VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

Automatizované testování PLC kódu pro TwinCAT 3 PLC

Automated testing of PLC code for TwinCAT 3 PLC

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Jan Prax
AUTHOR

VEDOUCÍ PRÁCE — Ing. Jan Roupec, Ph.D.
SUPERVISOR

BRNO 2018

ZADÁNÍ VŠKP 1

(tento list nahradíte oficiálním zadáním práce)

ABSTRAKT

Tato práce pojednává o objektově orientovaném programování a testování software obecně. Čtenář je v rámci této práce seznámen s vývojovým prostředím TwinCAT 3, ve kterém je vytvořena knihovna pro jednotkové testy. V tomto prostředí bude knihovna také implementována.

ABSTRACT

This thesis deals with an object oriented programming and testing of a software in general. A reader is also acquainted with the TwinCAT 3 integrated development environment in which a unit testing framework is created. This framework is also going to be implemented in this environment.

KLÍČOVÁ SLOVA

TwinCAT 3, paradigma, vývojář, programátor, tester, implementovat

KEYWORDS

TwinCAT 3, paradigm, developer, programmer, tester, implement

BIBLIOGRAFICKÁ CITACE

Prax, J. *Automatizované testování PLC kódu pro TwinCAT 3 PLC.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky.

PODĚKOVÁNÍ

Chtěl bych poděkovat panu doktoru Janu Roupcovi za ochotu a dobré vedení této práce. Také bych rád poděkoval panu inženýru Michalu Špačkovi za jeho odborné rady k tématu této práce.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. Jana Roupce, Ph.D. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 20. 5. 2018 ……

Jan Prax

OBSAH

1 ÚVOD

Automatizované testování je dnes rozsáhle používaná metodika testování vyvíjeného programu. Tento typ testování je rozdělen na více poddruhů, nejvíce bude probráno testování jednotkovými testy, protože tyto testy zabírají majoritní část pro zjištění chyby v testovaném programu. Jednotkové testy kontrolují pouze malé části programu na samostatnou činnost těchto malých částí, tudíž tento typ testu není závislý na vstupních datech do vyvíjeného programu.

Pro zjednodušení práce při testování programu se využívají specializované testovací knihovny. Pro vývoj počítačových a mobilních aplikací v sociální sféře je těchto knihoven na výběr dostatečné množství (například xUnit, nUnit atd.).

V průmyslové sféře, která je konzervativnější z důvodu udržování starých strojů, protože obměna starých za nové je velmi nákladná, se u mnoha firem automatizované testování ještě vůbec nevyužívá. Proto součástí této práce je vývoj knihovny pro jednotkové testy.

Tato knihovna je vytvořena ve vývojovém prostředí TwinCAT 3. Vývojové prostředí TwinCAT 3 bylo vytvořeno firmou Beckhoff, která se zabývá výrobou automatizačního hardware a software. Prostředí TwinCAT 3 bylo vytvořeno pro jednodušší vývoj aplikací pro programovatelné logické automaty a průmyslové počítače firmy Beckhoff. V prostředí TwinCAT3 je tato knihovna následně implementována.

Vývoj knihovny dodržuje postupy objektově orientovaného programování, které se rozvinuly z důvodu rychlejšího vývoje aplikace, intuitivnějšího a jednoduššího zapsání kódu a výrazně lepší přehlednosti pro pozdější úpravy ve srovnání se staršími programovacími paradigmaty.

Automatizované testování, a tedy i knihovna vytvořená jako součást této práce, přináší výhody při finálním odevzdávání programu, a to navzdory tomu, že programátor musí napsat i kód pro testy, což mu ulehčuje tato knihovna. Program je dříve dokončen díky rychlé kontrole chyb, kterou mu poskytují tyto testy. Výsledkem toho je odevzdání programu, ve kterém se vyskytuje minimum chyb. Z toho vyplývají minimální reklamace, které by způsobily snížení finálního výdělku či hodnocení.

Jednotkové testy, také poskytují výhodu při úpravě programu, kdy je třeba upravit pouze část, ale není známo, zda se nepoškodí jiná funkčnost programu. Programátor spustí tento typ testu a tím ověří, zda úprava nezpůsobila chybu.

V této práci je zařazena rešeršní část, která slouží k pochopení tématu této práce a knihovny. V rešeršní části jsou popsány základy objektově orientovaného programování, testování software a základy vývojového prostředí TwinCAT 3. Přestože jsou uvedeny základy těchto postupů, je k pochopení této práce předpokládána alespoň základní znalost programování.

2 MOTIVACE

K napsání této práce mi bylo motivací vytvoření jednoduché knihovny pro jednotkové testy ve vývojovém prostředí TwinCAT 3. Jelikož s tímto prostředím často pracuji, tak mi tato knihovna může ulehčit práci při vytváření programu a jeho následné úpravě. Také jsem měl zájem rozšířit si obzory v oboru automatizovaného testování software a chtěl jsem se zlepšit v programování ve vývojovém prostředí TwinCAT 3.

Dále mě inspiroval fakt, že tato práce může sloužit jako základ do objektově orientovaného programování, testování software, programování v prostředí TwinCAT 3 a také jako návod k použití mé knihovny.

3 OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

Objektově orientované programování je programovací paradigma, které je založeno na konceptu objektů. Tento styl programování přináší zjednodušení ve vývoji software, protože přináší rozdělení programu na samostatné celky, které lze přidávat a upravovat samostatně, což výrazně zlehčuje práci oproti starým programovacím paradigmatům. Tyto celky mohou být využívány vícekrát. OOP se snaží dosáhnout podobnosti s realitou kde objekt má své vlastnosti a schopnosti [1]. A vykonává na základě svých vlastností a schopností postupně určité jednoduché funkce, které se spojují dohromady, aby bylo jednoduše možné vykonat komplexní úkoly. Zde bude rozebrán typ založený na takzvaných třídách, jelikož je to nejpopulárnější typ OOP a je použit jako součást vývoje software pro tuto práci. [1, 2, 3, 4, 5]

3.1 Třídy a objekty v OOP

Třída předepisuje vlastnosti (atributy) a schopnosti (metody), kterými bude objekt neboli instance třídy disponovat. Objektů jedné třídy může být vytvořeno libovolné množství, neboť třída je pouze předpis, jaké vlastnosti a schopnosti má objekt mít, ale už nepřipisuje do těchto požadavků žádná data. Tato data například znamenají, že máme třídu Auto tato třída má vlastnosti (atributy) Typ pohonu, Barva a schopnost (metodu) Směr jízdy. To je vše, co nám tato třída může předepsat.

Jelikož až pro objekt (instanci třídy) se připisují přesná data, například Auto1 bude instancí třídy Auto a Auto2 bude také instancí třídy Auto. Auto1 má atribut Barva = červená, atribut Typ pohonu = pohon předních kol a metodu Směr jízdy = dozadu, ale Auto2 má atribut Barva = zelená, atribut Typ pohonu = pohon zadních kol a metodu Směr jízdy = dopředu. [1, 3, 4, 5]

3.2 Techniky OOP

3.2.1 Zapouzdření

Umožňuje data uzavřít pouze pro přístup v třídě a jejích potomcích, či je otevřít jako veřejně přístupné v projektu. Toto zapouzdření chrání vnitřní data zvolených prvků před nepovolenými úpravami zvenčí, jelikož tyto úpravy by mohly poškodit bezpečnost a soudružnost programu. [1, 3, 4, 6]

3.2.2 Dědičnost

Dědičnost v OOP povoluje programátorovi vzít již vytvořenou třídu, tato třída se pak nazývá rodič, a tuto třídu podědit. (To znamená, že se udělá kopie této třídy, kterou lze následně nadstavit dalšími metodami a atributy.) Následně tuto třídu, kterou nazýváme potomkem, nadstavit dalšími metodami a atributy. [3, 4, 5]

3.2.3 Polymorfismus

Lze hovořit o třech typech polymorfismu [3, 7, 8, 9]:

Ad Hoc: Umožňuje mít metody se stejným názvem, ale s jinou funkcí. Podmínkou je, že tyto metody se musí lišit typem parametru jinak tento typ polymorfismu není možný.

Parametrický: Díky tomuto typu může být, metoda je napsána bez přiřazených parametrů. A při její implementaci mohou být použity parametry požadovaného typu.

Podtypový: Tento typ polymorfismu umožňuje objektu, který je potomkem nadřazeného objektu, použít metodu nadřazeného objektu.

4 TESTOVÁNÍ SOFTWARE OBECNĚ

4.1 Testování software obecně

Testování je postup, při kterém se testuje program na chyby. Tyto testy přináší informace o kvalitě a stavu vyvíjeného programu. Díky testům lze objektivně posoudit rizika nasazení programu, protože testy nám mohou odhalit chyby, kterých se programátor dopustil nebo částí kódu, které pozapomenul přidat. Některé chyby se mohou vyskytnout i pří pozdější úpravě programu z důvodu vývoje, či z důvodu opravy jiné chyby a kvůli této změně se poškodí jiná, dříve fungující část programu. Brzké odhalení těchto chyb přináší zvýšení bezpečnosti a zrychlení finálního nasazení. [10, 11, 12, 13, 14]

4.2 Typy testování

4.2.1 Statické

Při tomto testování se nepřechází k spouštění testovacího kódu. Pouze se přechází ke kontrole další osobou. Částí statického testování je také spuštění nástrojů, které zkontrolují syntaxi, definici proměnných, nepoužité proměnné či nefunkční kód. [11, 15]

4.2.2 Dynamické

Pro dynamické testy je potřeba, připravit spustitelný testovací kód. Testují se různé úseky kódu, samostatné funkce či metody. Pro tyto funkce nebo metody se napíše testovací kód, do kterého se vloží požadované vstupy a očekávané výstupy. Kód se spustí a ověří funkčnost testovaných prvků. Pro napsání testovacího kódu s výhodou pomáhají testovací knihovny, které programátorovi píšícímu testy značně ulehčují jejich napsání.

Problém je, že ani v menším programu není možné pokrýt všechny možnosti, které mohou nastat. Proto je snaha docílit alespoň pokrytí velkého množství, a hlavně nejvíce pravděpodobných možností, které mohou nastat. Toto pokrytí se dosahuje nejlépe kombinací více typů testů, které zde budou vysvětleny. [10, 11, 14]

4.3 Testovací přístupy

4.3.1 Black box (Černá skříňka) testování

Tyto testy jsou nazývány černou skříňkou, protože tester (člověk, který provádí testy) nevidí zdrojový kód, ale pouze testuje implementaci tohoto kódu. I když tester nezná zdrojový kód, tak musí být seznámen, jak by měl celkový program fungovat. Když je seznámen s funkcí finálního programu, tak testuje vstupy a výstupy. Aby takto mohl tester tyto vstupní, či výstupní informace otestovat musí být od vývojáře připraveny testovací scénáře, na kterých je možno otestovat všechna data, která mohou být vložena.

Tento typ testů se používá pro testy vyšších úrovní, (testuje se i to co vidí uživatel), pro jednotkové testy se používá pouze výjimečně (viz. Testovací úrovně). Výhodou je, že tester nemusí být seznámen s programovacím jazykem, kterým byl program napsán, a nemusí být zkušený v programování, aby mohl provádět tyto testy. [11, 16, 17]

4.3.2 White-box (Bílá/Průhledná skříňka) testování

Tento typ testu je nazýván bílou skříňkou, protože tester vidí zdrojový kód. Proto se testují hlavně vnitřní části programu. Důležité je, aby tester znal vnitřní strukturu programu a jazyk ve kterém je tento program napsán. Důležité je, aby testy pokryly, pokud možno, alespoň jednou všechny funkcionality. Pokud není možné pokrýt všechny funkcionality, tak je potřeba napsat testy alespoň pro ty nejvíce důležité části kódu. Výhodou je, že jde vidět, které části kódu již byly otestovány.

Použití je z velké části na nejnižší úrovni testů, to jsou jednotkové testy (k této úrovni nemá koncový uživatel přístup). Někdy se využívá i pro vyšší úrovně testů, ale zde je již nevýhodou přílišná komplexnost testů. [11, 18, 19]

4.3.3 Grey-box (Šedá skříňka) testování

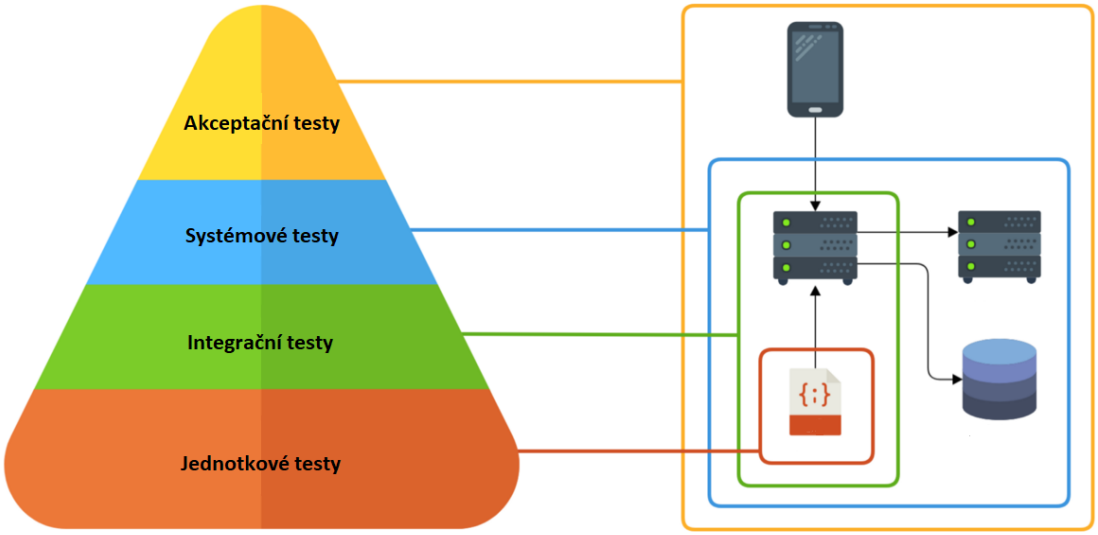
Testování zvané šedá skříňka, jak již z předchozích dvou odstavců a jména vychází, je průnik obou předchozích testovacích přístupů. Testuje se opět chování vstupů na výstupy, ale tester již musí mít částečnou znalost testovaného kódu. Díky alespoň částečné znalosti testovaného kódu, má tester mnohem lepší šanci zhodnotit co otestovat a co již testováno bylo. Většinou není vhodné, aby tester měl přístup k celému kódu, jelikož by mohl narušit zbytek kódu, který už byl otestován a funguje správně.

Tento typ je blíže testování zvanému černá skříňka, jelikož se testuje to, co má uživatel vidět, ale ten už nebude mít přístup k zdrojovému kódu. [11, 20, 21, 22]

4.4 Úrovně testování

4.4.1 Úrovně testování obecně

Testování se dělí na více úrovní, aby bylo možné otestovat co nejvíce možných scénářů, které by mohly nastat, když bude program používán koncovými zákazníky/zadavateli. Protože otestovat kvalitně program na jedné úrovni by bylo příliš komplexní a v případě většiny středních a velkých projektů nemožné. Proto dělíme tyto testy na hlavní čtyři úrovně (obr. 1) (úrovní je více, ale ty nejsou využívány tak často, a nezasahují do tématu téhle práce). Tyto úrovně při spojení dohromady by měly otestovat napsané části programu a poukázat na ještě chybějící části programu, aby tento program fungoval bezchybně jako celek. [10, 11, 14, 23, 24, 25, 26]



Obr. 1: Úrovně testů [25]

4.4.2 Jednotkové testy

Jednotkové testy mají za úkol prověřit správnost fungování pouze malých částí programu. Tyto části programu mohou být funkce, třídy v OOP, metody či atributy tříd, Je možné testovat i konstruktory a destruktory tříd.

Tyto testy jsou hlavním zástupcem testů zvaných bílá skříňka, kdy tyto testy jsou většinou napsány a prováděny vývojářem v průběhu psaní programu.

Hlavní výhodou těchto testů je fakt, že testují pouze malé části programu, do kterých jsou vkládány různé vstupy a následné výstupy jsou porovnávány s očekávanými hodnotami výstupu. Toto lze provádět pravidelně, při jakémkoli přidání nebo změně části programu. Díky tomu lze kontrolovat, jestli přidání nových funkcionalit programu nezpůsobilo chybu v předešlých částech programu.

Jednotkové testy jsou základním pilířem testovací pyramidy a neměly by chybět v žádném testování programu, jelikož přinášejí odhalení většiny chyb, které jsou nalezeny v průběhu celého cyklu testovacích úrovní, a to již ve stádiu vývoje. Toto přináší rychlejší vývoj programu a rychlejší testování v dalších úrovních testovacího cyklu, protože chyby nalezené v dalších úrovních je třeba oznámit vývojáři, který je musí opravit a následně znovu poslat na testování. Přitom by nebylo jisté, jestli se po opravě nepokazila jiná část programu, a proto jsou jednotkové testy nezbytnou částí testovacího procesu. [10, 11, 13, 14, 23, 24, 25, 26]

4.4.3 Integrační testy

V těchto testech se ověřuje funkčnost interakcí mezi různými částmi programu, které spolu mají komunikovat, když program bude používán. Integrační testy probíhají tak, že se buď postupně testuje každé spojení na vzájemné interakce nebo že se otestují všechna spojení najednou. Tyto testy jsou prováděny do té doby, než je vše spojeno (integrováno) a program funguje vcelku jako systém. [11, 23, 24, 25, 26]

4.4.4 Systémové testy

Systémové testy jsou prováděny na celkově spojeném (integrovaném) programu. Testování celkového systému je nejlépe provedeno skupinou testerů, kteří se nepodíleli na vývoji programu. Jelikož by mohli většinu věcí otestovat pouze na to, jak to funguje ale třeba nechtěně by vynechali postup, jak program bude využíván, a přitom by tento postup mohl být chybný.

Ověřuje se, jestli systém jako celek splňuje požadavky na kvalitu, výkon a bezpečnost, které byly požadovány zákazníkem či zadavatelem. Například se testuje kvalita a spolehlivost zadávání vstupu a jejich odeslání, přihlášení/odhlášení z celkového programu. Výsledkem těchto testů je program, který je otestovaný na interakce od uživatele jako celek. [11, 23, 24, 25, 26]

4.4.5 Akceptační testy

Akceptační testy jsou vykonávány na straně zákazníka/zadavatele a je kontrolováno, jestli byly splněny všechny požadavky na chod programu. Protože tyto požadavky se v průběhu času, ve kterém byl program vyvíjen mohou měnit nebo mohly být pochopeny z části jinak, než jak si to zákazník či zadavatel představoval.

Pokud program zákazníkovi/zadavateli vyhovuje a nebyly nalezeny chyby, které by se musely oznámit vývojovému týmu programátorů, pak je program možné označit za dokončený. [11, 23, 24, 25, 26]

5 TWINCAT 3

5.1 TwinCAT 3 obecně

TwinCAT 3 byl vyvinut firmou Beckhoff jako komplexní vývojové prostředí, které poskytuje možnost programování a hardwarové konfigurace přidaných PLC modulů. Toto vývojové prostředí poskytuje volbu z velké nabídky programovacích jazyků, které budou vypsány dále. Jako základ prostředí TwinCAT 3 bylo použito vývojové prostředí Visual Studio od firmy Microsoft. Při vývoji prostředí TwinCAT 3, bylo dbáno, aby dodržovalo normu IEC 61131-3 a její rozšíření o OOP. [27, 28, 29]

5.2 Podporované jazyky v prostředí TwinCAT 3

5.2.1 Real-time (jazyky reálného času)

Grafické: [28, 29, 30]
LD – jazyk příčkového diagramu
FBD – jazyk funkčních bloků
SFC – sekvenční funkční diagram
CFC – kontinuální funkční diagram [31]
Podpora Integrace Matlab/Simulink přes rozšíření [32].
UML [29]

Textové: [28, 29, 30]
IL – jazyk seznamu instrukcí
ST – jazyk strukturovaného textu, tento jazyk bude použit pro vývoj knihovny v této práci a v příkladech této práce.
C/C++ [33]

5.2.2 Non-real-time (klasické jazyky)

Tyto jazyky podporované vývojovým prostředím TwinCAT 3, umožňují tvorbu úprav, či programů pro TwinCAT 3, které nepoběží v realném čase. Jsou to vyšší programovací jazyky, a tak v nich programy jsou zpravidla vytvořeny jednodušeji a rychleji než v nižších programovacích jazycích (např. ST). A protože jsou to rozšířené jazyky, tak v nich může provádět úpravy i programátor, který nemá zkušenost s programováním pro PLC.

Tyto podporované jazyky jsou například C#, .NET, Visual Basic, C++, Silverlight. [28, 29]

5.3 Vytvoření projektu v TwinCAT 3

Kliknutím na ozubené kolo v modrém se otevře nabídka pro spuštění prostředí TwinCAT 3, ve které pro spuštění TwinCAT 3 je třeba kliknout na TwinCAT XAE (obr. 2).

Obr. 2: Spuštění TwinCAT

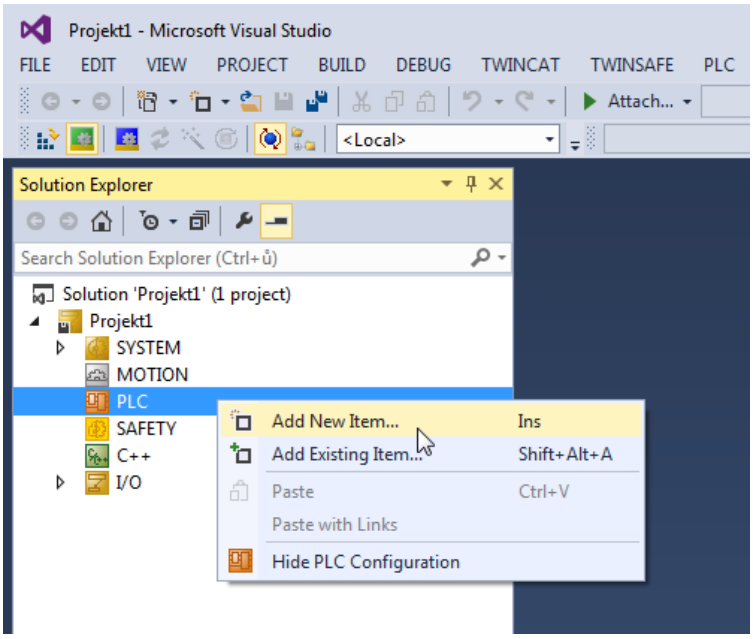
Následně se klikne na New Project, to otevře okno se jménem New Project a zde se klikne na Templates – TwinCAT Projects a klikne se na TwinCAT XAE Project (XML format). Do políčka Name se vypíše jméno projektu a klikne se na tlačítko OK (obr. 3).

Obr. 3: Vytvoření projektu

5.4 Vytvoření PLC projektu v TwinCAT 3

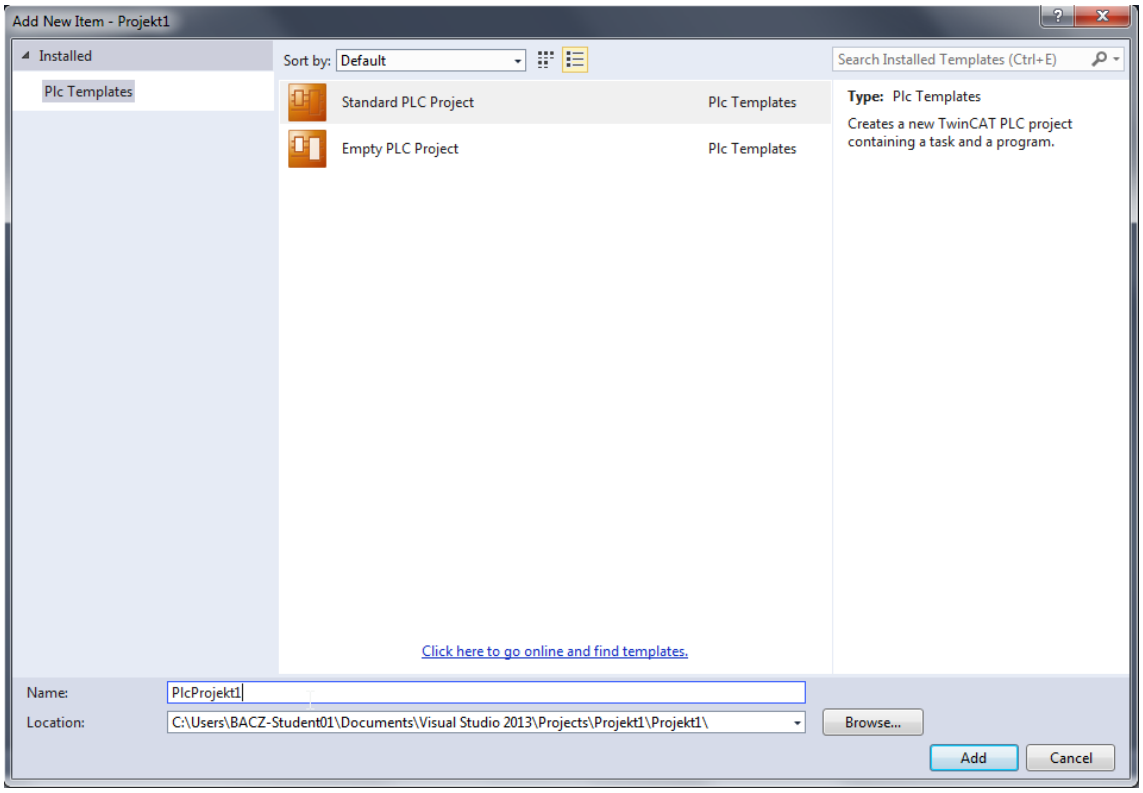
PLC projekt je v prostředí TwinCAT3 do celkového projektu/řešení (solution), vytvoření celkového projektu bylo ukázáno v minulé kapitole.

Pro vytvoření PLC projektu se klikne pravým tlačítkem myši na PLC, následně levým tlačítkem myši na Add New Item... (obr. 4)



Obr. 4: Vytvoření PLC projektu

Zobrazí se okno Add New Item (obr. 5) kde se klikne na Standard PLC Project. Do políčka Name se zadá název PLC projektu a klikne se na tlačítko Add. Tímto je projekt vytvořen.



Obr. 5: Výběr PLC projektu

6 VLASTNÍ ŘEŠENÍ – KNIHOVNA PRO JEDNOTKOVÉ TESTY V TWINCAT 3

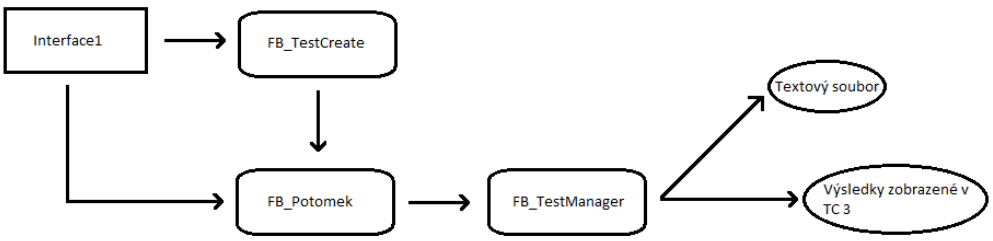
6.1 Význam knihovny pro jednotkové testy

Tato knihovna vznikla za účelem usnadnění testování programu, jelikož poskytuje rozhraní, jak psát jednotkové testy ve vývojovém prostředí TwinCAT 3 a takto napsané testy následně sesbírá, vyhodnotí a zobrazí výsledky přímo v prostředí TwinCAT 3. Tyto výsledky také vypíše do textového souboru, který si následně lze prohlédnout či zálohovat.

6.2 Vytváření knihovny

6.2.1 Návrh

Před napsáním této knihovny bylo nutné rozvrhnout postup, aby byla představa, z jakých částí se knihovna bude skládat. Návrh knihovny splňuje postupy objektově orientovaného programování. Ve finále se rozhodlo, že se základ knihovny bude skládat ze dvou hlavních funkčních bloků. Funkční blok je ve vývojovém prostředí TwinCAT 3 jiný název pro třídu, jak ji známe s OOP. Název funkční blok je jiná věc než jazyk funkčních bloků, a proto je třeba dávat pozor na to, aby tyto dva pojmy nebyly zaměňovány. Základem je funkční blok FB_TestCreate, který sbírá data z funkčních bloků, které z něj budou poděděny. Aby z těchto poděděných bloků mohly vzniknout předlohy, které se budou jednoduše používat pro jednotkové testy. Další základní funkční blok této knihovny je FB_TestManager, který přebírá data z FB_TestCreate a předává je k vypsání. Bloky, které budou poděděny z FB_TestCreate budou implementovat rozhraní Interface1, aby testy byly psány a shromažďovány jednotně (obr. 6).



Obr. 6: Návrh funkce knihovny

6.3 Popis vytvořených funkčních bloků

6.3.1 FB_TestCreate

Je funkční blok, který má za úkol sesbírat adresy a jména instancí funkčních bloků, které z tohoto bloku budou dědit. Programátor poděděním bloku FB_TestCreate získá metody tohoto bloku, o které se nemusí starat a které mu zajišťují vyhodnocení jeho testů.

Tento funkční blok implementuje rozhraní Interface1, aby mohl získávat výsledky testů z poděděných funkčních bloků, které též budou implementovat rozhraní Interface1. Implementace tohoto rozhraní v poděděných funkčních blocích je nutná, aby bylo sjednoceno vyhodnocování těchto testů.

6.3.2 Metody FB_TestCreate

Metoda **FB_Init** se zavolá kdykoli je vytvořena instance funkčního bloku, kterému náleží. Tohoto se s výhodou využilo k sesbírání adres a jmen těchto funkčních bloků. K tomuto sesbírání adres slouží metoda **Hand_Over_Adress**.

6.3.3 FB_TestManager

Má na starosti sesbírat výsledky z testů. Tyto testy jsou instance bloků, které byly poděděny z FB_TestCreate. Tyto výsledky funkční blok FB_TestManager upraví do pole přehledných výpisů. Toto pole je zobrazeno jak v prostředí TwinCAT 3, tak i v textovém souboru.

6.3.4 Metody FB_TestManager

Metoda **CollectResults** má za úkol, postupně sesbírat výsledky testů z adres, které jsme získali z instancí funkčních bloků poděděných z funkčního bloku FB_TestCreate a tyto výsledky zapsat do pole.

Metoda **ConvertConcat** se stará o to, aby výsledky testů byly dobře čitelné pro osobu, která je bude kontrolovat. Proto výsledky převede na text a spojí je s popiskem a zapíše do pole řetězců.

Metoda **WriteInto_Txt** zapíše pole řetězců, vzniklé jako výstup z metody ConvertConcat, do textového souboru na adrese C:\TwinCAT\Test_log_Results.txt. Pokud soubor na této adrese neexistuje, tak je vytvořen. Pokud soubor na zadané adrese existuje, tak je přepsán.

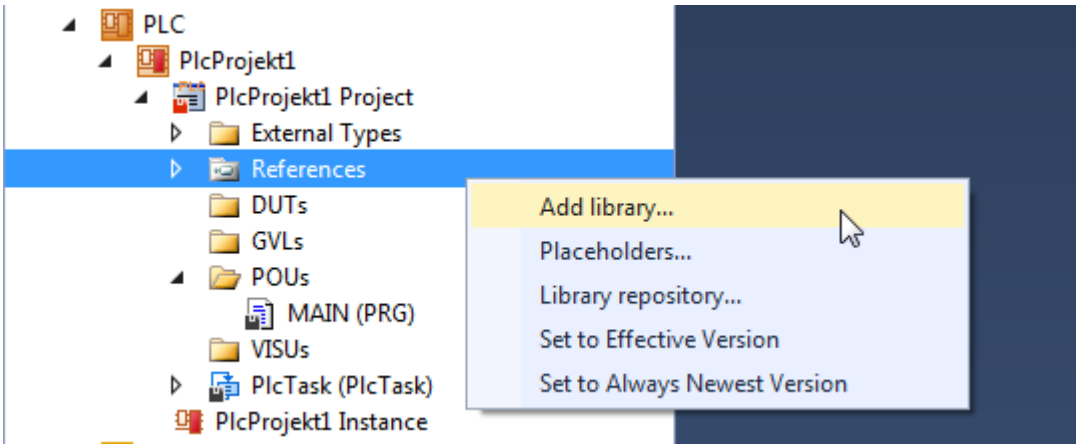
6.3.5 Rozhraní Interface1

Když funkční blok ve vývojovém prostředí TwinCAT 3 implementuje nějaké rozhraní, musí též implementovat jeho metody. Proto se vytvořilo rozhraní Interface1 s metodou TestMethod, aby se sjednotil vzhled jednotkových testů napsaných za použití této knihovny, a též aby se sjednotilo shromažďování výsledků těchto testů.

6.4 Instalace a přidání knihovny Unit_Test_TC3

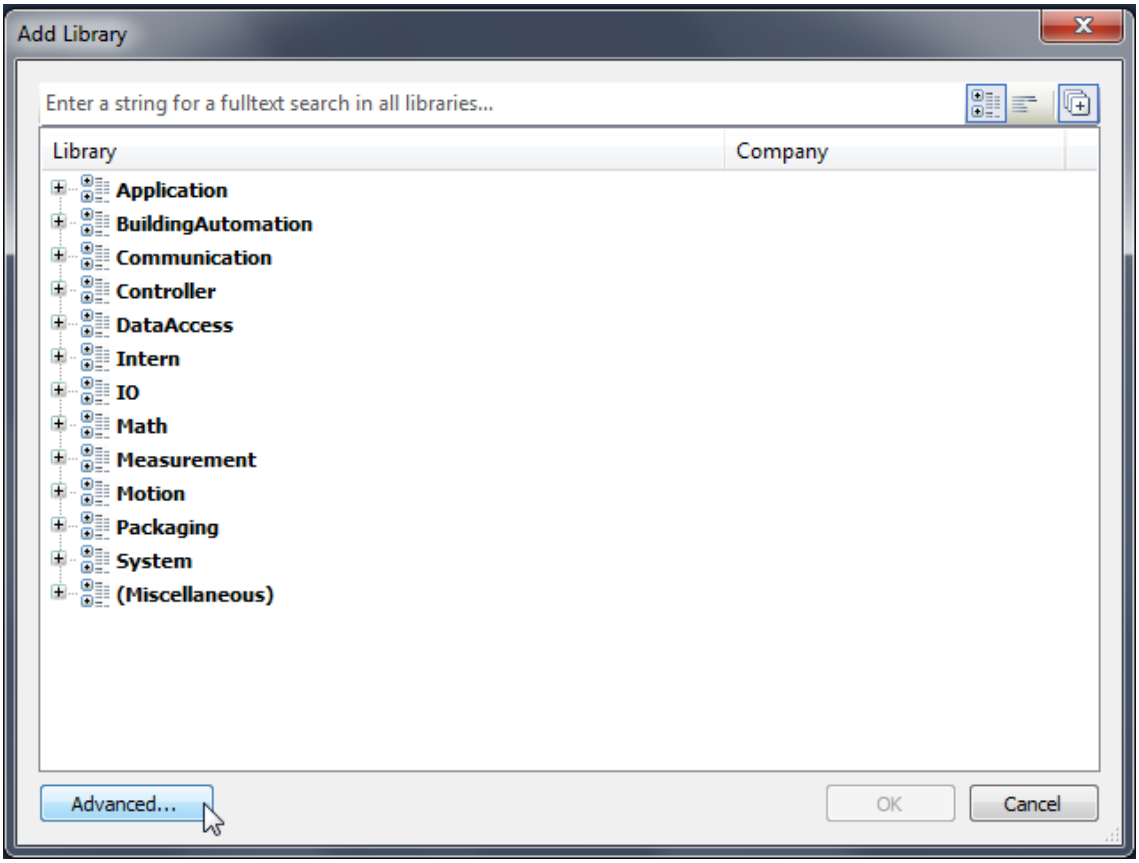
6.4.1 Instalace knihovny Unit_Test_TC3

V PLC projektu se klikne pravým tlačítkem myši na References a levým tlačítkem myši na Add library… (obr. 7)



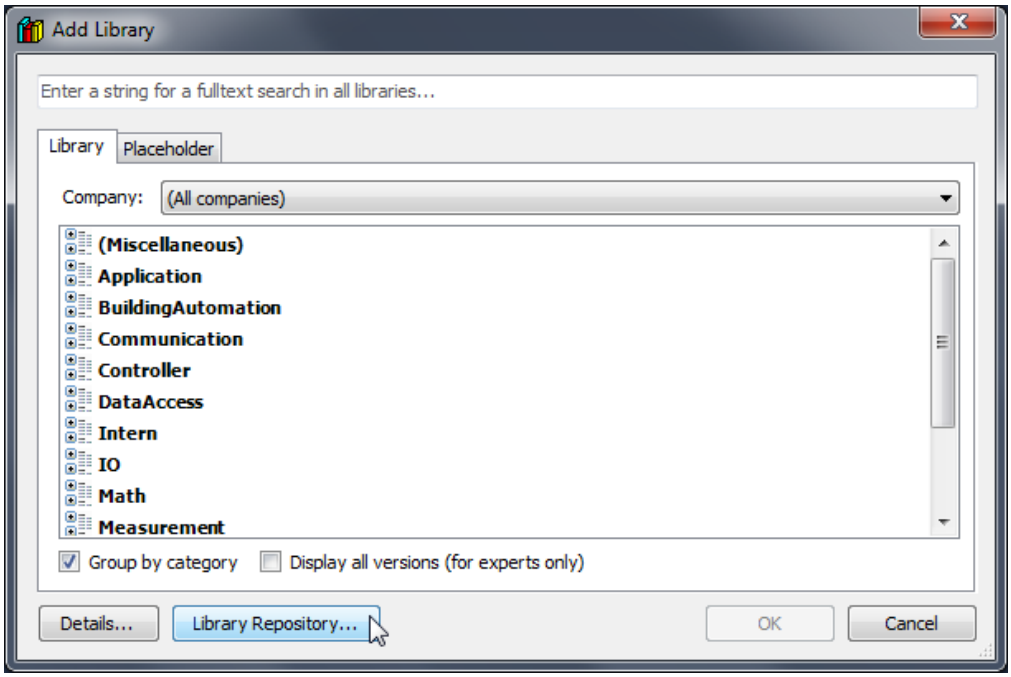
Obr. 7: Instalace knihovny 1

Zobrazí se okno, kde se klikne na tlačítko Advanced… (obr. 8)



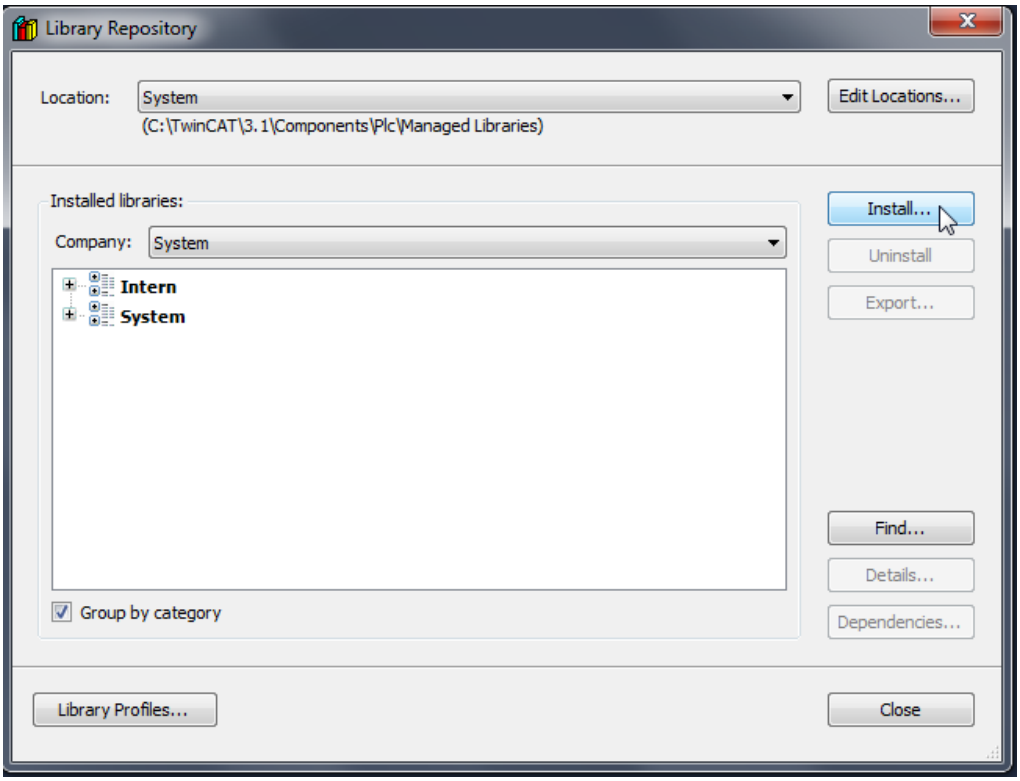
Obr. 8: Instalace knihovny 2

Následně se klikne na tlačítko Library Repository… (obr. 9)



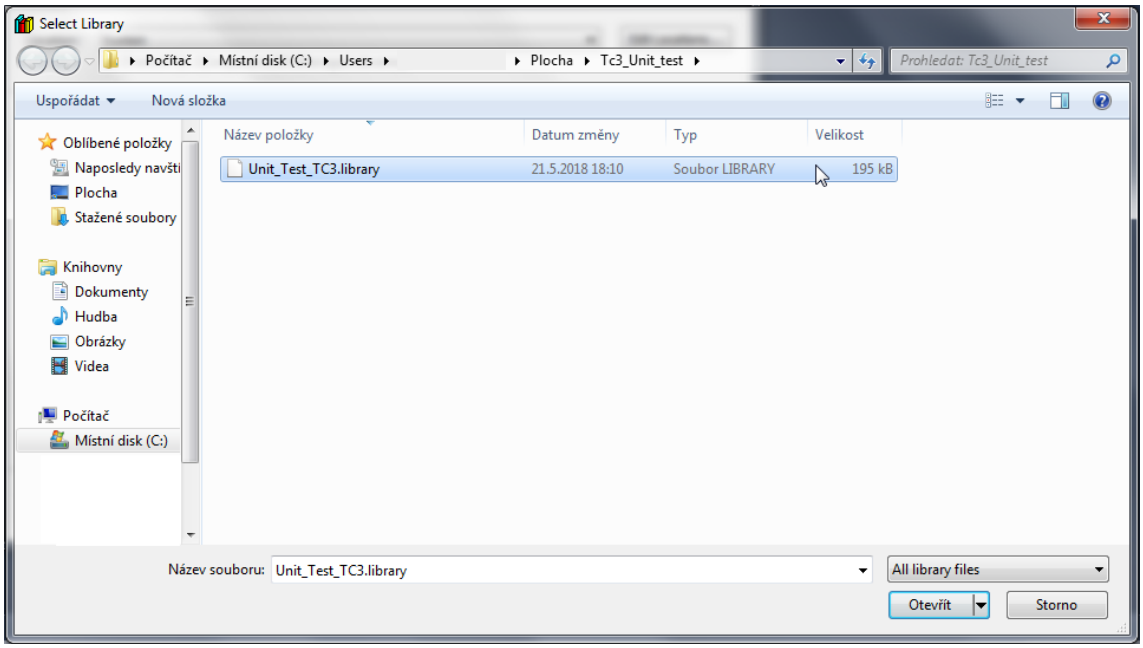
Obr. 9: Instalace knihovny 3

V okně Library repository se klikne na tlačítko Install… (obr. 10)



Obr. 10: Instalace knihovny 4

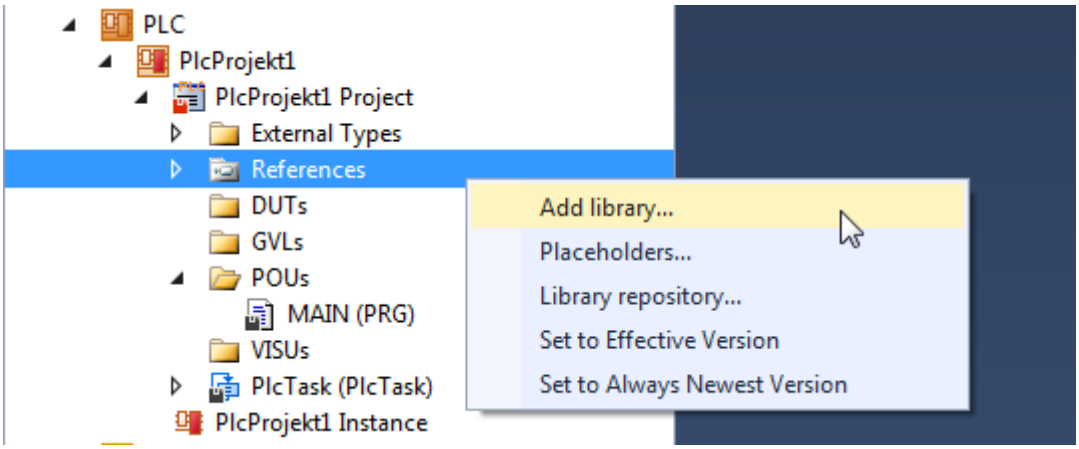
Po kliknutí na tlačítko Install (obr. 10), se zobrazí okno, ve kterém lze vyhledat knihovnu vytvořenou jako součást této práce (obr. 11). Proto je třeba tuto knihovnu uložit do počítače, či na přenosné úložiště, aby ji bylo možné v tomto okně dohledat.



Obr. 11: Instalace knihovny 5, vyhledání knihovny

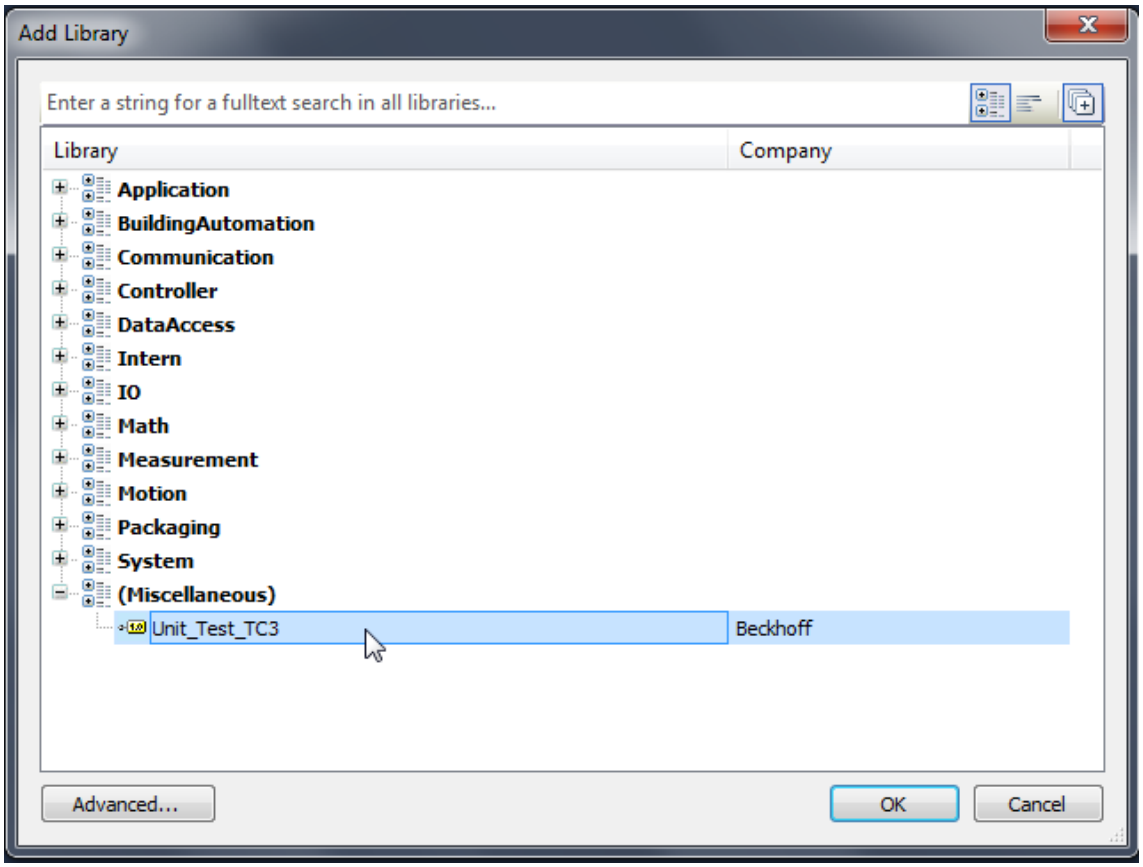
6.4.2 Přidání knihovny Unit_Test_TC3

Pro přidání knihovny se opět (jako v kroku instalace této knihovny) v PLC projektu klikne pravým tlačítkem myši na References a levým tlačítkem myši na Add library… (obr. 12)



Obr. 12: Přidání knihovny

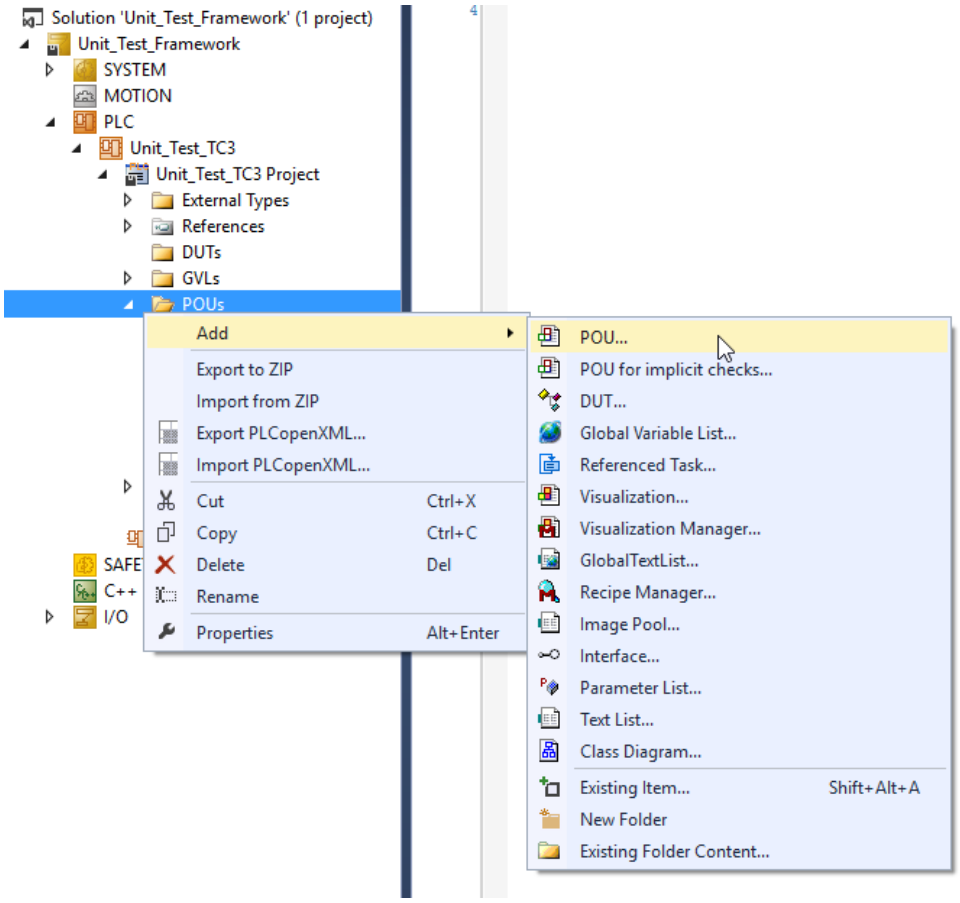
Objeví se okno Add Library (obr. 13). a v tomto okně se klikne na (Miscellaneous), kde se zvolí knihovna Unit_Test_TC3 a potvrdí se tlačítkem OK. Tímto je knihovna přidána a je možné využívat její funkční bloky a metody.



Obr. 13: Přidání knihovny Unit_Test_TC3

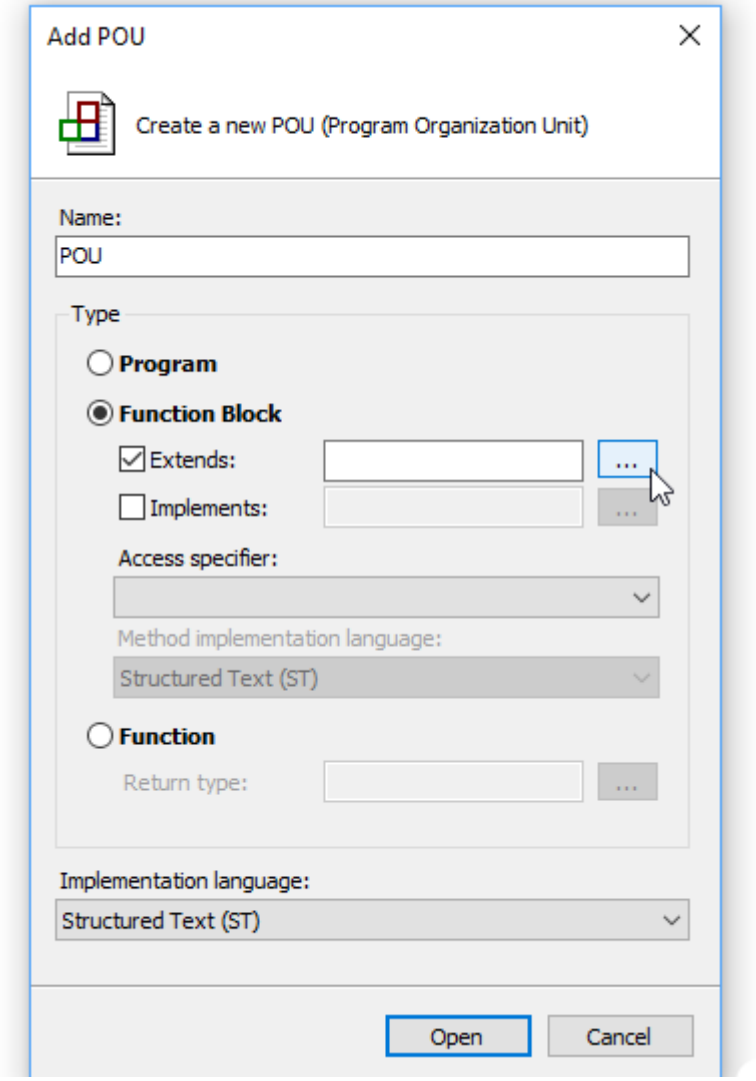
6.5 Vytvoření funkčního bloku pro jednotkové testy v této knihovně

Ve vytvořeném PLC projektu se klikne pravým tlačítkem myši na POUs, následně levým tlačítkem myši na Add a následně na POU… (obr. 14)



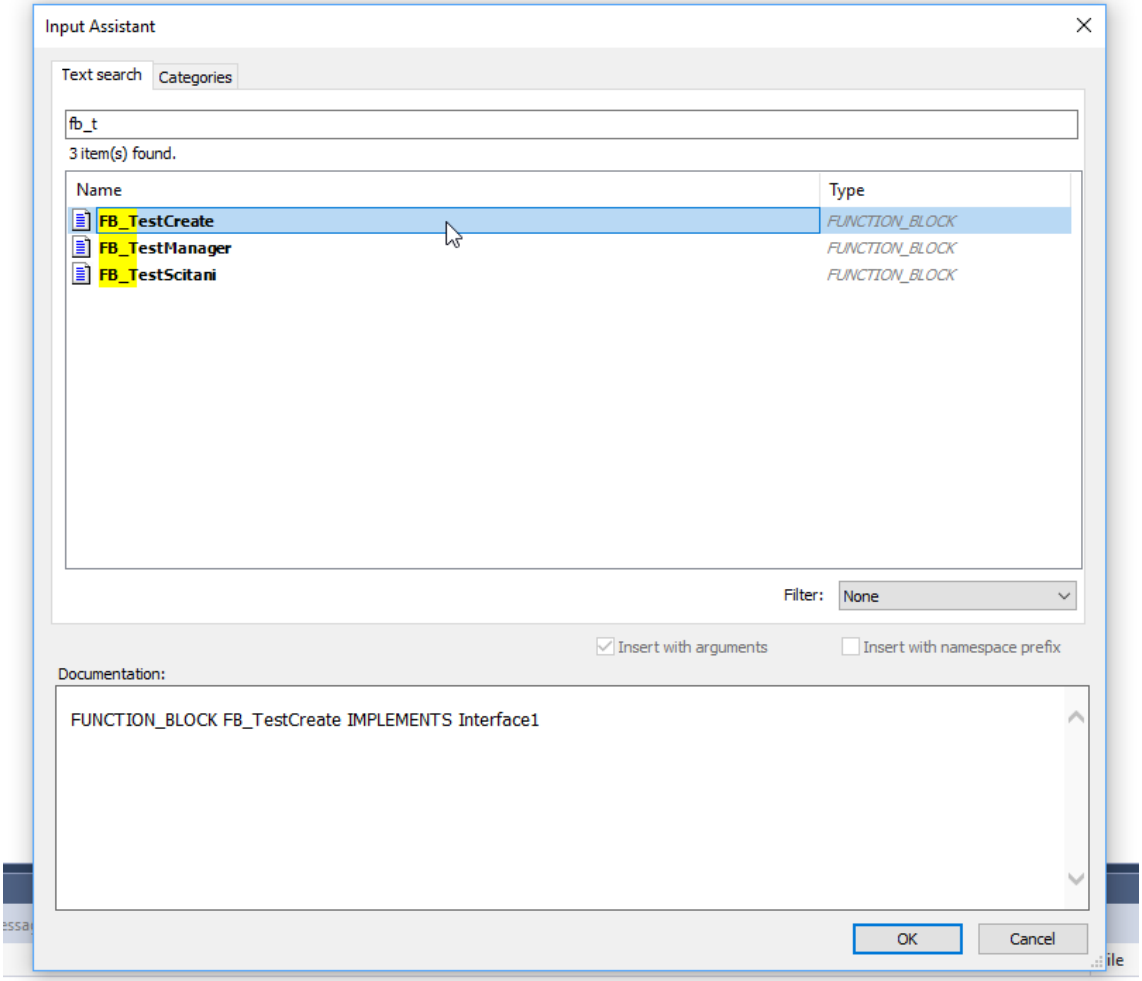
Obr. 14: Přidání POU v PLC projektu

Otevře se okno pro zjednodušené zadávání funkčního bloku (obr. 15), kde je třeba kliknout na výběr Function Block a následně zaškrtnout políčko Extends to znamená, že vytvářený funkční blok podědí metody funkčního bloku, který se následně vybere. Potom se klikne na políčko s třemi tečkami (viz. Pozice kurzoru na (obr. 15)).



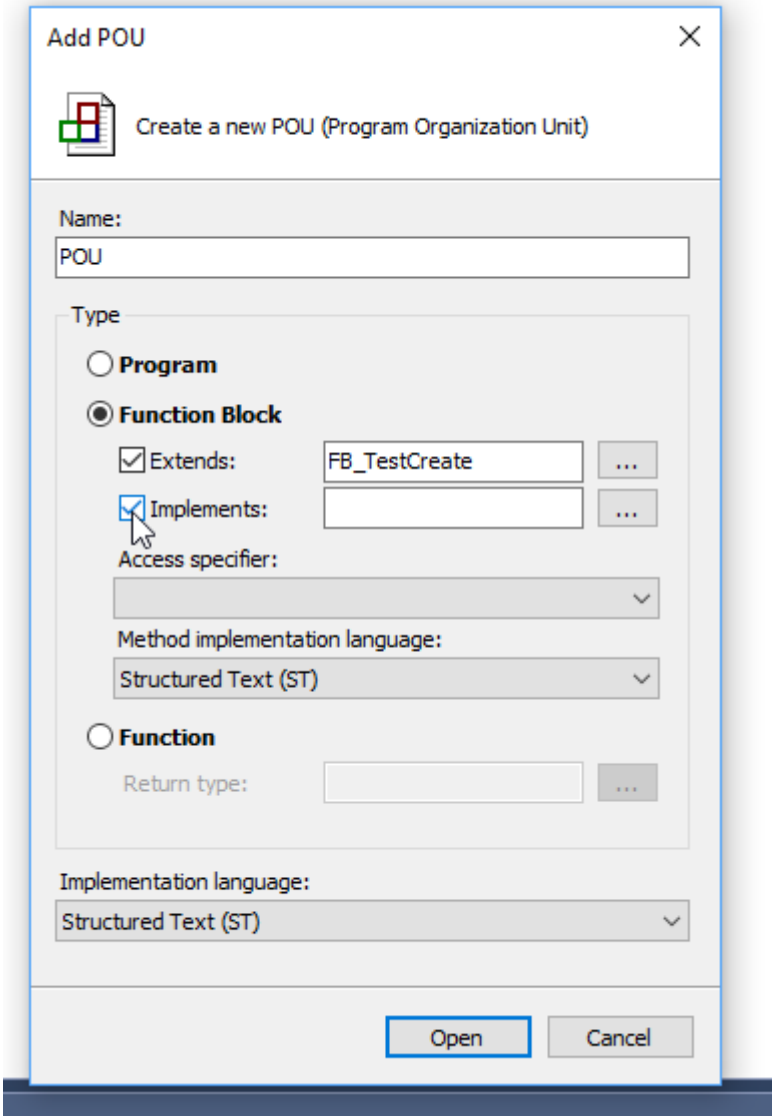
Obr. 15: Add POU okno pro zadávání

Otevře se pomocník při zadávání (obr. 16) kde je třeba kliknout na políčko Text search a do zadávacího řádku napsat, alespoň část názvu funkčního bloku FB_TestCreate. Potom se FB_TestCreate označí kliknutím a výběr se potvrdí kliknutím na tlačítko OK.



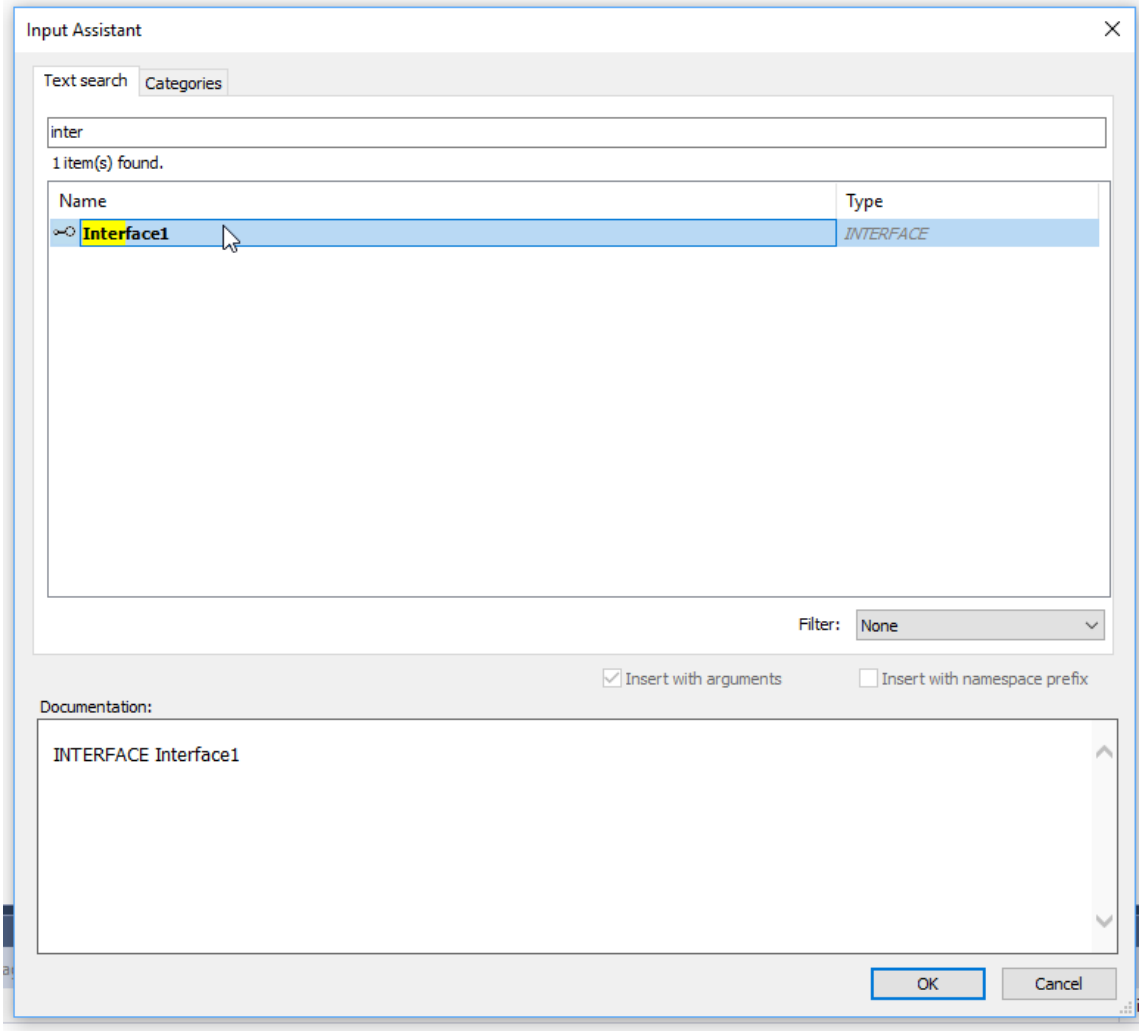
Obr. 16: Pomocník při zadávání

Výběr se následně vrátí do okna Add POU (obr. 17), kde se zaškrtne políčko Implements a klikne se na tlačítko s třemi tečkami, které je na stejném řádku jako Implements.



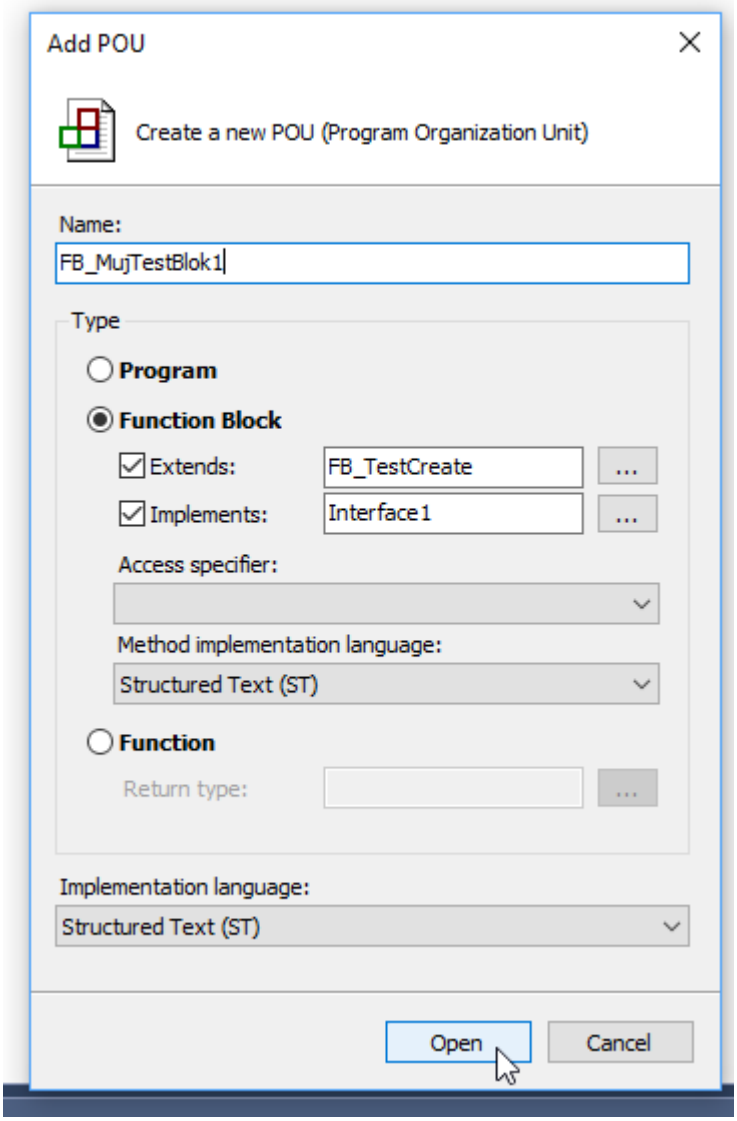
Obr. 17: Add POU okno pro zadávání 2

Otevře se pomocník při zadávání (obr. 18), kde je třeba kliknout na políčko Text search a do zadávacího řádku napsat, alespoň část názvu rozhraní Interface1. Potom se Interface1 označí kliknutím a výběr se potvrdí kliknutím na tlačítko OK.



Obr. 18: Pomocník při zadávání 2

Výběr se vrátí do okna Add POU (obr. 19) a zde již stačí přidat pouze název a kliknout na tlačítko Open.



Obr. 19: Add POU přidání názvu

Takto (obr. 20) vypadá okno vytvořeného funkčního bloku tímto postupem, ke kterému se přidala metoda TestMethod, jelikož když funkční blok implementuje rozhraní, tak se mu přidají i metody tohoto rozhraní.

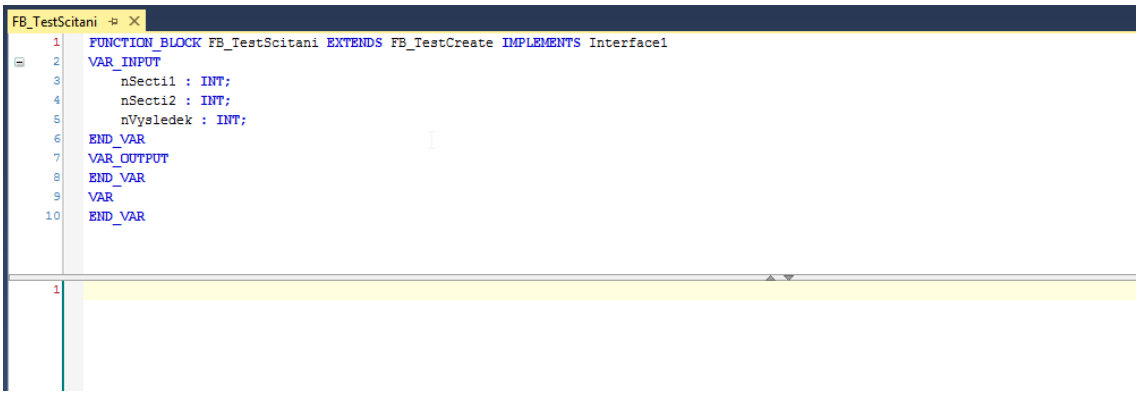
V tomto funkčním bloku, tedy hlavně metodě TestMethod, jen zbývá napsat test, jehož instance bude možné používat na testování různých vstupů a výstupů.

Obr. 20: Funkční blok pro testy

7 ZHODNOCENÍ A DISKUZE

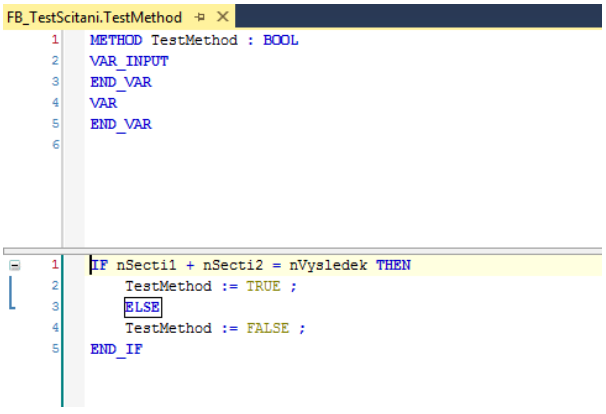
Jako výsledek této práce byla vytvořena knihovna s názvem Unit_Test_TC3. V předchozí kapitole bylo ukázáno, jak tuto knihovnu nainstalovat a přidat. V této kapitole bude ukázáno, jak použití této knihovny funguje.

Na obrázku (obr. 21) jde vidět, že byl vytvořen funkční blok s názvem FB_TestScitani, který má tři vstupy. Tyto vstupy používá metoda TestMethod, kterou tento funkční blok obsahuje kvůli implementaci Interface1.



Obr. 21: FB_TestScitani vstupy

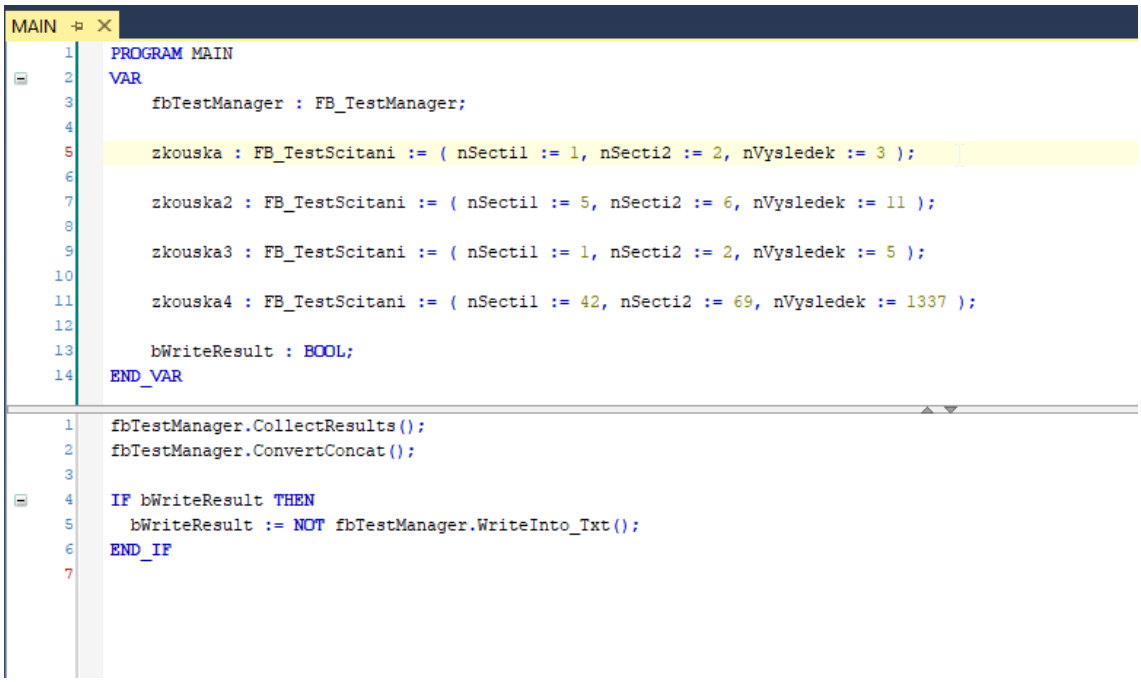
Metoda TestMethod funkčního bloku FB_TestScitani testuje, jestli dvě čísla nSecti1 a nSecti2, když se sečtou, dávají stejný výsledek, jako byl zadán programátorem do vstupu nVysledek (obr. 22).



Obr. 22: FB_TestScitani a jeho TestMethod

Tímto je napsán testovací funkční blok, kterému je třeba vytvořit instance.

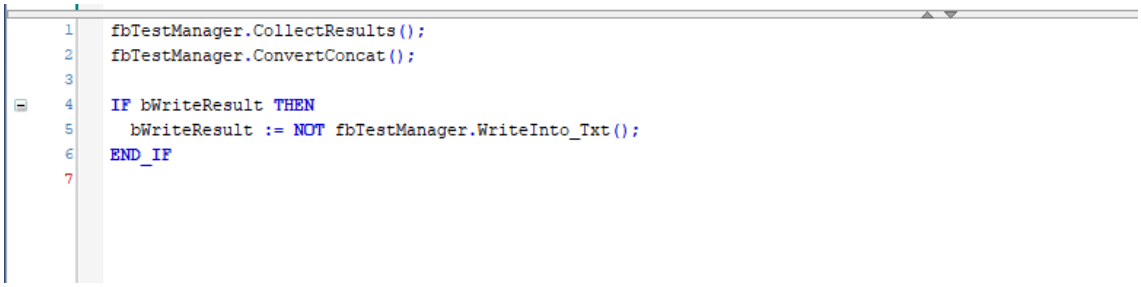
Tyto instance byly pro ukázku vytvořeny v programu MAIN a byly nazvány zkouska s příslušným číslem. Těmto instancím byly následně přiřazeny hodnoty pro součty. Je vidět, že hodnoty byly zadány tak (obr. 23), aby zkouska a zkouska2 vycházely správně, ale zkouska3 a zkouska4, aby daly špatné výsledky a tím pádem byly vyhodnoceny jako nesprávné.



Obr. 23: instance FB_TestScitani

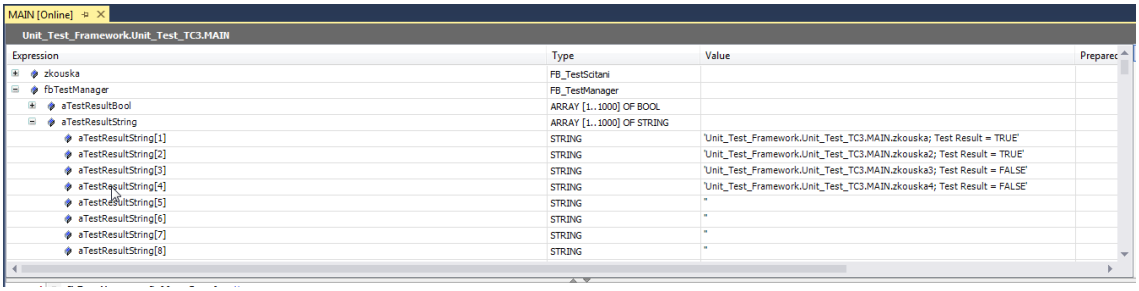
Pro vyhodnocení a vypsání výsledků je třeba zavolat metody CollectResults a ConvertConcat (obr. 24). Tyto metody jsou metody instance funkčního bloku fbTestManager.

Pro vypsání výsledků do textového souboru je třeba zavolat metodu WriteInto_Txt. Tato metoda stejně jako dvě předešlé metody je instancí funkčního bloku fbTestManager. Metoda WriteInto_Txt je uzavřená v podmínce if, aby proběhla jenom jednou, jelikož program MAIN běží cyklicky (obr. 24).



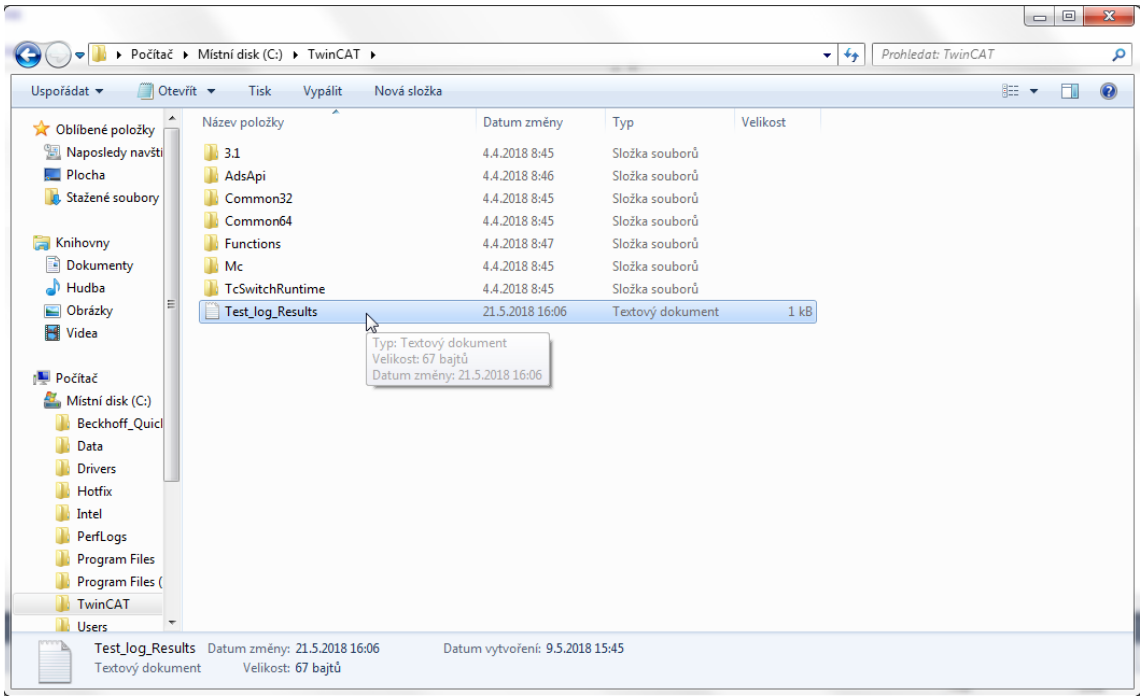
Obr. 24: FB_TestScitani_TestMethod

Z obrázku (obr. 25) lze vidět, že po spuštění programu se testy vyhodnotily správně a jsou vypsány v prostředí TwinCAT 3 v přehledné podobně v kolonce value.



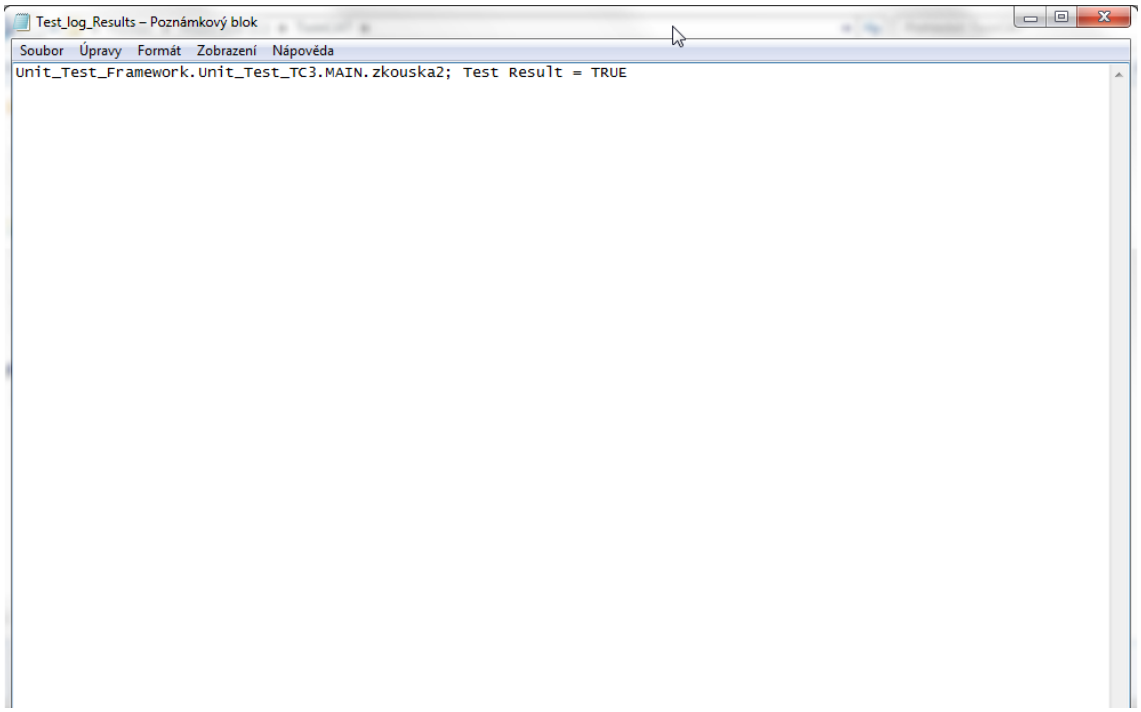
Obr. 25: Výsledky testů

Jsou vypsány (obr. 26) i v souboru Test_log_Results na adrese C:\TwinCAT.



Obr. 26: : Soubor s výpisem výsledků

Takto (obr. 27) vypadá vypsání testů v souboru Test_log_Results. Bohužel se vypisuje pouze jeden test, jelikož nebylo ještě dořešeno řádkování, které bude vyřešeno v dalších verzích knihovny.



Obr. 27: Výpis výsledků v souboru

Kromě problému s řádkováním výpisu v souboru Test_log_Results probíhá instalace a přidání knihovny v pořádku. Spouštění a vyhodnocování testů také funguje bez chyb.

8 ZÁVĚR

K pochopení této práce byly vysvětleny potřebné informace, které jsou s ní úzce spjaty. Tyto informace obsáhly základy objektově orientovaného programování a jeho prvky, Testování Software (kódu) v obecném rozhledu, ale některé části jsou probrány detailněji, a to hlavně jednotkové testy, neboť v rámci této práce byla vytvořena knihovna pro automatizované testování jednotkovými testy. Do souhrnu těchto informací byl zařazen i TwinCAT 3, aby se čtenář seznámil s tímto vývojovým prostředím.

Vytvořená knihovna pro automatizované testování PLC kódu v prostředí TwinCAT3 je zaměřena na nejnižší úroveň testování v rámci pyramidy testů. Obsahuje funkční bloky a metody, které vytvářejí rozhraní pro psaní jednotkových testů a také tyto testy sesbírávají a vyhodnocují. Čtenáři je vysvětleno, jak tuto knihovnu nainstalovat, přidat a používat ve vývojovém prostředí TwinCAT 3. Také je čtenáři vysvětleno, jak tato knihovna funguje.

Funkčnost této knihovny byla podrobena zkoušení a ověřilo se, že knihovna jde bezproblémově nainstalovat, přidat a také, že knihovna v pořádku funguje. Část tohoto zkoušení je vidět na obrázcích v rámci této práce. Knihovna byla po tomto zkoušení vyhodnocena jako funkční, ale byla nalezena chyba výpisu do souboru, která se opraví v dalších verzích této knihovny.

9 SEZNAM POUŽITÉ LITERATURY

[1] Čápka, David. 1. díl - Úvod do objektově orientovaného programování v C#. [webová stránka]. [cit. 2018-05-08]. Dostupné z: https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-uvod-do-objektove-orientovaneho-programovani

[2] Objektově orientované programování. [webová stránka]. [cit. 2018-05-08]. Dostupné z: https://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD

[3] Object-oriented programming. [webová stránka]. [cit. 2018-05-08]. Dostupné z: https://en.wikipedia.org/wiki/Object-oriented_programming

[4] Class-based programming. [webová stránka]. [cit. 2018-05-08]. Dostupné z: https://en.wikipedia.org/wiki/Class-based_programming

[5] Object oriented programming. [webová stránka]. [cit. 2018-05-09]. Dostupné z: https://python.swaroopch.com/oop.html

[6] Serrano, Harold. The three types of encapsulation in OOP. [webová stránka]. 2016-05-30 [cit. 2018-05-09]. Dostupné z: https://www.haroldserrano.com/blog/the-tree-types-of-encapsulation-in-object-oriented-programming

[7] C#-Polymorphism. [webová stránka]. [cit. 2018-05-11]. Dostupné z: https://www.tutorialspoint.com/csharp/csharp_polymorphism.htm

[8] Polymorphism (computer science). [webová stránka]. [cit. 2018-05-11]. Dostupné z: https://en.wikipedia.org/wiki/Polymorphism_(computer_science)

[9] Polymorfismus (programování). [webová stránka]. [cit. 2018-05-11]. Dostupné z: https://cs.wikipedia.org/wiki/Polymorfismus_(programov%C3%A1n%C3%AD)

[10] Hammil, Paul. Unit test frameworks. O'Reilly Media. 2009-02 [cit. 2018-05-11]. ISBN 10: 0596006896 ISBN 13: 9780596006891

[11] Software testing. [webová stránka]. [cit. 2018-05-11]. Dostupné z: https://en.wikipedia.org/wiki/Software_testing

[12] Sharma, Lakshay. Software testing. [webová stránka]. [cit. 2018-05-11]. Dostupné z: http://toolsqa.com/software-testing/software-testing/

[13] Garvey, Cormac. Unit testing for Industrial Automation software development. What is it?. [webová stránka]. [cit. 2018-05-11]. Dostupné z: http://hal-software.com/unit-testing-for-industrial-automation-software-development-what-is-it/

[14] Feathers, C., Michael. Working effectively with legacy code. Prentice Hall International. 2004-10-2 [cit. 2018-05-11]. ISBN 10: 0131177052 ISBN 13: 978-0131177055

[15] Static testing. [webová stránka]. [cit. 2018-05-11]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/static_testing.htm

[16] Black box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: http://softwaretestingfundamentals.com/black-box-testing/

[17] Black box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/black_box_testing.htm

[18] White box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: http://softwaretestingfundamentals.com/white-box-testing/

[19] White box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/white_box_testing.htm

[20] Grey box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: https://www.guru99.com/grey-box-testing.html

[21] Grey box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/grey_box_testing.htm

[22] Gray box testing. [webová stránka]. [cit. 2018-05-17]. Dostupné z: https://www.tutorialspoint.com/software_testing_dictionary/grey_box_testing.htm

[23] Software testing levels. [webová stránka]. [cit. 2018-05-17]. Dostupné z: http://softwaretestingfundamentals.com/software-testing-levels/

[24] Software testing-levels. [webová stránka]. [cit. 2018-05-17]. Dostupné z: https://www.tutorialspoint.com/software_testing/software_testing_levels.htm

[25] Peck, Nathan. Microservice testing: Introduction. [webová stránka]. 2017-09-19 [cit. 2018-05-17]. Dostupné z: https://medium.com/@nathankpeck/microservice-testing-introduction-347d2f74095e

[26] Pearson, LaTonya. The four levels of software testing. [webová stránka]. 2015-09-11 [cit. 2018-05-19]. Dostupné z: https://www.seguetech.com/the-four-levels-of-software-testing/

[27] TwinCAT 3 Getting started. [online prezentace]. [cit. 2018-05-20]. Dostupné z: https://download.beckhoff.com/download/document/catalog/TwinCAT_3_Booklet.pdf

[28] Beckhoff Automation GmbH, TwinCAT 3 eXtended Automation (XA). 2012-04 [online prezentace]. [cit. 2018-05-20]. Dostupné z: https://download.beckhoff.com/download/document/catalog/TwinCAT_3_Booklet.pdf

[29] Halva, Tomáš. Drha, Štěpán, Beckhoff Česká republika s. r. o. Začínáme s Beckhoffem. [webová stránka]. 2016-03-14 [cit. 2018-05-20]. Dostupné z: http://www.controlengcesko.com/index.php?id=47&no_cache=1&tx_ttnews%5btt_news%5d=6205&cHash=b85074aafd&type=98

[30] Programovací jazyky pro PLC. [webová stránka]. [cit. 2018-05-21]. Dostupné z: https://coptkm.cz/portal/?doc=3905

[31] Continuous Function Chart Editor(CFC). [webová stránka]. [cit. 2018-05-21]. Dostupné z: https://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/TcPlcCtrl_EditorCFC.htm&id

[32] Matlab/Simulink TwinCAT 3. [online prezentace]. 2017-07-06 [cit. 2018-05-21]. Dostupné z: https://download.beckhoff.com/download/document/automation/twincat3/TwinCAT_3_Matlab_Simulink_EN.pdf

[33] Create TwinCAT 3 C++ project [webová stránka]. [cit. 2018-05-21]. Dostupné z: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/54043195639144971.html&id

10 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

OOP = objektově orientované programování
POU = program organization unit (programová organizační jednotka)

11 SEZNAM PŘÍLOH

DVD

PŘÍLOHY

Na DVD:
Unit_Test_TC3.lib
Bakalarska_prace_2018_JanPrax.pdf