenario, které je potřeba vyhodnotit (je možné označit i všechny zároveň). Po označení se vyvolá pravým tlačítkem myši menu a vybere možnost inicialize scenario. Tím jsou aktualizované změny, které byly provedeny po poslední inicializaci. Pokud to je první inicialize, ikona u jména testovaného subjektu dostane žlutou barvu.



Obr. 46 Inicializace scenaria

V této fázi byl subjekt plně připraven k testování. Spuštění samotného testu jsem provedl tlačítkem  . Jiná možnost spuštění testu je v menu project příkazem execute test. Po vyvolání tohoto příkazu Tessy zobrazí nabídku s výběrem druhu testu. Výběr je ukázán na Obr. 47.



Obr. 47 Výběr testu

Při prvním spuštění testu je důležité vygenerovat drivery, a proto musí být tato položka označena. Při testování jsem generoval drivery při každém startu. Zkoušel jsem test bez této položky, ale žádné znatelné zrychlení jsem nezaznamenal. U velkých projektů, nebo složitých funkcí má generování driveru značný vliv na rychlost vyhodnocení testu, a proto bych doporučil vytvořit driver pouze při prvním spuštění. V některých případech se musí drivery generovat neustále a to například při zásahu do stub functions, nebo změně v SCE. Pokud se pouze mění hodnoty proměnných v TDE, stačí pouze spustit vyhodnocení testu bez nutnosti vytvářet driver. Při průběhu testování je nutné mít spuštěný simulátor Trace32. Pokud jsou v něm přidané „breakpointy“ a je zatržena položka „define breakpoint“ při spuštění testu, celý proces

se na uvedeném místě zastaví. Pro pokračování v testu musí tester odkrokovat program v simulátoru.

Jak ukazuje Obr. 47 na výběr je z několika typů testů. Tessy podporuje obě dostupné varianty měření pokrytí kódu, které se označují C1 a C2. Vzhledem k různým významům těchto způsobů měření pokrytí bych oba pojmy vysvětlil. C1 testuje rozhodnutí a větve v podmínce, které se následně vyhodnotí a zobrazí se v podobě stromu pokrytí. Pod označením C2 se skrývá MCC (Multiple Conditions Coverage – vícestavové pokrytí) a MC/DC (Modified Condition/Decision Coverage – změna podmínky/rozhodnutí o pokrytí). MC/DC testuje určitou podmínku ve funkci na pravda/nepravda, nezáleží na výsledku ostatních podmínek. Při vnořených podmínkách existuje několik větví postupu kódem a vzniká velké množství kombinací. Tessy propočítává všechny tyto kombinace a hledá nejlepší výsledek pokrytí pro danou kombinaci. Velkou nevýhodou tohoto řešení, je velká prodleva výpočtu, a proto je počet kombinací omezen na osm možností v základním nastavení. Počet kombinací se může zvýšit, ale za cenu delšího výpočetního času. Pokud daná funkce bude příliš složitá a Tessy neobsáhne všechny možné kombinace, bude pracovat pouze s dostupnými informacemi. Pro lepší pochopení těchto označení je dobré si pamatovat, že testování pokrytí kódu pomocí C1 testuje větve a C2 testuje podmínky kódu. V menu pro spuštění testu máme na výběr z těchto možností:

1) None – žádné testování pokrytí kódu se neprovede
2) C1 - Test Object – provede testování pokrytí kódu pro větve testovaného objektu
3) C1 - Test Object and called Functions – test jako v předešlém případě a navíc se otestují všechny funkce, které jsou testovaným objektem volány
4) C2 - Test Object – proběhne test MCC a MC/DC pro vybraný objekt
5) C2 - Test Object and called Functions – test jako v bodě 4 navíc s otestováním všech dalších funkcí, které jsou testovaným objektem volány
6) C1/C2 Test Object – kombinace testů v bodech dva a čtyři
7) C1/C2 Test Object and called Functions – kombinace testů v bodech tři a pět

Jelikož se u funkčního testování neřeší pokrytí kódu, není důležité, který typ testu použiji. Na doporučení kolegů jsem veškeré testy prováděl pomocí C1 – Test Object. Po dokončení vyhodnocení testu může nastat několik variant, které shrnuji níže:

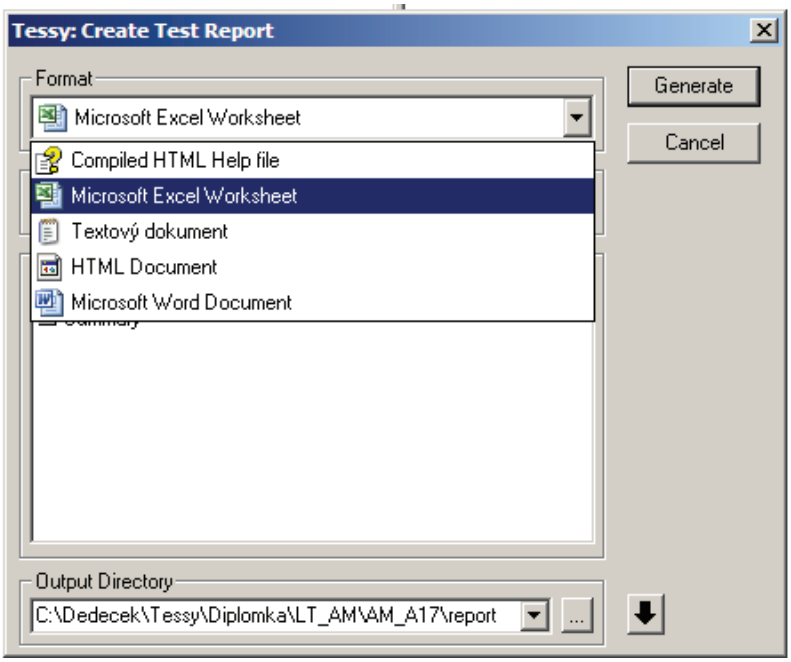
1) Správně vyhodnoceny výstupy v TDE i v SCE
2) Správně vyhodnoceny výstupy v TDE, chybně v SCE
3) Chybně vyhodnoceny v TDE, správně v SCE
4) Chybně vyhodnoceny v TDE i v SCE

Pokud vyjde chybný výsledek testu, je nutné zkontrolovat zadané parametry. Jestliže je vše v pořádku, musí se analyzovat zdrojový kód a určit příčinu vzniku chyb. Nalezené chyby a jejich možnou nápravu uvádí tester v error listu. Po dokončení testování se již jen vygeneruje test report. Generováním test reportu se věnuji v následující kapitole.

5.4.4 Výsledky testů

Poslední fází testování softwaru je generování výsledků zpráv, kde jsou přehledně zobrazeny odezvy na zadané hodnoty a vyhodnocení testů. Tyto zprávy z měření, neboli test reporty se tvoří opět pomocí programu Tessy a není tedy potřeba žádný dodatečný SW. Zprávy je možno generovat až po vyhodnocení testů. Nezáleží na jejich výsledcích i zpráva, která vykazuje chybně vyhodnocený test, který prošel řádným testováním a neobsahuje žádnou chybu ve vstupních datech, je cenný. Dalo by se říci, že chybné reporty jsou cennější, jelikož ukazují na chyby v testovaném softwaru. Pokud to jsou chyby závažnějšího charakteru, jejich odhalení je o to důležitější, jelikož nedojde jejich implementací do zařízení k poruše a možným následkům, ať na lidském zdraví, nebo vzniku materiálních škod. Zde je vidět opodstatněnost testů.

Nyní již přejdu k návodu, jak zmíněné zprávy z testování vytvořit. Pokud souhlasím s výsledky testů a nebudu již žádné proměnné upravovat, stačí pouze dvakrát poklepat na scenario v modulu, z kterého chci generovat výsledný dokument. Zobrazí se nastavení, které můžete vidět na Obr. 48.



Obr. 48 Vytvoření zprávy z testování

Zde si pouze vyberu požadovaný formát a jednu z přednastavených předloh. Pokud chci zvolit jiný než přednastavený adresář pro uložení výsledné zprávy, je tato možnost změny v řádku output directory. Posledním krokem je pouze zadání příkazu pro generování a Tessy požadované zprávy vytvoří. Výsledné zprávy z testování, které jsem prováděl, uvádím příloze. Pro modul AM_14 to je příloha 5, pro modul AM_16 příloha 6 a modul AM_17 zobrazuje příloha 7. Uvedené zprávy jsou zobrazeny ve standardním formátu, který je součástí Tessy. Společnost MBtech Bohemia s.r.o. má své vlastní formátování pro závěrečné zprávy, které předkládá svým zákazníkům. Je možno i pro konkrétního zákazníka danou zprávu dodatečně upravit, ale to již záleží pouze na dohodě. Zprávy uvedené v příloze slouží pouze pro představu a pro úplnost, jakého výsledku se testováním dosáhne.

6 Závěr

Cílem diplomové práce bylo vytvořit koncept pro funkční testování pomocí softwarové aplikace Tessy 2.9. Postupem času, jak jsem pronikal do problematiky testování hlouběji, jsem zjistil, že testy jsou pouze jednou z částí procesu ověřování funkčnosti SW aplikací. Nedílnou součástí jsou verifikační a validační postupy, které jsou definovány v normách, z nichž nejzásadnější v oblasti automobilového průmyslu je norma ISO 26262, která definuje a popisuje ASIL, jako jeden z klíčových prvků při funkční bezpečnosti. Testování je pouze klíčem k získání potřebných legislativních oprávnění, které jsou nezbytné při uvádění výrobků na trh.

Jedním z nejdůležitějších dokumentů při vytváření funkční aplikace je specifikace návrhu softwaru, kde se definují veškeré funkce, vstupní a výstupní proměnné, průběhy jednotlivých veličin. Tento dokument by měl mít SW tester k dispozici, aby lépe pochopil dané funkce a při testování věděl, co je bráno ještě jako standardní chování a co již uvádět jako chybu. Druhým důležitým dokumentem je specifikace testování softwaru, kde je popsáno, jak má být aplikace otestována. V dokumentu jsou uvedeny veškeré potřebné informace k plnému otestování všech funkcí, které se mohou u testovaného produktu vyskytnout. Dokument obsahuje tabulky vstupů a odpovídajících výstupů, grafy průběhů fyzikálních veličin. Bez těchto údajů nelze provést funkční testování. Oba dokumenty jsem měl k dispozici pro všechny tři testované moduly. Dokumenty byly upraveny společností MBtech Bohemia s.r.o., abych je mohl využívat bez zásahu do autorských práv a obchodního tajemství.

Práce SW testera nezahrnuje pouze čtení kódu a tedy dokonalou znalost testovaného programovacího jazyka, ale také administrativní prvky, jako jsou rozvržení časového harmonogramu pro celý průběh testování, vytváření dokumentů, kde popisuje nalezené chyby a případně navrhuje řešení na jejich odstranění. Dalším prvkem je komunikace se zákazníkem o dodržování termínů a řešení vzniklých problémů během testování.

Časový harmonogram je velmi důležitým prvkem v práci SW testera. Do tohoto dokumentu se zahrnují veškeré práce související se všemi fázemi testování. Prvním krokem je vytvoření databáze a projektu, přeložení a otevření modulů. Následuje testování a odstraňování chyb ze zdrojového kódu. Dalším krokem je vytvoření dokumentů s výsledky testů a dokument obsahující nalezené chyby. Posledním krokem je odeslání veškerých zpráv z průběhu testů zákazníkovi.

Harmonogram mého testování neobsahoval fáze komunikace se zákazníkem, a jelikož jsem při testování nenarazil na žádné chybné vyhodnocení testu, neuvádím dokument, který by popisoval nalezené chyby v kódu. Bez těchto fází jsem ale absolvoval celý časový harmonogram testování. Nejprve jsem vytvořil databázi - diplomka, v které jsem založil projekt LT_AM, v tomto projektu jsem testoval předložené moduly AM_14, AM_16 a AM_17. Zdrojové kódy musely být upraveny ze stejného důvodu, jako dokumenty specifikace návrhu SW a specifikace testování SW. Hlavní úpravy spočívaly v přepsání jmen proměnných, do smyslu funkcí nebyl učiněn žádný zásah. Po této fázi vytvoření databáze a projektu jsem začal provádět funkční testování.

Funkční test, v programu Tessy označován jako komponent test, je ověření správné reakce výstupních hodnot proměnných na zadané vstupní údaje. Hlavní rozdíl oproti unit testu vidím v pokrytí kódu. Pro unit test je hlavní cíl pokrýt testování kódu na 100% bez ohledu na výstupní hodnoty. Vstupní hodnoty slouží pouze k dosažení 100% pokrytí. Tester hledá takové vstupní hodnoty, aby v co nejnižším počtu testovacích smyček dosáhl úplného pokrytí kódu. U komponent testu se nehledí na pokrytí kódu a počet testovaných smyček, ale pouze se zkoumají vstupní a výstupní hodnoty. Z mého pohledu jako začínajícího testera mi přijde rozumnější zkoumat kódy funkčně, jelikož je výhodnější vidět reakci na vstupní parametry, než mít pouze ošetřeno, že se daná větev kódu aktivuje, pokud je stimulována určitým údajem, byť v reálném provozu nesmyslném. Moji domněnku začínají potvrzovat i vývojáři, jelikož zájem o funkční testování rapidně roste. Nemyslím si, že testování pokrytí kódu úplně vymizí, jen se bude provádět spolu s funkčním testováním.

Fázi testování jsem chtěl provádět pomocí CTE, jak je běžné u unit testovaní a pokrytí kódu. Po konzultaci s pracovníky MBtech Bohemia s r.o. jsme dospěli k závěru, že je tento proces zbytečný a k provedení komponent testu stačí využít SCE a TDE. Využití CTE je určitě možné a pokud bude zákazník vyžadovat propojení obou testů, bude výhoda této možnosti u komponent testu neocenitelná. Z tohoto důvodu jsem také věnoval podkapitolu vytváření testcase v CTE, aby při pozdějším využití této funkce byla diplomová práce komplexním návodem pro tvorbu komponent testu.

Závěrečnou fází testování je vytváření zpráv z průběhu testů. Zprávy z testování uvádím k nahlédnutí v příloze pět pro modul AM_14, v příloze šest pro modul AM_16 a v příloze sedm pro modul AM_17.

Testování modulů probíhalo na aktuální verzi Tessy 2.9, jelikož v nižší verzi a to 2.6, kterou má společnost MBtech Bohemia s.r.o. k dispozici, není možné provádět komponent test. Společnost Razorcat přišla s tímto druhem testování až v této verzi, bylo to dáno konkurenčním bojem, protože jiné testovací aplikace tento test již obsahovali a společnost Razorcat si nemohla dovolit ztratit místo na trhu. Dalším důvodem zavedení funkčního testování je poptávka zákazníků, kteří již nechtějí testovat pouze pokrytí kódu, ale i funkce aplikace.

Diplomová práce mi dala pohled na průběh testování SW aplikací. Jelikož jsem její realizaci prováděl za plného provozu ve společnosti MBtech Bohemia s.r.o., udělal jsem si představu, jak takový proces funguje na konkrétních příkladech. Získal jsem cenné zkušenosti z mezinárodní společnosti a ty mohu využít pro následnou praxi v oboru softwarové verifikace.

7 Použitá literatura

[1] PATTON, Ron. *Testování softwaru*. Vyd. 1. Praha: Computer Press, 2002, 313 s. Programování. ISBN 80-722-6636-5.

[2] STEPHENS, Matt a Doug ROSENBERG. *Testování softwaru řízené návrhem*. Vyd. 1. Brno: Computer Press, 2011, 336 s. ISBN 978-80-251-3607-2.

[3] HARPER, Allen, Shon HARRIS, Chris EAGLE, Jonathan NESS a Michael LESTER. *Hacking*: *manuál hackera*. 1. vyd. Praha: Grada, 2008, 399 s. ISBN 978-80-247-1346-5.

[4] EELES, Peter a Peter CRIPPS. *Architektura softwaru*. Vyd. 1. Brno: Computer Press, 2011, 328 s. ISBN 978-80-251-3036-0.

[5] HUNT, Andrew a David THOMAS. *Programátor pragmatik*: *jak se stát lepším, programátorem a vytvářet kvalitní software*. Vyd. 1. Překlad Ivo Magera, Lubomír Ptáček. Brno: Computer Press, 2007, 266 s. ISBN 978-80-251-1660-9.

[6] VIRIUS, Miroslav. *Jazyky C a C++ kompletní průvodce*. 2., aktualiz. vyd. Praha: Grada, 2011, 367 s. Knihovna programátora (Grada). ISBN 978-80-247-3917-5.

[7] PROKOP, Jiří. *Algoritmy v jazyku C a C++ praktický průvodce*. 1. vyd. Praha: Grada, 2009, 153 s. ISBN 978-80-247-2751-6

[8] RAZORCAT DEVELOPMENT GMBH 2009. *Tessy 2.9*: *User manual*. Berlín, 2009

[9] Testování softwaru: Problémy s pokrytím kódu. In: *Testovanisoftwaru* [online]. 5.10. 2009 [cit. 2012-04-20]. Dostupné z: http://testovanisoftwaru.blogspot.com/

[10] BUECHNER, Frank. HITEX GMBH. *Tessy Tutorial*: *Component / Integration Testing With Tessy*. Karlsruhe, 2009.

[11] SOUKENÍK, Martin. *Analýza funkční bezpečnosti parních turbín s příslušenstvím*. Zlín, 2010. Diplomová práce. Univerzita Tomáše Bati ve Zlíně

[12] SCHAFFNER, Johanna. *Gefahrenanalyse und Sicherheitskonzept nach ISO 26262 für Fahrerassistenzsysteme*. Lindau, 2011

[13] What is the ISO 26262 Functional Safety Standard?. NATIONAL INSTRUMENTS. *National Instruments* [online]. 23.2.2012 [cit. 2012-04-21]. Dostupné z: http://zone.ni.com/devzone/cda/tut/p/id/13647

[14] SCIENTIFIC TOOLWORKS, Inc. *Understand*: *User Guide and Reference Manual*. 2.5. St George, 2010. Dostupné z: http://www.scitools.com/documents/manuals/pdf/understand.pdf

[15] *Tortoisesvn* [online]. 2012 [cit. 2012-04-29]. Dostupné z: http://tortoisesvn.tigris.org/

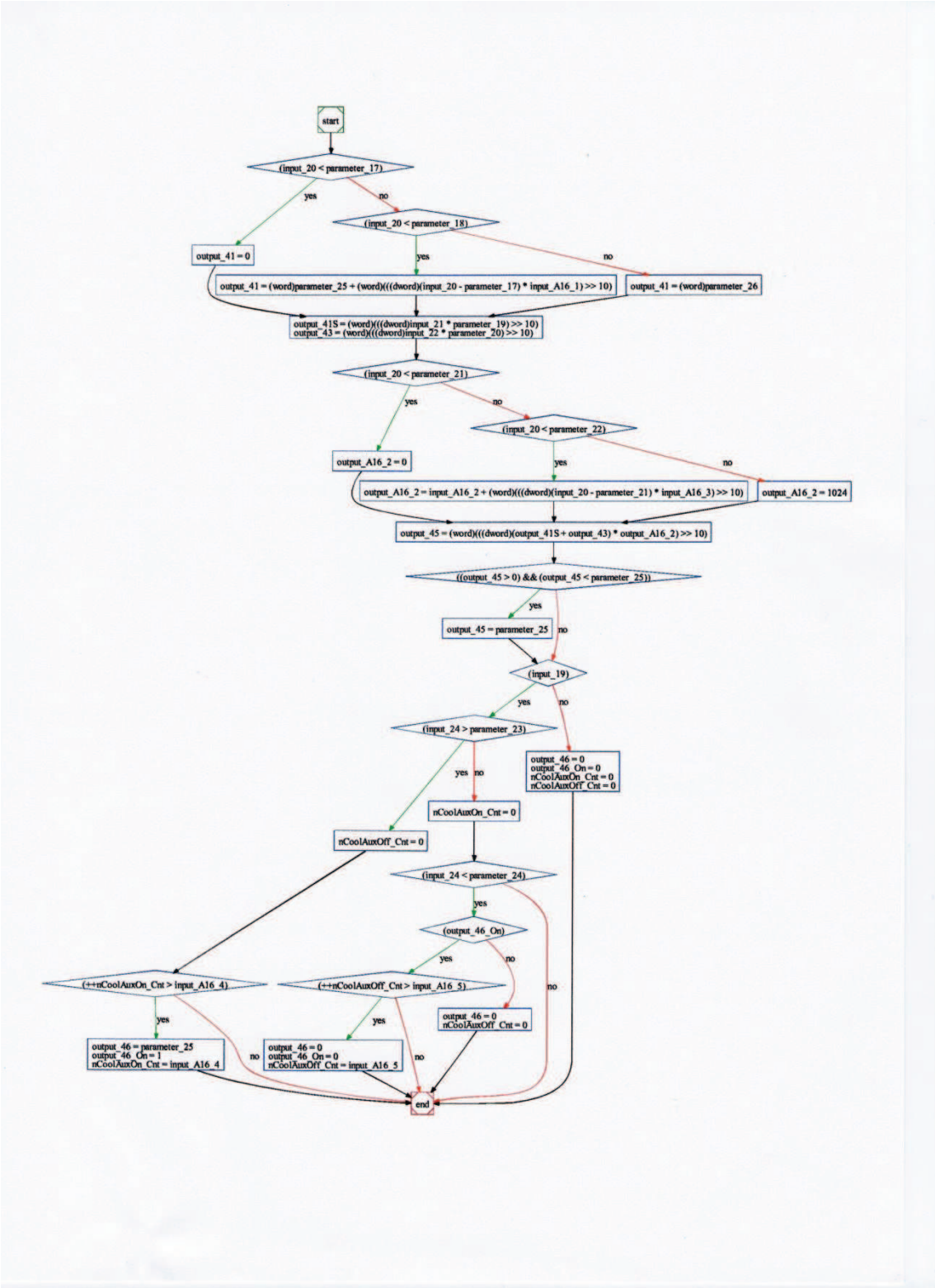
[16] LAUTERBACH DATENTECHNIK GMBH. *TRACE 32 In-Circuit Debugger*: *Quick Installation and Tutorial*. Hofolding, 1998. Dostupné z: http://www.lauterbach.com/tutorial.pdf

[17] KOVÁŘ, Josef, Zuzana PROKOPOVÁ a Ladislav ŠMEJKAL. *Programování PLC*. Zlín, 2010. Dostupné z: http://www.spszl.cz/soubory/plc/programovani_plc.pdf

8 Seznam obrázků

9 Přílohy

Příloha 1 control flow



Příloha 2 Tabulky ze specifikace testování SW modulu AM_14

	Proměnná / příznak	**Výchozí hodnoty vstupů a očekávané hodnoty výstupů**				
Vstupy	*input_1*	0	250	500	750	1000
	parameter_3	112	112	112	112	112
Výstupy	*output_1*	0	223	446	669	892

Tab. 2 Specifikace procedury_1 test 1

	Proměnná / příznak	**Výchozí hodnoty vstupů a očekávané hodnoty výstupů**					
Vstupy	*input_1*	500	500	500	1000	1000	1000
	parameter_2	88	88	88	88	88	88
	parameter_3	112	112	112	112	112	112
	input_2	200	446	500	400	892	1000
Výstupy	*output_1*	446	446	446	892	892	892
	output_2	176	392	500	352	784	1000
	output_3	224	499	500	448	999	1000

Tab. 3 Specifikace procedury_1 test 2

	Proměnná / příznak	**Výchozí hodnoty vstupů a očekávané hodnoty výstupů**				
Vstupy	*input_5*	0	10000	15000	20000	22000
	parameter_4	60	60	60	60	60
	parameter_7	10000	10000	10000	10000	10000
	parameter_8	100	100	100	100	100
	constant_1	32	32	32	32	32
Výstupy	*output_A14_5*	60	60	79	99	100

Tab. 4 Specifikace procedury_2 test 1

Příloha 3 Tabulky ze specifikace testování SW modulu AM_16

	Proměnná / příznak	**Výchozí hodnoty vstupů a očekávané hodnoty výstupů**						
Vstupy	*input_18*	1	1	1	1	0	1	0
	input_19	1	1	1	1	1	0	0
	input_20	-40	49	50	55	60	160	60
	input_A16_1	3072	3072	3072	3072	3072	3072	3072
	parameter_17	50	50	50	50	50	50	50
	parameter_18	60	60	60	60	60	60	60
	parameter_25	15	15	15	15	15	15	15
	parameter_26	45	45	45	45	45	45	45
Výstupy	*output_41*	0	0	15	30	45	45	0

Tab. 5 Specifikace procedury_1 test 1

	Proměnná / příznak	**Výchozí hodnoty vstupů a očekávané hodnoty výstupů**						
Vstupy	*input_20*	-40	34	35	45	55	60	160
	input_A16_2	512	512	512	512	512	512	512
	input_A16_3	20971	20971	20971	20971	20971	20971	20971
	parameter_21	35	35	35	35	35	35	35
	parameter_22	60	60	60	60	60	60	60
Výstupy	*output_A16_2*	0	0	512	716	921	1024	1024

Tab. 6 Specifikace Procedury_1 test 2

	Proměnná / příznak	Výchozí hodnoty vstupů a očekávané hodnoty výstupů								
Vstupy	*input_21*	0	1024	1024	1024	0	0	0	1024	1024
	input_22	0	0	0	0	1024	1024	1024	1024	1024
	input_20	60	0	35	60	0	35	60	0	60
	input_A16_2	512	512	512	512	512	512	512	512	512
	input_A16_3	20971	20971	20971	20971	20971	20971	20971	20971	20971
	parameter_19	30	30	30	30	30	30	30	30	30
	parameter_20	15	15	15	15	15	15	15	15	15
	parameter_21	35	35	35	35	35	35	35	35	35
	parameter_22	60	60	60	60	60	60	60	60	60
	parameter_25	15	15	15	15	15	15	15	15	15
Výstupy	*output_41S*	0	30	30	30	0	0	0	30	30
	output_43	0	0	0	0	15	15	15	15	15
	output_A16_2	1024	0	512	1024	0	512	1024	0	1024
	output_45	0	0	15	30	0	15	15	0	45

Tab. 7 Specifikace procedury_1 test 3

	Proměnná / příznak	Očekávané hodnoty
Výstupy	*output_41*	0
	output_41S	0
	output_43	0
	output_A16_2	0
	output_45	0
	output_46	0
	output_47	0
	output_47End	0

Tab. 8 Specifikace procedury_2 test 1

Příloha 4 Tabulky ze specifikace testování SW modulu AM_17

	Proměnná / příznak	Výchozí hodnoty vstupů a očekávaná hodnota výstupu			
Vstupy	*input_49*	0	1	0	1
	input_50	0	0	1	1
Výstup	*foutput_52_0*	0	1	1	1

Tab. 9 Specifikace procedury_1 test 1

Příloha 5 Výsledky testů modulu AM_14

Test Report

Tessy V2.9.45

Test	**Passed**
Project	**LT_AM1**
Module	**AM_14**
Testobject	**Scenarios**
User	LDEDECE
Host	localhost
Date	17.04.2012
Time	14:14:52
C1 Coverage	**100.00 %**
MCC Coverage	**n.a. %**
MC/DC Coverage	**n.a. %**

Total Testcases	**16**
Ok	**16**
Failed	0
Not Executed	0

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Table of Contents

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testobject Properties

Test Directory	C:\Dedecek\Tessy\Diplomka\LT_AM1\AM_14
Environment	Keil C166 --- LAUTERBACH TRACE32 - (Default)
Kind of Test	Component Test
Linker Options	

Sources

```
C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\AM_A14\bt_slip.c
       INCDIR("C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib\st10";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib\dis32.3xx";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\inc";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\inc\st10")
```

Attributes

Init Script $(TESSY_TESTAREA)\trace32_init_c167.cmm

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Test Results

Testcase 1 Test procedury_1				Passed
Description	Sloupec 1			
Specification	Specifikace procedury 1 tabulka 1			

Teststep 1.1				Passed
Name	**Input Value**			
constant_1	0			
input_1	0			
input_2	0			
input_5	0			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0			
parameter_3	112			
parameter_4	0x0000			
parameter_7	0			
parameter_8	0			
Name	**Actual Value**		**Expected Value**	
output_1	0	==	0	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x0000	==	0x0000	✓

Testcase 2 Test procedury_1				Passed
Description	Sloupec 2			
Specification	Specifikace procedury 1 tabulka 1			

Teststep 2.1				Passed
Name	**Input Value**			
constant_1	0			
input_1	250			
input_2	0			
input_5	0			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0			
parameter_3	112			
parameter_4	0x0000			
parameter_7	0			
parameter_8	0			
Name	**Actual Value**		**Expected Value**	
output_1	223	==	223	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x00DF	==	0x00DF	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 3 Test procedury_1				Passed
Description	Sloupec 3			
Specification	Specifikace procedury_1 tabulka 1			

Teststep 3.1				Passed
Name	**Input Value**			
constant_1	0			
input_1	500			
input_2	0			
input_5	0			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0			
parameter_3	112			
parameter_4	0x0000			
parameter_7	0			
parameter_8	0			
Name	**Actual Value**		**Expected Value**	
output_1	446	==	446	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x01BE	==	0x01BE	✓

Testcase 4 Test procedury_1				Passed
Description	Sloupec 4			
Specification	Specifikace procedury_1 tabulka 1			

Teststep 4.1				Passed
Name	**Input Value**			
constant_1	0			
input_1	750			
input_2	0			
input_5	0			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0			
parameter_3	112			
parameter_4	0x0000			
parameter_7	0			
parameter_8	0			
Name	**Actual Value**		**Expected Value**	
output_1	669	==	669	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x029D	==	0x029D	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 5 Test procedury_1				Passed
Description	Sloupec 5			
Specification	Specifikace procedury_1 tabulka 1			

Teststep 5.1				Passed
Name	**Input Value**			
constant_1	0			
input_1	1000			
input_2	0			
input_5	0			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0			
parameter_3	112			
parameter_4	0x0000			
parameter_7	0			
parameter_8	0			
Name	**Actual Value**		**Expected Value**	
output_1	892	==	892	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x037C	==	0x037C	✓

Testcase 6 Test procedury_1				Passed
Description	Sloupec 1			
Specification	Specifikace procedury_1 tabulka 2			

Teststep 6.1				Passed
Name	**Input Value**			
constant_1	0			
input_1	500			
input_2	200			
input_5	0			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	88			
parameter_3	112			
parameter_4	0x0000			
parameter_7	0			
parameter_8	0			
Name	**Actual Value**		**Expected Value**	
output_1	446	==	446	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	176	==	176	✓
output_3	224	==	224	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x01BE	==	0x01BE	✓
output_2 [10 ms]	0x00B0	==	0x00B0	✓
output_3 [10 ms]	0x00E0	==	0x00E0	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 7 Test procedury_1					Passed
Description	Sloupec 2				
Specification	Specifikace procedury_1 tabulka 2				

Teststep 7.1					Passed
Name	**Input Value**				
constant_1	0				
input_1	500				
input_2	446				
input_5	0				
output_2	0x0000				
output_3	0x0000				
output_4	0x0000				
output_5	0x0000				
parameter_2	88				
parameter_3	112				
parameter_4	0x0000				
parameter_7	0				
parameter_8	0				
Name	**Actual Value**		**Expected Value**		
output_1	446	==	446	✓	
output_11	0x0000	==	0x0000	✓	
output_12	0x0000	==	0x0000	✓	
output_14	0x0000	==	0x0000	✓	
output_16	0x0000	==	0x0000	✓	
output_2	392	==	392	✓	
output_3	499	==	499	✓	
output_4	0x0000	==	0x0000	✓	
output_5	0x0000	==	0x0000	✓	
output_A14_5	0x0000	==	0x0000	✓	
output_1 [10 ms]	0x01BE	==	0x01BE	✓	
output_2 [10 ms]	0x0188	==	0x0188	✓	
output_3 [10 ms]	0x01F3	==	0x01F3	✓	

Testcase 8 Test procedury_1					Passed
Description	Sloupec 3				
Specification	Specifikace procedury_1 tabulka 2				

Teststep 8.1					Passed
Name	**Input Value**				
constant_1	0				
input_1	500				
input_2	500				
input_5	0				
output_2	0x0000				
output_3	0x0000				
output_4	0x0000				
output_5	0x0000				
parameter_2	88				
parameter_3	112				
parameter_4	0x0000				
parameter_7	0				
parameter_8	0				
Name	**Actual Value**		**Expected Value**		
output_1	446	==	446	✓	
output_11	0x0000	==	0x0000	✓	
output_12	0x0000	==	0x0000	✓	
output_14	0x0000	==	0x0000	✓	
output_16	0x0000	==	0x0000	✓	
output_2	500	==	500	✓	
output_3	500	==	500	✓	
output_4	0x0000	==	0x0000	✓	
output_5	0x0000	==	0x0000	✓	
output_A14_5	0x0000	==	0x0000	✓	
output_1 [10 ms]	0x01BE	==	0x01BE	✓	
output_2 [10 ms]	0x01F4	==	0x01F4	✓	
output_3 [10 ms]	0x01F4	==	0x01F4	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 9 Test procedury_1		Passed
Description	Sloupec 4	
Specification	Specifikace procedury_1 tabulka 2	

Teststep 9.1 — Passed

Name	Input Value
constant_1	0
input_1	1000
input_2	400
input_5	0
output_2	0x0000
output_3	0x0000
output_4	0x0000
output_5	0x0000
parameter_2	88
parameter_3	112
parameter_4	0x0000
parameter_7	0
parameter_8	0

Name	Actual Value		Expected Value	
output_1	892	==	892	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	352	==	352	✓
output_3	448	==	448	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x037C	==	0x037C	✓
output_2 [10 ms]	0x0160	==	0x0160	✓
output_3 [10 ms]	0x01C0	==	0x01C0	✓

Testcase 10 Test procedury_1		Passed
Description	Sloupec 5	
Specification	Specifikace procedury_1 tabulka 2	

Teststep 10.1 — Passed

Name	Input Value
constant_1	0
input_1	1000
input_2	892
input_5	0
output_2	0x0000
output_3	0x0000
output_4	0x0000
output_5	0x0000
parameter_2	88
parameter_3	112
parameter_4	0x0000
parameter_7	0
parameter_8	0

Name	Actual Value		Expected Value	
output_1	892	==	892	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	784	==	784	✓
output_3	999	==	999	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	0x0000	==	0x0000	✓
output_1 [10 ms]	0x037C	==	0x037C	✓
output_2 [10 ms]	0x0310	==	0x0310	✓
output_3 [10 ms]	0x03E7	==	0x03E7	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 11 Test procedury_1					Passed
Description	Sloupec 6				
Specification	Specifikace procedury_1 tabulka 2				

Teststep 11.1					Passed
Name	**Input Value**				
constant_1	0				
input_1	1000				
input_2	1000				
input_5	0				
output_2	0x0000				
output_3	0x0000				
output_4	0x0000				
output_5	0x0000				
parameter_2	88				
parameter_3	112				
parameter_4	0x0000				
parameter_7	0				
parameter_8	0				
Name	**Actual Value**		**Expected Value**		
output_1	892	==	892	✓	
output_11	0x0000	==	0x0000	✓	
output_12	0x0000	==	0x0000	✓	
output_14	0x0000	==	0x0000	✓	
output_16	0x0000	==	0x0000	✓	
output_2	1000	==	1000	✓	
output_3	1000	==	1000	✓	
output_4	0x0000	==	0x0000	✓	
output_5	0x0000	==	0x0000	✓	
output_A14_5	0x0000	==	0x0000	✓	
output_1 [10 ms]	0x037C	==	0x037C	✓	
output_2 [10 ms]	0x03E8	==	0x03E8	✓	
output_3 [10 ms]	0x03E8	==	0x03E8	✓	

Testcase 12 Test procedury_2					Passed
Description	Sloupec 1				
Specification	Specifikace procedury_2 tabulka 1				

Teststep 12.1					Passed
Name	**Input Value**				
constant_1	32				
input_1	0x0000				
input_2	0x0000				
input_5	0				
output_2	0x0000				
output_3	0x0000				
output_4	0x0000				
output_5	0x0000				
parameter_2	0x0000				
parameter_3	0x0000				
parameter_4	60				
parameter_7	10000				
parameter_8	100				
Name	**Actual Value**		**Expected Value**		
output_1	0xFFFF	==	0xFFFF	✓	
output_11	0x0000	==	0x0000	✓	
output_12	0x0000	==	0x0000	✓	
output_14	0x0000	==	0x0000	✓	
output_16	0x0000	==	0x0000	✓	
output_2	0x0000	==	0x0000	✓	
output_3	0x0000	==	0x0000	✓	
output_4	0x0000	==	0x0000	✓	
output_5	0x0000	==	0x0000	✓	
output_A14_5	60	==	60	✓	
output_A14_5 [10 ms]	0x003C	==	0x003C	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 13 Test procedury_2				Passed
Description	Sloupec 2			
Specification	Specifikace procedury_2 tabulka 1			

Teststep 13.1				Passed
Name	**Input Value**			
constant_1	32			
input_1	0x0000			
input_2	0x0000			
input_5	10000			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0x0000			
parameter_3	0x0000			
parameter_4	60			
parameter_7	10000			
parameter_8	100			
Name	**Actual Value**		**Expected Value**	
output_1	0xFFFF	==	0xFFFF	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	60	==	60	✓
output_A14_5 [10 ms]	0x003C	==	0x003C	✓

Testcase 14 Test procedury_2				Passed
Description	Sloupec 3			
Specification	Specifikace procedury_2 tabulka 1			

Teststep 14.1				Passed
Name	**Input Value**			
constant_1	32			
input_1	0x0000			
input_2	0x0000			
input_5	15000			
output_2	0x0000			
output_3	0x0000			
output_4	0x0000			
output_5	0x0000			
parameter_2	0x0000			
parameter_3	0x0000			
parameter_4	60			
parameter_7	10000			
parameter_8	100			
Name	**Actual Value**		**Expected Value**	
output_1	0xFFFF	==	0xFFFF	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	79	==	79	✓
output_A14_5 [10 ms]	0x004F	==	0x004F	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 15 Test procedury_2 — Passed

Description	Sloupec 4
Specification	Specifikace procedury_2 tabulka 1

Teststep 15.1 — Passed

Name	Input Value
constant_1	32
input_1	0x0000
input_2	0x0000
input_5	20000
output_2	0x0000
output_3	0x0000
output_4	0x0000
output_5	0x0000
parameter_2	0x0000
parameter_3	0x0000
parameter_4	60
parameter_7	10000
parameter_8	100

Name	Actual Value		Expected Value	
output_1	0xFFFF	==	0xFFFF	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	99	==	99	✓
output_A14_5 [10 ms]	0x0063	==	0x0063	✓

Testcase 16 Test procedury_2 — Passed

Description	Sloupec 5
Specification	Specifikace procedury_2 tabulka 1

Teststep 16.1 — Passed

Name	Input Value
constant_1	32
input_1	0x0000
input_2	0x0000
input_5	22000
output_2	0x0000
output_3	0x0000
output_4	0x0000
output_5	0x0000
parameter_2	0x0000
parameter_3	0x0000
parameter_4	60
parameter_7	10000
parameter_8	100

Name	Actual Value		Expected Value	
output_1	0xFFFF	==	0xFFFF	✓
output_11	0x0000	==	0x0000	✓
output_12	0x0000	==	0x0000	✓
output_14	0x0000	==	0x0000	✓
output_16	0x0000	==	0x0000	✓
output_2	0x0000	==	0x0000	✓
output_3	0x0000	==	0x0000	✓
output_4	0x0000	==	0x0000	✓
output_5	0x0000	==	0x0000	✓
output_A14_5	100	==	100	✓
output_A14_5 [10 ms]	0x0064	==	0x0064	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Příloha 6 Výsledky testů modulu AM_16

Test Report

Tessy V2.9.45

Test	**Passed**
Project	**LT_AM1**
Module	**AM_16**
Testobject	**Scenarios**
User	LDEDECE
Host	localhost
Date	17.04.2012
Time	14:17:55
C1 Coverage	**100.00 %**
MCC Coverage	**n.a. %**
MC/DC Coverage	**n.a. %**

Total Testcases	**23**
Ok	**23**
Failed	0
Not Executed	0

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Table of Contents

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testobject Properties

Test Directory	C:\Dedecek\Tessy\Diplomka\LT_AM1\AM_16
Environment	Keil C166 --- LAUTERBACH TRACE32 - (Default)
Kind of Test	Component Test
Linker Options	

Sources

```
C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\AM_A16\b_cooling.c
       INCDIR("C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\inc";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\projects\LT_AM";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib\dis32.3xx";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib\st10")
```

Attributes

Init Script $(TESSY_TESTAREA)\trace32_init_c167.cmm

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Test Results

Testcase 1 Test procedury_1					Passed
Description	Sloupec 1				
Specification	Specifikace procedury_1 tabulka 1				
Teststep 1.1					**Passed**
Name	**Input Value**				
input_20	-40				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	3072				
input_A16_2	0x0000				
input_A16_3	0x0000				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	50				
parameter_18	60				
parameter_19	0				
parameter_20	0				
parameter_21	0				
parameter_22	0				
parameter_25	15				
parameter_26	45				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0C00	==	0x0C00	✓	
input_A16_2	0x0000	==	0x0000	✓	
input_A16_3	0x0000	==	0x0000	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	0	==	0	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	0x0000	==	0x0000	✓	
output_41 [10 ms]	0x0000	==	0x0000	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 2 Test procedury_1					Passed
Description	Sloupec 2				
Specification	Specifikace procedury_1 tabulka 1				

Teststep 2.1					Passed
Name	**Input Value**				
input_20	49				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	3072				
input_A16_2	0x0000				
input_A16_3	0x0000				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	50				
parameter_18	60				
parameter_19	0				
parameter_20	0				
parameter_21	0				
parameter_22	0				
parameter_25	15				
parameter_26	45				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0C00	==	0x0C00	✓	
input_A16_2	0x0000	==	0x0000	✓	
input_A16_3	0x0000	==	0x0000	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	0	==	0	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	0x0400	==	0x0400	✓	
output_41 [10 ms]	0x0000	==	0x0000	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 3 Test procedury B_1					Passed
Description	Sloupec 3				
Specification	Specifikace procedury_1 tabulka 1				

Teststep 3.1					Passed
Name	**Input Value**				
input_20	50				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	3072				
input_A16_2	0x0000				
input_A16_3	0x0000				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	50				
parameter_18	60				
parameter_19	0				
parameter_20	0				
parameter_21	0				
parameter_22	0				
parameter_25	15				
parameter_26	45				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0C00	==	0x0C00	✓	
input_A16_2	0x0000	==	0x0000	✓	
input_A16_3	0x0000	==	0x0000	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	15	==	15	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	0x0400	==	0x0400	✓	
output_41 [10 ms]	0x000F	==	0x000F	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 4 Test procedury_1					Passed
Description	Sloupec 4				
Specification	Specifikace procedury_1 tabulka 1				

Teststep 4.1					Passed
Name	**Input Value**				
input_20	55				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	3072				
input_A16_2	0x0000				
input_A16_3	0x0000				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	50				
parameter_18	60				
parameter_19	0				
parameter_20	0				
parameter_21	0				
parameter_22	0				
parameter_25	15				
parameter_26	45				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0C00	==	0x0C00	✓	
input_A16_2	0x0000	==	0x0000	✓	
input_A16_3	0x0000	==	0x0000	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	30	==	30	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	0x0400	==	0x0400	✓	
output_41 [10 ms]	0x001E	==	0x001E	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 5 Test procedury_1 — Passed

Description	Sloupec 5
Specification	Specifikace procedury_1 tabulka 1

Teststep 5.1 — Passed

Name	Input Value
input_20	60
input_21	0x0000
input_22	0x0000
input_24	0x0000
input_5	0x0000
input_A16_1	3072
input_A16_2	0x0000
input_A16_3	0x0000
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	50
parameter_18	60
parameter_19	0
parameter_20	0
parameter_21	0
parameter_22	0
parameter_25	15
parameter_26	45

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0C00	==	0x0C00	✓
input_A16_2	0x0000	==	0x0000	✓
input_A16_3	0x0000	==	0x0000	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	45	==	45	✓
output_41S	0x0000	==	0x0000	✓
output_43	0x0000	==	0x0000	✓
output_45	0x0000	==	0x0000	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	0x0400	==	0x0400	✓
output_41 [10 ms]	0x002D	==	0x002D	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 6 Test procedury_1					Passed
Description	Sloupec 6				
Specification	Specifikace procedury_1 tabulka 1				

Teststep 6.1					Passed
Name	**Input Value**				
input_20	160				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	3072				
input_A16_2	0x0000				
input_A16_3	0x0000				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	50				
parameter_18	60				
parameter_19	0				
parameter_20	0				
parameter_21	0				
parameter_22	0				
parameter_25	15				
parameter_26	45				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0C00	==	0x0C00	✓	
input_A16_2	0x0000	==	0x0000	✓	
input_A16_3	0x0000	==	0x0000	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	45	==	45	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	0x0400	==	0x0400	✓	
output_41 [10 ms]	0x002D	==	0x002D	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 7 Test procedury_1					Passed
Description	Sloupec 1				
Specification	Specifikace procedury_1 tabulka 2				

Teststep 7.1					Passed
Name	**Input Value**				
input_20	-40				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	0x0000				
input_A16_2	512				
input_A16_3	20971				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	0x0000				
parameter_18	0x0000				
parameter_19	0				
parameter_20	0				
parameter_21	35				
parameter_22	60				
parameter_25	0x0000				
parameter_26	0x0000				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0000	==	0x0000	✓	
input_A16_2	0x0200	==	0x0200	✓	
input_A16_3	0x51EB	==	0x51EB	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	0x0000	==	0x0000	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	0	==	0	✓	
output_A16_2 [10 ms]	0x0000	==	0x0000	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 8 Test procedury_1 — Passed

Description	Sloupec 2
Specification	Specifikace procedury_1 tabulka 2

Teststep 8.1 — Passed

Name	Input Value
input_20	34
input_21	0x0000
input_22	0x0000
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	0
parameter_20	0
parameter_21	35
parameter_22	60
parameter_25	0x0000
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0x0000	==	0x0000	✓
output_43	0x0000	==	0x0000	✓
output_45	0x0000	==	0x0000	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	0	==	0	✓
output_A16_2 [10 ms]	0x0000	==	0x0000	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 9 Test procedury_1					Passed
Description	Sloupec 3				
Specification	Specifikace procedury_1 tabulka 2				

Teststep 9.1					Passed
Name	**Input Value**				
input_20	35				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	0x0000				
input_A16_2	512				
input_A16_3	20971				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	0x0000				
parameter_18	0x0000				
parameter_19	0				
parameter_20	0				
parameter_21	35				
parameter_22	60				
parameter_25	0x0000				
parameter_26	0x0000				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0000	==	0x0000	✓	
input_A16_2	0x0200	==	0x0200	✓	
input_A16_3	0x51EB	==	0x51EB	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	0x0000	==	0x0000	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	512	==	512	✓	
output_A16_2 [10 ms]	0x0200	==	0x0200	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 10 Test procedury_1		Passed
Description	Sloupec 4	
Specification	Specifikace procedury_1 tabulka 2	

Teststep 10.1		Passed
Name	**Input Value**	
input_20	45	
input_21	0x0000	
input_22	0x0000	
input_24	0x0000	
input_5	0x0000	
input_A16_1	0x0000	
input_A16_2	512	
input_A16_3	20971	
input_A16_4	0x0000	
input_A16_5	0x0000	
output_41	0	
output_41S	0x0000	
output_43	0x0000	
output_45	0x0000	
output_46	0x0000	
output_47	0x0000	
output_A16_2	0x0000	
parameter_17	0x0000	
parameter_18	0x0000	
parameter_19	0	
parameter_20	0	
parameter_21	35	
parameter_22	60	
parameter_25	0x0000	
parameter_26	0x0000	

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0x0000	==	0x0000	✓
output_43	0x0000	==	0x0000	✓
output_45	0x0000	==	0x0000	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	716	==	716	✓
output_A16_2 [10 ms]	0x02CC	==	0x02CC	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 11 Test procedury_1					Passed
Description	Sloupec 5				
Specification	Specifikace procedury_1 tabulka 2				

Teststep 11.1					Passed
Name	**Input Value**				
input_20	55				
input_21	0x0000				
input_22	0x0000				
input_24	0x0000				
input_5	0x0000				
input_A16_1	0x0000				
input_A16_2	512				
input_A16_3	20971				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	0x0000				
parameter_18	0x0000				
parameter_19	0				
parameter_20	0				
parameter_21	35				
parameter_22	60				
parameter_25	0x0000				
parameter_26	0x0000				
Name	**Actual Value**		**Expected Value**		
input_21	0x0000	==	0x0000	✓	
input_22	0x0000	==	0x0000	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0000	==	0x0000	✓	
input_A16_2	0x0200	==	0x0200	✓	
input_A16_3	0x51EB	==	0x51EB	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	0x0000	==	0x0000	✓	
output_41S	0x0000	==	0x0000	✓	
output_43	0x0000	==	0x0000	✓	
output_45	0x0000	==	0x0000	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	921	==	921	✓	
output_A16_2 [10 ms]	0x0399	==	0x0399	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 12 Test procedury_1		Passed
Description	Sloupec 6	
Specification	Specifikace procedury_1 tabulka 2	

Teststep 12.1 — Passed

Name	Input Value
input_20	60
input_21	0x0000
input_22	0x0000
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	0
parameter_20	0
parameter_21	35
parameter_22	60
parameter_25	0x0000
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0x0000	==	0x0000	✓
output_43	0x0000	==	0x0000	✓
output_45	0x0000	==	0x0000	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	1024	==	1024	✓
output_A16_2 [10 ms]	0x0400	==	0x0400	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 13 Test procedury_1		Passed
Description	Sloupec 7	
Specification	Specifikace procedury_1 tabulka 2	

Teststep 13.1 — Passed

Name	Input Value
input_20	160
input_21	0x0000
input_22	0x0000
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	0
parameter_20	0
parameter_21	35
parameter_22	60
parameter_25	0x0000
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0x0000	==	0x0000	✓
output_43	0x0000	==	0x0000	✓
output_45	0x0000	==	0x0000	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	1024	==	1024	✓
output_A16_2 [10 ms]	0x0400	==	0x0400	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 14 Test procedury_1		Passed
Description	Sloupec 1	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 14.1		Passed
Name	**Input Value**	
input_20	60	
input_21	0	
input_22	0	
input_24	0x0000	
input_5	0x0000	
input_A16_1	0x0000	
input_A16_2	512	
input_A16_3	20971	
input_A16_4	0x0000	
input_A16_5	0x0000	
output_41	0	
output_41S	0x0000	
output_43	0x0000	
output_45	0x0000	
output_46	0x0000	
output_47	0x0000	
output_A16_2	0x0000	
parameter_17	0x0000	
parameter_18	0x0000	
parameter_19	30	
parameter_20	15	
parameter_21	35	
parameter_22	60	
parameter_25	15	
parameter_26	0x0000	

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0	==	0	✓
output_43	0	==	0	✓
output_45	0	==	0	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	1024	==	1024	✓
output_41S [10 ms]	0x0000	==	0x0000	✓
output_43 [10 ms]	0x0000	==	0x0000	✓
output_A16_2 [10 ms]	0x0400	==	0x0400	✓
output_45 [10 ms]	0x0000	==	0x0000	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 15 Test procedury_1		Passed
Description	Sloupec 2	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 15.1		Passed

Name	Input Value
input_20	0
input_21	1024
input_22	0
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	30
parameter_20	15
parameter_21	35
parameter_22	60
parameter_25	15
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0400	==	0x0400	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	30	==	30	✓
output_43	0	==	0	✓
output_45	0	==	0	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	0	==	0	✓
output_41S [10 ms]	0x001E	==	0x001E	✓
output_43 [10 ms]	0x0000	==	0x0000	✓
output_A16_2 [10 ms]	0x0000	==	0x0000	✓
output_45 [10 ms]	0x0000	==	0x0000	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 16 Test procedury_1		Passed
Description	Sloupec 3	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 16.1		Passed

Name	Input Value
input_20	35
input_21	1024
input_22	0
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	30
parameter_20	15
parameter_21	35
parameter_22	60
parameter_25	15
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0400	==	0x0400	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	30	==	30	✓
output_43	0	==	0	✓
output_45	15	==	15	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	512	==	512	✓
output_41S [10 ms]	0x001E	==	0x001E	✓
output_43 [10 ms]	0x0000	==	0x0000	✓
output_A16_2 [10 ms]	0x0200	==	0x0200	✓
output_45 [10 ms]	0x000F	==	0x000F	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 17 Test procedury_1		Passed
Description	Sloupec 4	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 17.1		Passed
Name	**Input Value**	
input_20	60	
input_21	1024	
input_22	0	
input_24	0x0000	
input_5	0x0000	
input_A16_1	0x0000	
input_A16_2	512	
input_A16_3	20971	
input_A16_4	0x0000	
input_A16_5	0x0000	
output_41	0	
output_41S	0x0000	
output_43	0x0000	
output_45	0x0000	
output_46	0x0000	
output_47	0x0000	
output_A16_2	0x0000	
parameter_17	0x0000	
parameter_18	0x0000	
parameter_19	30	
parameter_20	15	
parameter_21	35	
parameter_22	60	
parameter_25	15	
parameter_26	0x0000	

Name	Actual Value		Expected Value	
input_21	0x0400	==	0x0400	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	30	==	30	✓
output_43	0	==	0	✓
output_45	30	==	30	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	1024	==	1024	✓
output_41S [10 ms]	0x001E	==	0x001E	✓
output_43 [10 ms]	0x0000	==	0x0000	✓
output_A16_2 [10 ms]	0x0400	==	0x0400	✓
output_45 [10 ms]	0x001E	==	0x001E	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 18 Test procedury_1 — Passed

Description	Sloupec 5
Specification	Specifikace procedury_1 tabulka 3

Teststep 18.1 — Passed

Name	Input Value
input_20	0
input_21	0
input_22	1024
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	30
parameter_20	15
parameter_21	35
parameter_22	60
parameter_25	15
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0400	==	0x0400	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0	==	0	✓
output_43	15	==	15	✓
output_45	0	==	0	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	0	==	0	✓
output_41S [10 ms]	0x0000	==	0x0000	✓
output_43 [10 ms]	0x000F	==	0x000F	✓
output_A16_2 [10 ms]	0x0000	==	0x0000	✓
output_45 [10 ms]	0x0000	==	0x0000	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 19 Test procedury_1		Passed
Description	Sloupec 6	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 19.1		Passed

Name	Input Value
input_20	35
input_21	0
input_22	1024
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	30
parameter_20	15
parameter_21	35
parameter_22	60
parameter_25	15
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0400	==	0x0400	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0	==	0	✓
output_43	15	==	15	✓
output_45	15	==	15	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	512	==	512	✓
output_41S [10 ms]	0x0000	==	0x0000	✓
output_43 [10 ms]	0x000F	==	0x000F	✓
output_A16_2 [10 ms]	0x0200	==	0x0200	✓
output_45 [10 ms]	0x000F	==	0x000F	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 20 Test procedury_1		Passed
Description	Sloupec 7	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 20.1		Passed
Name	**Input Value**	
input_20	60	
input_21	0	
input_22	1024	
input_24	0x0000	
input_5	0x0000	
input_A16_1	0x0000	
input_A16_2	512	
input_A16_3	20971	
input_A16_4	0x0000	
input_A16_5	0x0000	
output_41	0	
output_41S	0x0000	
output_43	0x0000	
output_45	0x0000	
output_46	0x0000	
output_47	0x0000	
output_A16_2	0x0000	
parameter_17	0x0000	
parameter_18	0x0000	
parameter_19	30	
parameter_20	15	
parameter_21	35	
parameter_22	60	
parameter_25	15	
parameter_26	0x0000	

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0400	==	0x0400	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	0	==	0	✓
output_43	15	==	15	✓
output_45	15	==	15	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	1024	==	1024	✓
output_41S [10 ms]	0x0000	==	0x0000	✓
output_43 [10 ms]	0x000F	==	0x000F	✓
output_A16_2 [10 ms]	0x0400	==	0x0400	✓
output_45 [10 ms]	0x000F	==	0x000F	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 21 Test procedury_1		Passed
Description	Sloupec 8	
Specification	Specifikace procedury_1 tabulka 3	

Teststep 21.1		Passed

Name	Input Value
input_20	0
input_21	1024
input_22	1024
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	512
input_A16_3	20971
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	30
parameter_20	15
parameter_21	35
parameter_22	60
parameter_25	15
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0400	==	0x0400	✓
input_22	0x0400	==	0x0400	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0200	==	0x0200	✓
input_A16_3	0x51EB	==	0x51EB	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0x0000	==	0x0000	✓
output_41S	30	==	30	✓
output_43	15	==	15	✓
output_45	0	==	0	✓
output_46	0x0000	==	0x0000	✓
output_47	0x0000	==	0x0000	✓
output_47End	0x0000	==	0x0000	✓
output_A16_2	0	==	0	✓
output_41S [10 ms]	0x001E	==	0x001E	✓
output_43 [10 ms]	0x000F	==	0x000F	✓
output_A16_2 [10 ms]	0x0000	==	0x0000	✓
output_45 [10 ms]	0x0000	==	0x0000	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 22 Test procedury_1					Passed
Description	Sloupec 9				
Specification	Specifikace procedury_1 tabulka 3				

Teststep 22.1					Passed
Name	**Input Value**				
input_20	60				
input_21	1024				
input_22	1024				
input_24	0x0000				
input_5	0x0000				
input_A16_1	0x0000				
input_A16_2	512				
input_A16_3	20971				
input_A16_4	0x0000				
input_A16_5	0x0000				
output_41	0				
output_41S	0x0000				
output_43	0x0000				
output_45	0x0000				
output_46	0x0000				
output_47	0x0000				
output_A16_2	0x0000				
parameter_17	0x0000				
parameter_18	0x0000				
parameter_19	30				
parameter_20	15				
parameter_21	35				
parameter_22	60				
parameter_25	15				
parameter_26	0x0000				
Name	**Actual Value**		**Expected Value**		
input_21	0x0400	==	0x0400	✓	
input_22	0x0400	==	0x0400	✓	
input_24	0x0000	==	0x0000	✓	
input_5	0x0000	==	0x0000	✓	
input_A16_1	0x0000	==	0x0000	✓	
input_A16_2	0x0200	==	0x0200	✓	
input_A16_3	0x51EB	==	0x51EB	✓	
input_A16_4	0x0000	==	0x0000	✓	
input_A16_5	0x0000	==	0x0000	✓	
output_41	0x0000	==	0x0000	✓	
output_41S	30	==	30	✓	
output_43	15	==	15	✓	
output_45	45	==	45	✓	
output_46	0x0000	==	0x0000	✓	
output_47	0x0000	==	0x0000	✓	
output_47End	0x0000	==	0x0000	✓	
output_A16_2	1024	==	1024	✓	
output_41S [10 ms]	0x001E	==	0x001E	✓	
output_43 [10 ms]	0x000F	==	0x000F	✓	
output_A16_2 [10 ms]	0x0400	==	0x0400	✓	
output_45 [10 ms]	0x002D	==	0x002D	✓	

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testcase 23 Test procedury_2 — Passed

Description	Sloupec 1
Specification	Specifikace procedury_2 tabulka 1

Teststep 23.1 — Passed

Name	Input Value
input_20	0x0000
input_21	0x0000
input_22	0x0000
input_24	0x0000
input_5	0x0000
input_A16_1	0x0000
input_A16_2	0x0000
input_A16_3	0x0000
input_A16_4	0x0000
input_A16_5	0x0000
output_41	0x0000
output_41S	0x0000
output_43	0x0000
output_45	0x0000
output_46	0x0000
output_47	0x0000
output_A16_2	0x0000
parameter_17	0x0000
parameter_18	0x0000
parameter_19	0x0000
parameter_20	0x0000
parameter_21	0x0000
parameter_22	0x0000
parameter_25	0x0000
parameter_26	0x0000

Name	Actual Value		Expected Value	
input_21	0x0000	==	0x0000	✓
input_22	0x0000	==	0x0000	✓
input_24	0x0000	==	0x0000	✓
input_5	0x0000	==	0x0000	✓
input_A16_1	0x0000	==	0x0000	✓
input_A16_2	0x0000	==	0x0000	✓
input_A16_3	0x0000	==	0x0000	✓
input_A16_4	0x0000	==	0x0000	✓
input_A16_5	0x0000	==	0x0000	✓
output_41	0	==	0	✓
output_41S	0	==	0	✓
output_43	0	==	0	✓
output_45	0	==	0	✓
output_46	0	==	0	✓
output_47	0	==	0	✓
output_47End	0	==	0	✓
output_A16_2	0	==	0	✓
output_41 [10 ms]	0x0000	==	0x0000	✓
output_41S [10 ms]	0x0000	==	0x0000	✓
output_43 [10 ms]	0x0000	==	0x0000	✓
output_A16_2 [10 ms]	0x0000	==	0x0000	✓
output_45 [10 ms]	0x0000	==	0x0000	✓
output_46 [10 ms]	0x0000	==	0x0000	✓
output_47 [10 ms]	0x0000	==	0x0000	✓
output_47End [10 ms]	0x0000	==	0x0000	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Příloha 7 Výsledky testů modulu AM_17

Test Report

Tessy V2.9.45

Test	**Passed**
Project	**LT_AM1**
Module	**AM_17**
Testobject	**Scenarios**
User	LDEDECE
Host	localhost
Date	16.04.2012
Time	10:01:02
C1 Coverage	**100.00 %**
MCC Coverage	**n.a. %**
MC/DC Coverage	**n.a. %**

Total Testcases	**4**
Ok	**4**
Failed	0
Not Executed	0

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Table of Contents

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Testobject Properties

Test Directory	C:\Dedecek\Tessy\Diplomka\LT_AM1\AM_17
Environment	Keil C166 --- LAUTERBACH TRACE32 - (Default)
Kind of Test	Component Test
Linker Options	

Sources

```
C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\AM_A17\diag_mas.c
       INCDIR("C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib\st10";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib\dis32.3xx";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\inc";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\projects\LT_AM";
      "C:\MBtech\LT_AM\ENG1_4_SC\TestSources_2\Includes\lib")
```

Attributes

Init Script $(TESSY_TESTAREA)\trace32_init_c167.cmm

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com

Test Results

Testcase 1 Diagnostika_1				Passed
Description	Sloupec 1			
Specification	Specifikace diagnostika_1 tabulka 1			
Teststep 1.1				**Passed**
Name	**Input Value**			
input_49	0			
input_50	0			
Name	**Actual Value**		**Expected Value**	
output_52	0	==	0	✓
output_52 [10 ms]	0x0000	==	0x0000	✓

Testcase 2 Diagnostika_1				Passed
Description	Sloupec 2			
Specification	Specifikace diagnostika_1 tabulka 1			
Teststep 2.1				**Passed**
Name	**Input Value**			
input_49	1			
input_50	0			
Name	**Actual Value**		**Expected Value**	
output_52	1	==	1	✓
output_52 [10 ms]	0x0001	==	0x0001	✓

Testcase 3 Diagnostika_1				Passed
Description	Sloupec 3			
Specification	Specifikace diagnostika_1 tabulka 1			
Teststep 3.1				**Passed**
Name	**Input Value**			
input_49	0			
input_50	1			
Name	**Actual Value**		**Expected Value**	
output_52	1	==	1	✓
output_52 [0 ms]	0x0001	==	0x0001	✓

Testcase 4 Diagnostika_1				Passed
Description	Sloupec 4			
Specification	Specifikace diagnostika_1 tabulka 1			
Teststep 4.1				**Passed**
Name	**Input Value**			
input_49	1			
input_50	1			
Name	**Actual Value**		**Expected Value**	
output_52	1	==	1	✓
output_52 [10 ms]	0x0001	==	0x0001	✓

RazorCat Development GmbH
Witzlebenplatz 5 · 14057 Berlin · Tel. +49 (0) 30 - 536 357 - 0 · Fax +49 (0) 30 - 536 357 – 60
support@razorcat.com