VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATICKÉ REGRESNÍ TESTY PRO SW BALÍČKY SDK A CMSIS

AUTOMATIC REGRESSION TESTS FOR SDK AND CMSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ SVOBODA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ADAM CRHA**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Svoboda Tomáš**
Program: Informační technologie
Název: **Automatické regresní testy pro SW balíčky SDK a CMSIS**
Automatic Regression Tests for SDK and CMSIS
Kategorie: Analýza a testování softwaru

Zadání:

1. Analyzujte problematiku differencování XML souborů včetně provedení rešerše dostupných metod a nástrojů
2. Seznamte se systémem pro kontinuální integraci Atlassian Bamboo, vývojovým prostředím MCUXpresso, IAR Workbench, uVision, technologiemi CMSIS-pack.
3. Na základě získaných znalostí navrhněte automatizované regresní testy pro SW balíčky nad kompatibilními IDE rozhraními a pro porovnání mezi novými a předešlými verzemi SW balíčků.
4. Navrhněte přehledné rozhranní pro vizualizaci výsledků regresních testů.
5. Implementujte návrh z bodu 3 a 4 s ohledem na maximální výkonnost.
6. Vyhodnoťte navržené řešení a diskutujte možnosti dalšího rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1. - 4.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Crha Adam, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 26. října 2018

Abstrakt

Tato bakalářská práce se zabývá návrhem a popisem implementace automatických regresních testů pro softwarové balíčky SDK a CMSIS s využitím systému pro kontinuální integraci Atlassian Bamboo. Cílem bylo vytvořit sadu testů nad vývojovým prostředím MCUXpresso, IAR Workbench a uVision. Dále vytvořit sadu testů pro porovnávání souborových struktur a souborů mezi novými a předešlými verzemi softwarových balíčků. Podstatnou částí této práce je teoretický základ týkající se algoritmu a porovnávání strukturovaných souborů XML a jeho modifikací. Pro vizualizaci výsledků regresních testů, kvůli velkému objemu dat, bylo potřeba vytvořit přehledné výstupní rozhraní.

Abstract

In my bachelor thesis, I deal with design and description of implementation of automatic regression tests for SDK and CMSIS software packages using the continuous integration system Atlassian Bamboo. The goal was to create a set of tests over IDE interfaces MCUXpresso, IAR Workbench, uVision. Furthermore, to compare file structures and files between new and previous versions of software packages. An essential part of my work is also a theoretical basis concerning the algorithm and comparison of XML files and his modifications. For visualizing large amounts of data from the regression test output, it was necessary to create a clear interface.

Klíčová slova

XML, Python, MCUXpresso, IAR Workbench, uVision, Bash, regresní testování, automatické testy, IDE testy, porovnávací testy, SDK, CMSIS, SW balíčky, optimalizace

Keywords

XML, Python, MCUXpresso, IAR Workbench, uVision, Bash, regression testing, automatic tests, IDE tests, comparison tests, SDK, CMSIS, SW packages, optimalization

Citace

SVOBODA, Tomáš. *Automatické regresní testy pro SW balíčky SDK a CMSIS*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Adam Crha

Automatické regresní testy pro SW balíčky SDK a CMSIS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Adama Crhy. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Svoboda
14. května 2019

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Adamovi Crhovi, za poskytnutí odborné pomoci při řešení této práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V posledních desetiletích se důležitost informačních technologií v životě lidí neustále zvyšovala. Zároveň se zvyšovala i složitost systémů a tím vznikla potřeba tyto systémy testovat. Začaly vznikat programy, které slouží k ověření správné činnosti těchto systémů. Tyto programy typicky slouží pro testování softwaru. V dnešní době je testování nedílnou součástí vývoje softwaru. S postupem času vznikaly různé testovací aplikační rámce nebo celé testovací systémy pro rozsáhlejší projekty. Tato práce využívá těchto testovacích systémů. Konkrétně se jedná o testovací systém pro kontinuální integraci a nasazení `Atlassian Bamboo`.

Cílem této práce je vytvoření automatizovaných regresních testů pro softwarové balíčky `CMSIS` a `SDK` firmy `NXP`. V první fázi regresních testů se stáhnou už hotové vygenerované softwarové balíčky pomocí `Backdooru` z jedné z instancí `MCUXpresso` serveru. V druhé fázi se spustí sada testů nad kompatibilními `IDE` rozhraními `uVision`, `MCUXpresso IDE`a `IAR Workbench`, které mají za úkol naklonovat, provést přeložení a sestavení ukázkových projektů ze stažených softwarových balíčků. Dále se porovnávají stávající s nově vygenerovanými softwarovými balíčky a v poslední, třetí fázi se výsledná data testů zpracují do přehledného výstupního rozhraní a uloží na síťový disk. Část výstupního rozhraní se rozešle emailem.

V kapitole 2 je představena problematika porovnávání souborové struktury softwarových balíčků a `XML` souborů. Je zde popsán formát `XML` a algoritmus pro jeho porovnávání. Dále se zde zaměřuje na analýzu již existujících řešení a důležitosti automatizovaného testování. V kapitole 3 jsou popsány požadavky na projekt, struktura regresních testů, výstupního rozhraní a zadních vrátek webové aplikace `MCUXpresso`. Kapitola 4 obsahuje popis vývojových prostředí, ve kterých se projekt vytvářel a která sloužila jako testovací prostředí. Dále zde najdete důvody použití `Atlassian` nástrojů. Kapitola 5 se věnuje samotné implementaci automatických regresních testů. Závěrečná kapitola 6 shrnuje výsledky práce a další možná rozšíření.

Kapitola 2

Úvod do problematiky

V této kapitole je představena problematika automatizovaného testování, porovnávání souborových struktur a specifických souborových formátů. Také je zde popsán `XML` formát, který je podstatný pro tuto práci.

2.1 Automatizované testování

Automatizované testování softwaru je důležitou součástí vývoje softwaru. V praxi bývá často podceňovanou disciplínou, a to i přesto, že přináší výrazné úspory prostředků a zvyšuje kvalitu výsledného produktu. Automatizovat lze od testování jednotlivých manuálních testů výsledného produktu až po celý proces vývoje softwaru. Se zvyšováním výpočetního výkonu počítačů je realizace automatizovaného testování stále dostupnější.

"Pokud jsou pro daný software vytvořené manuální testovací případy, které se opakovaně vykonávají, můžeme se ptát, zda by nebylo možné tyto testy vykonávat automaticky."

Touto myšlenkou vzniklo zadání této bakalářské práce.

Hlavním účelem automatizace testů softwaru je časová úspora a zefektivnění postupu testování softwaru. Pokud jsou testy prováděny bez možnosti zásahu lidského faktoru, výrazně se tím snižuje možnost chybného provedení postupu testování softwaru a tím také pravděpodobnost bezporuchového provozu softwaru. Pokud hovoříme o plné automatizaci, lze testovací sady spouštět v kteroukoliv dobu. Běžně se v praxi využívá této skutečnosti a testy se spouští během noci.

V této práci se zabývám automatizovanými regresními a komparačními testy. Komparační testy jsou součástí regresních testů. Při vytváření testů je nutné se seznámit s testovaným softwarem a znát očekávané vstupy a výstupy. Architektura testovací sady musí umožňovat automatizaci. Proto je nutné tyto testy s tímto vědomím vytvářet.

Obecně se snažíme testy pokrýt co největší část aplikace. U aplikací s krátkou životností v řádech měsíců nebo s jedinou vydanou verzí se nám nevyplatí vytvářet tyto testy. Naopak u aplikací, kde předpokládáme delší životnost v řádech několika let a více vydávaných verzí, se nám zcela jistě realizace automatizovaných testů vyplatí.

U automatizovaných testů se pokrývají především části, u kterých je reálná hrozba výskytu závažných chyb. Při realizaci testů se však musí přihlédnout k tomu, zda daná část aplikace nebude často upravována. Znamenalo by to častější udržování automatizovaných testů.

Automatizované testování má i svá negativa. V agilním vývoji softwaru, kde dochází k častým změnám softwaru, může automatizované testování v konečném důsledku zpomalit

vývoj. Proto by se měla zvolit vhodná časová investice pro vývoj a udržování automatizovaných testů.

Regresní testování se zaměřuje zejména na kontinuální spouštění funkčních a nefunkčních testů, aby bylo ověřeno, zda testovaný software po změně stále funguje. Především tedy, zda změna v kódu nezpůsobila chybu v jiné části softwaru.

2.2 XML formát

XML (Extensible Markup Language) je značkovací jazyk určený na serializaci dat pomocí stromové struktury elementů a atributů. Jazyk je určen hlavně pro výměnu dat mezi aplikacemi a pro publikování dokumentů, u kterých popisuje strukturu z hlediska obsahu jednotlivých částí. Prezentace dokumentu (vzhled) může být definována pomocí kaskádových stylů. Další možností zpracování je transformace do jiného typu dokumentu, nebo do jiné aplikace XML [19]. Hlavními konkurenty XML jsou například JSON[1] či YAML[2].

Hlavní výhoda těchto jazyků spočívá v čitelnosti člověkem i strojem. Pro člověka je jednoduché vytvořit či editovat XML dokumenty v libovolném textovém editoru. Na druhou stranu nevýhodou XML je velikost souboru. XML obsahuje velké množství formátovacích informací. Zabírá tedy více místa na disku a zpracování dat je výpočetně náročnější oproti JSON, YAML či binární podobě dat. Velké množství formátovacích informací nemusí být nutně na škodu, protože standart říká, že stručnost zdělených informací má při vývoji minimální prioritu. XML strom může být libovolně hluboko vnořený, což je dalším faktorem, který bude mít negativní dopad na výpočetní výkon a zátěž RAM paměti (nebezpečí stack overflow[3] atp.).

V současné době je XML hodně využívaným datovým formátem. V oblasti systémové integrace, kde se přenáší data v určitém formátu mezi dvěma systémy, formát XML téměř zcela nahradil v minulosti používané technologie používající binární formáty (např. před tím populární technologii CORBA[4]).

2.3 Softwarové balíčky

Softwarové balíčky SDK a CMSIS firmy NXP jsou soubory volně dostupných ovladačů, middleware a referenčních příkladů aplikací s otevřeným zdrojovým kódem. Každý softwarový balíček patří ke konkrétní desce, kitu nebo procesoru. Slouží pro urychlení vývoje softwaru na zařízeních vyrobených firmou NXP.

2.3.1 SDK balíčky

Softwarový balíček SDK (Software Development Kit) firmy NXP je sada vývojových nástrojů primárně určená pro vývojové prostředí MCUXpresso IDE. SDK balíčky jsou volně dostupné ke stažení z webové aplikaci MCUXpresso SDK Builder [24], která je pro vás vytvoří.

Webová aplikace MCUXpresso SDK Builder se nachází na 5 produkčních serverech:

[1] JSON – JavaScript Object Notation
[2] YAML – Ain't Markup Language
[3] stack overflow – přetečení na zásobníku v paměti
[4] CORBA – Common Object Request Broker Architecture

- **Kex[5]-test:** server, na kterém se vyvíjí a testuje samotná funkcionalita webu včetně testování vyvíjených SDK balíčků. Po odladění chyb se aplikace přesouvá na Kex-stage.
- **Kex-stage:** primární server pro vývoj, kde se nachází RC[6] MCUXpresso SDK Builder na uvolnění na produkční server. Zde jsou odchyceny a opraveny poslední nedostatky.
- **Kex-sandbox a kex-demo:** jsou servery pro experimentování s SDK generátorem balíčků.
- **Kex-nxp:** produkční server dostupný pro koncové zákazníky, který prošel celým procesem testovaní na jednotlivých serverech [24].

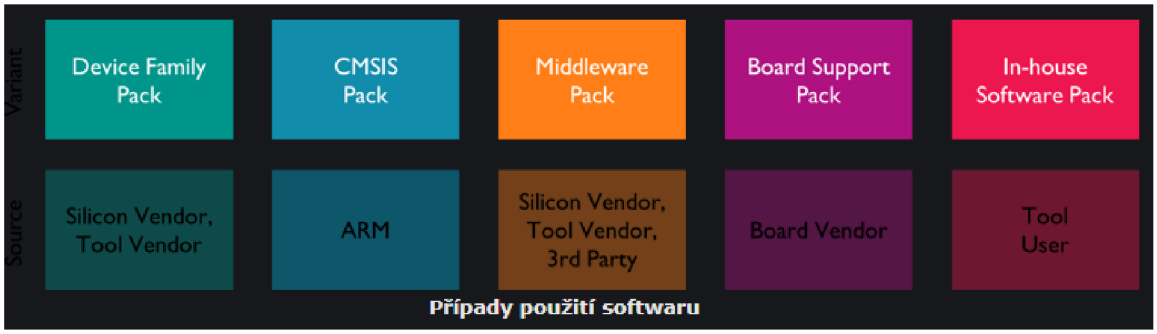
SDK balíčky nejsou většinou přímo předpřipravené pro stažení. Existuje okolo 15 miliónů kombinací balíčků podle zadané specifikace. Specifikace se skládá:

- ze zvoleného zařízení, pro které má být balíček vytvořen,
- z operačního systému, na kterém bude vyvíjeno a s tím spojené vývojové prostředí,
- ze seznamu volitelných middleware položek.

Pro každou kombinaci předem zmíněných specifikací existují příslušné komponenty. Oproti CMSIS balíčkům si uživatel zvolí specifikace a stáhne si jediný SDK balíček, ve kterém nalezne vše, co požadoval.

2.3.2 CMSIS balíčky

Softwarový balíček vytvořený firmou NXP s formátem CMSIS (Cortex Microcontroller Software Interface Standard) je rovněž jako SDK balíček sada vývojových nástrojů, ale primárně určená pro vývojové prostředí uVision firmy ARM. Oproti SDK balíčkům umožňují CMSIS balíčky konzistentní podporu zařízení k procesoru a jeho periferním zařízením [2]. CMSIS balíčky jsou volně dostupné na stránkách ARMu [4].



Obrázek 2.1: Použití softwarových CMSIS balíčků [1]

CMSIS-Packs je formát balíčku pro distribuci softwarových komponent. Tento formát je součástí specifikace CMSIS, vyvinutý firmou ARM. Softwarové komponenty jsou snadno volitelné a všechny závislosti na jiném softwaru jsou zvýrazněny. Typy softwarových CMSIS balíčků je možné vidět na obrázku 2.1. Horní vrstva představuje typy balíčků a dolní vrstva subjekt, který daný typ balíčku produkuje.

[5]KEX – Kinetis Expert
[6]RC – Release Candidate

Softwarové komponenty obsahují knihovny, zdrojové moduly, konfigurační soubory, šablony zdrojového kódu a dokumentaci. Softwarové komponenty mohou být obecné pro podporu široké škály zařízení a aplikací.

Dříve se softwarové moduly těžko integrovaly, protože zdrojové a hlavičkové soubory měly nejasné požadavky, nekonzistentní dokumentaci nebo chybějící informace o licenci. Z tohoto důvodu bylo jednodušší vytvořit formát `CMSIS` oproti generátoru `SDK` balíčků s konkrétní specifikací. I do dnes `CMSIS` formátu zůstaly některé výhody oproti `SDK` balíčkům. Použití softwarových `CMSIS` balíčků je následující [1]:

- **Balíček Device Family Pack (DFP):** obsahuje soubory systému `CMSIS` / spouštěcí soubory, ovladače a algoritmy flash pro rodinu zařízení s mikrokontrolerem.
- **CMSIS Software Pack:** obsahuje generické komponenty `CMSIS` (`CORE`, `DSP Library` a implementace `RTOS`) dodávané firmou `ARM` a také mohou obsahovat softwarové komponenty patřící do middlewaru (například zdrojový kód nebo knihovny).
- **Board Support Pack (BSP):** obsahuje dokumentaci, schémata a ovladače pro určité vývojové desky. V případě balíčků firmy `NXP` obsahují navíc ukázkové projekty pro danou desku.
- **Vnitřní softwarové balíčky:** obvykle obsahují softwarové komponenty, které lze distribuovat v rámci společnosti nebo technické skupiny.

2.4 Porovnání softwarových balíčků

V této sekci se podíváme na možné způsoby řešení porovnání různých typů souborů.

`CMSIS` a `SDK` balíčky jsou komprimované soubory do jednoho `.ZIP`[7] souboru. Každý typ balíčku má rozdílnou souborovou strukturu, ale hlavním problémem při porovnávání balíčků je obsah souborů. Softwarový balíček může obsahovat binární, textové nebo specificky strukturované soubory.

Hlavním důvodem, proč porovnávat softwarové balíčky je vidět rozdíly při zavedení změn:

- V datech, ze kterých se softwarové balíčky generují,
- v samotném procesu generování balíčků,
- v souborech třetích stran a souborech vyvíjených firmou `NXP`.

Dále při porovnání dvou verzí softwarových balíčků je možné určit, zda se jedná o změny chtěné, nebo se někde v procesu tvorby softwarových balíčků vyskytla chyba.

V případě nalezení souboru v softwarovém balíčku, který nemá svého bratra v druhém, právě porovnávaném balíčku, lze označit za chybějící soubor a jedná se o rozdíl mezi softwarovými balíčky.

Textové soubory

Textové soubory lze porovnat nástrojem `diff`, který nalezneme na všech operačních systémech unixového typu. Nástroj `diff` vznikl v roce 1974. Tento nástroj samozřejmě prošel

[7] ZIP – souborový formát pro kompresi a archivaci dat

poměrně dlouhým vývojem a různými rozšířeními, ať se to již týká vlastního interního algoritmu pro hledání rozdílů, tak i formátů výstupu, tj. způsobů, jakým `diff` zobrazuje rozdíly mezi porovnávanými soubory.

Není tedy nutné hledat další alternativy nástroje `diff`. Pomocí parametrů lze upravovat formát výstupu, ale i způsob porovnávání.

Rozdíly mezi strukturovanými a nestrukturovanými textovými formáty:

- Strukturované textové soubory mají stromovou logickou strukturu,
- strukturované textové soubory mohou mít zaměnitelné pořadí elementů a atributů,
- ve strukturovaných textových souborech nezáleží narozdíl od nestrukturovaným na tom, co se nachází na konkrétních řádcích.

Binární soubory

Binární soubor je soubor, který obsahuje z většiny data v nečitelné podobě. Může se jednat o data v binární podobě jedniček a nul, ale také o text s `ne-ASCII`[8] hodnotami a nečitelnými znaky. V binárních souborech se dále často vyskytuje znak s `ASCII` hodnotou `\x00`.

Na základě výše zmíněného popisu binárních souborů jsou tyto soubory detekovány a porovnávány velikosti souborů bajt po bajtu. Podrobněji popsáno v sekci 5.4.

2.4.1 XML porovnávání

XML soubory nelze porovnávat obvyklým způsobem i když se jedná o textové soubory. XML soubory reprezentují stromové datové struktury, tudíž se jedná o problémem v oblasti porovnávání grafů. Detekce změn ve stromových strukturách lze rozdělit na porovnávání seřazených stromů a neseřazených stromů.

Seřazeným XML stromem je myšlen strom, kde všechny listy(děti) jsou seřazeny abecedně podle názvu dítěte pro každého otce. Atributy všech elementů jsou rovněž abecedně seřazeny podle názvu atributu a v případě více hodnotového atributu jsou i hodnoty seřazeny abecedně. V neseřazených stromech na rozdíl od seřazených stromů nezáleží na horizontálních vztazích mezi sourozenci.

Pro porovnávání `XML` souborů existují algoritmy pro seřazené i neseřazené stromy. Pro neseřazené stromy je to například `X-Diff`, zatímco pro sežazené `XyDiff` [8] nebo `X-Tree Diff+` [21].

Při výběru algoritmu pro tuto práci byly rozhodující dva faktory: rychlost porovnání a spolehlivost výsledku. Při rozhodování, který z algoritmů bude použit, byla použita následující úvaha:

- Práce je vyvíjená jako regresní testování pro interní využití ve firmě, proto by výsledky měly být pokud možno co nejspolehlivější, přičemž kvalita grafické reprezentace výsledků není tolik důležitá,
- když regresní testy budou pravidelně testovat gigabajty dat, je předpoklad, že `XML` soubory budou komplexní a většího rozsahu,
- zároveň je důležité bezchybně zjistit rozdíly mezi soubory.

[8] ASCII – American Standard Code for Information Interchange

Záležet bude tedy jak na rychlosti, tak na pokud možno 100% přesnosti. Z porovnání algoritmů `X-Tree Diff+`, `XyDiff` a `X-Diff` [17] s ohledem na tyto faktory vyplývá, že ideální možnost bude použít algoritmus `X-Tree Diff+`.

2.4.2 Dostupné metody

Algoritmus `X-Tree Diff+` by měl být velmi efektivní v rychlosti s časovou složitostí `O(n)` a využití paměťového prostoru. `X-tree Diff+` používá speciálně uspořádaný označený strom, `X-tree`, který reprezentuje `XML` / `HTML`[9] dokumenty. Uzly `X-tree` mají speciální pole `MD`, které ukládá 128-bitovou hodnotu `hash` reprezentující strukturu a data podstromů, takže shodné dílčí podstromy tvoří staré a nové verze [17].

`X-Diff+` vypočítává vzdálenost pro úpravy mezi dvěma dokumenty. Během tohoto procesu `X-tree Diff+` používá pravidlo zpoždění dvojznačných shod, což znamená, že provádí přesné párování, kde má uzel ve staré verzi jedna ku jedné shodu s odpovídajícím novým uzlem. Výrazně snižuje možnost špatného párování.

Časové náklady `X-Diff+` nejsou přijatelné pro aplikaci v reálném čase. `X-tree Diff` je primárně určen k detekci útoků hackerů na webové stránky. `X-tree Diff` je tedy jednoduchý a běží rychle v `O(n)`. `XyDiff` a `X-tree Diff` jsou poměrně účinné v době provádění. Kvalita porovnaných `XML` souborů, které produkují, však není dost dobrá.

Zatímco heuristický algoritmus `XyDiff` dosahuje složitosti `O(n log n)`, díky čemu se stává nejnáročnějším algoritmem a zároveň nesplňuje 100% spolehlivost. Je určen pro `XML` datové sklady a verzování. Tento algoritmus používá hashování[10] k reprezentaci obsahu a struktury podstromů, a pojem vážení v procesu porovnávání. Spolehlivost výsledku z porovnání `XML` souborů podle algoritmu je vyobrazena na obrázku 2.3.

```
@@ -7337,10 +7337,10 @@
</component>
<component devices="LPC54005" id="docs.images.LPC54005" name="images" type="other" user_visible="true" version="1.0.0">
    <source path="docs/images" type="image">
-       <files mask="LPCXpresso54018.gif"/>
+       <files mask="device.jpg"/>
    </source>
    <source path="docs/images" type="image">
-       <files mask="device.jpg"/>
+       <files mask="LPCXpresso54018.gif"/>
    </source>
</component>
<component devices="LPC54005" id="platform.Include_common.LPC54005" name="Include_common" type="CMSIS_Include" user_visible="true" version="5.0.1">
@@ -7524,10 +7524,10 @@
</component>
<component devices="LPC54018" id="docs.images.LPC54018" name="images" type="other" user_visible="true" version="1.0.0">
    <source path="docs/images" type="image">
-       <files mask="LPCXpresso54018.gif"/>
+       <files mask="device.jpg"/>
    </source>
    <source path="docs/images" type="image">
-       <files mask="device.jpg"/>
+       <files mask="LPCXpresso54018.gif"/>
    </source>
</component>
<component devices="LPC54018" id="docs_external.LPC54018.LPC54018" name="LPC54018" type="documentation" user_visible="true" version="1.0.0">
```

Obrázek 2.2: Porovnávání elementů podle obsahu atributů podelementů

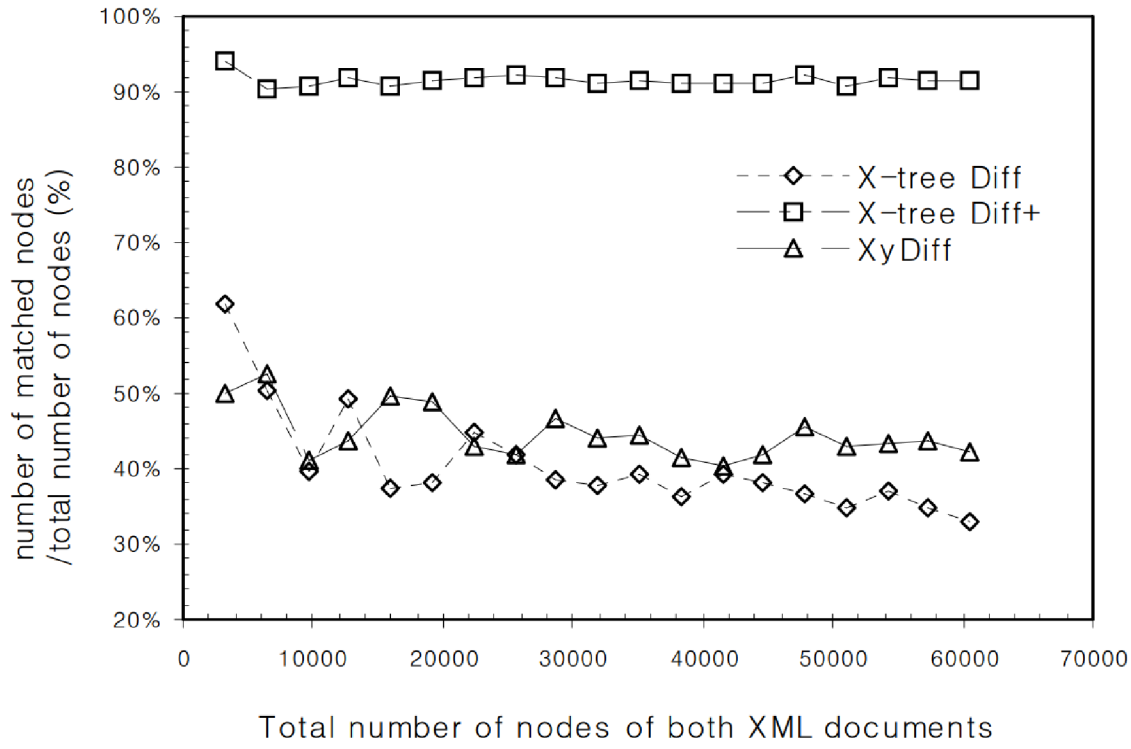
Při porovnávání stromových struktur není vždy zřejmé, která část prvního stromu odpovídá části druhého stromu. Tento problém algoritmu `X-Tree Diff+` je vyobrazen na Obrázku 2.2.

Element `component` má v obou `XML` stromech dvě děti se jménem `source`. Není ale jasné, který `source` odpovídá kterému, jelikož v `XML` nezáleží na pořadí elementů. Náš algoritmus seřadil správně názvy elementů a atributů, ale zdá se, že to v některých případech nestačí.

[9] HTML – Hypertext Markup Language

[10] hashování – převádění vstupních hodnot na jeho otisk v podobě čísla (hash)

Pokud jsou některé elementy a další vnořené podelementy na stejné stromové úrovni shodné, nedokáže si algoritmus poradit s jejich uspořádáním.



Obrázek 2.3: Odpovídající poměr každého algoritmu[17]

2.4.3 Algoritmus X-Tree Diff+

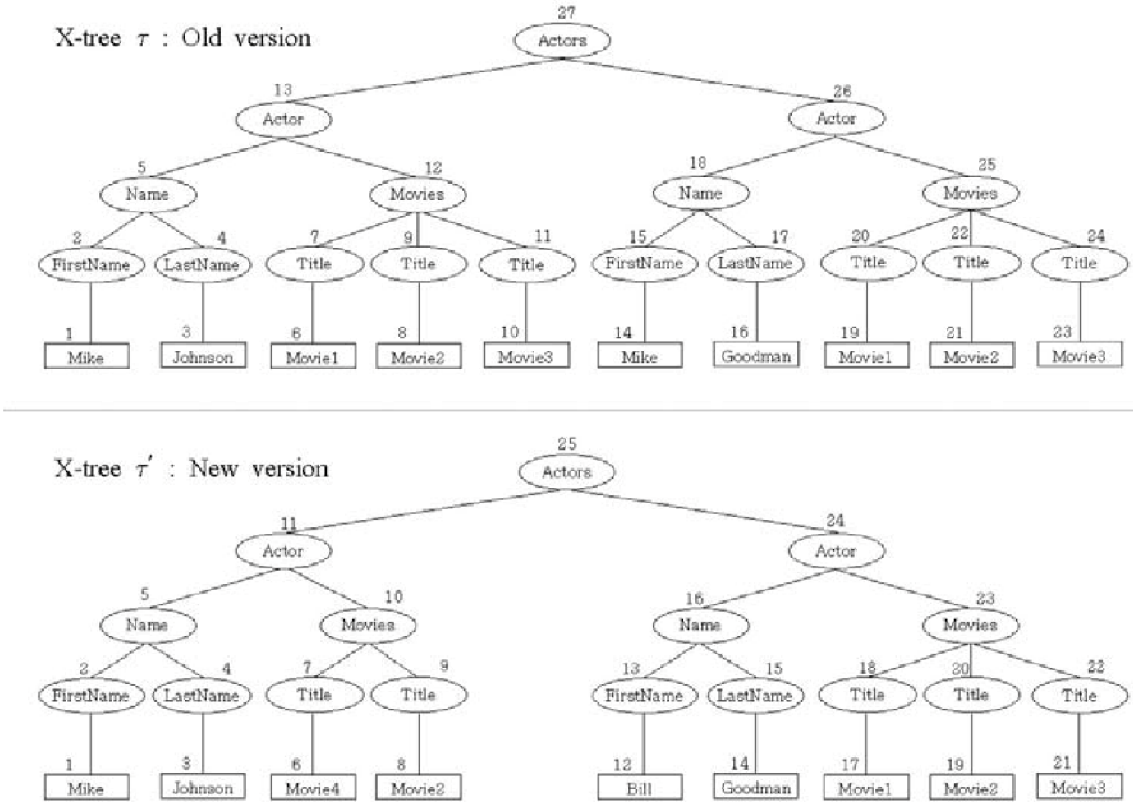
V této části je vysvětlen algoritmus `X-tree Diff+`. Na obrázku 2.4 je `X-Tree` τ stará verze, a `X-tree` τ' nová verze. Číslo vedle uzlu těchto `X-Tree` představuje pořadí návštěvy uzlu. Tato čísla jsou zvyklá identifikovat každý uzel ve stromu. Například τ_1 představuje nejlevější uzel v τ.

Předběžné zpracování

Samotná implementace se skládá z několika částí [17].

Krok 0 (sestavení X-tree): Převedeme `XML` dokumenty do `X-tree` τ a τ', a vygenerujeme `hash` tabulky. Během tohoto procesu jsou všechna pole uzlů `X-tree` správně inicializována. Kromě toho jsou pro každý uzel `X-tree` `nMD`, `tMD` a `iMD` vypočítány a uloženy hodnoty v polích `nMD`, `MD` a `iMD` navzájem.

Jsou generovány tři tabulky `hash` `O_Htable`, `N_Htable` a `N_IDHtable`. Záznamy pro první dvě hašovací tabulky jsou n-tice tvořené `MD`, ukazatelem `X-tree` a uzlem s klíčem `MD`. Všechny uzly s jedinečnou hodnotou `MD` v τ' jsou registrovány `N_Htable`, zatímco všechny uzly s ne jedinečným `MD` v τ jsou registrovány `O_Htable`. `N_IDHtable` je `hash` tabulka pro uzly s atributy `ID` v `X-tree` τ'. Položka se skládá z `iMD` (jako klíče) uzlu a ukazatele na uzel.



Obrázek 2.4: Příklad X-tree[17]

Krok 1 (Shoda identických podstromů s korespondencí 1:1 a shodnými uzly s ID atributy): Jako první najde algoritmus opakovaně pár identických podstromů. Jeden z τ a další z τ' a porovná je pomocí `NOP`[11]. Algoritmus na obrázku 2.5 popisuje tento proces. `N` a `M` představují kořeny podstromů v τ a τ', resp. Na konci algoritmus vypočítá seznam shod, nazvaný `M_List`, což jsou vstupní data pro krok 2. Za povšimnutí stojí, že po nalezení shody algoritmus nevchází do jejich podstromů pro párování. A to právě kvůli charakteristikám `MD`. Shody nalezené v tomto procesu mají vzájemnou korespondenci.

Po dokončení předchozího dílčího kroku se algoritmus pokusí spojit uzly se stejnými hodnotami `iMD`. Při přechodu `X-tree` τ do prvního řádu, je-li nalezen neshodný uzel s ID atributem, vyhledá se položka v `N_IDHtable` se stejnou hodnotou `iMD`. Pokud je vyhledávání úspěšné, položka je porovnána s `NOP`.

Na obrázku 2.5 jsou u obou $ST(\tau_5)aST(\tau_{17})$ jedinečné `X-tree` τ, a mohou být porovnány s $ST(\tau'_5)aST(\tau'_{15})$ `X-tree` τ'. Jsou vytvořeny shody jako $ST(\tau_5) \rightarrow ST(\tau'_5)$ a $ST(\tau_{17}) \rightarrow ST(\tau'_{15})$. Současně, jeden z $ST(\tau_{12})$ a $ST(\tau_{25})$ může být přiřazen do $ST(\tau'_{23})$. Protože však $ST(\tau_{12})$ a $ST(\tau_{25})$ nejsou jedinečné v `X-tree` τ, nelze v tomto kroku určit tuto shodu.

Krok 2 (Šíření shody nahoru): V tomto kroku se rozšiřují shody nalezené v kroku 1 nahoru ke kořenům. Pro každou shodu $A \rightarrow B$ určenou v kroku 1, se musíme rozhodnout, zda rodič `(P (A))` může být porovnán s `(P (B))`, na základě jejich štítků. Algoritmus je znázorněn na obrázku 2.6. V příkladu znázorněném na obrázku 2.4 se povyšuje odpovídající

[11]NOP – Nulová operace

```
For each node N in τ { /* Visit each node of τ in breadth-first order */
  If any entry of O_Htable does NOT have the same tMD value that the node N has then {
      If some entry of N_Htable has the same tMD value that the node N has then {
          Retrieve the node M from N_Htable;  Match the nodes N and M using NOP;
          Add the pair (N, M) of nodes to M_List;
          Stop visiting all the subtrees of the node N, then go on to next node in τ ;  }
      Else Go on to next node in τ ;  }
  Else Go on to the next node in τ ;
} // For
```

Obrázek 2.5: Odpovídající identické podstromy s jednostrannou korespondencí[17]

$\tau_5 \rightarrow \tau'_5$ nalezené v kroku 1 nahoru. Protože jejich nadřazené uzly mají stejné označení odpovídající v tomto kroku $\tau_{13} \rightarrow \tau'_{11}$. Opět platí, že mateřské uzly uzly v $\tau_{13} \rightarrow \tau'_{11}$ mají stejné označení a poté se určí odpovídající $\tau_{27} \rightarrow \tau'_{25}$. Podobným způsobem jsou zpracovány shody $\tau_{17} \rightarrow \tau'_{15}$ z kroku 2, $\tau_{18} \rightarrow \tau'_{16}$ a $\tau_{26} \rightarrow \tau'_{24}$.

```
/* Propagate each matching from M_List to its parents */
For matching (A, B) in M_List from Step 1 and 2 {
  pA = Parent(A); pB = Parent(B)
  While TRUE {
      /* None of parents have been matched. */
      If nPtr(pA) == NULL AND nPtr(pB) == NULL then {
          If Label(pA) == Label(pB) then {
              Match pA and pB using NOP;  pA = Parent(pA);  pB = Parent(pB);}
          Else Break;       }
      Else Break;
  } // While
} // For
```

Obrázek 2.6: Šíření shody nahoru (Backtracing)[17]

Krok 3 (Shoda zbývajících uzlů směrem dolů): V tomto kroku, směrem dolů od kořenů, se pokouší komunikovat s uzly, které zůstávají unikátní, dokud krok 3 nezačne. Při návštěvě uzlů v τ v pořadí první hloubky opakujeme následující:

(1) Najděte odpovídající uzel A τ, který má unikátní děti. Nechť `B` je `M (A)`, odpovídajícího uzlu `A`. Pro `A` a `B`, nechť `cA [1..s]` označuje seznam unikátních dětí `A` a `cB [1..t]` seznam unikátních uzlů `B`, kde s je také počet definovaných unikátních uzlů `A` a `t`.

(2) Pro `A`, `B`, `cA [1..s]` a `cB [1..t]` proveďte algoritmus na obrázku 2.7. Při provádění kroku 3 se používají dvě tabulky `hash`. Je-li v uzlu `A` nalezen τ, všechny unikátní uzly `cA [1..s]` jsou zapsány do obou `hash` tabulek, kde jedna hašovací tabulka používá hodnoty `MD` uzlů jako klíč a ostatní hodnoty odkazu a indexu jako klíč. Tyto hašovací tabulky používají k nalezení shody pro každou z nich unikátní uzel `cB[j]` uzlu `B` τ'.

```
For j in [1..t] {    /* For each cB[j] */
  If there is a cA[i] where tMD(cA[i]) = tMD(cB[j]) then
       Match ST(cA[i]) with ST(cB[j]) using NOP;
  Else If there is a cA[i] where Label(cA[i]) = Label(cB[j]) and Index(cA[i]) = Index(cB[j]) then
       Match cA[i] with cB[j] using NOP;
  }    } // For
```

Obrázek 2.7: Porovnání zbývajících uzlů směrem dolů[17]

Krok 4 (Vyladit existující shody): V kroku 4 se analyzuje kvalita shody a přizpůsobuje některé neefektivní shody. Kvalita shody pro uzel `N` je analyzována z pohledu toho, jak hodně je shoda `N` konzistentní se shodami dětí `N`. Pro uzel `N`, definujeme počet ($P\#$) pozitivních dětských shod, počet ($N\#$) negativních dětských shod a stupně konzistence shody $Con\#(N)$ takto:

- $P\#(N)$ = počet dětí C_k vyhovujících $[P(M(Ck(N)) = M(N)]$
- $N\#(N)$ = počet dětí C_k vyhovujících $[P(M(Ck(N)) \neq M(N)]$
- $Con\#(N) = P\#(N)$ / počet shodných dětí $= P\#(N)/(P\#(N) + N\#(N))$

$P\#(N)$ implikuje počet dětí, které vyhovují tomu, že rodič shodného uzlu potomka `N` je roven odpovídajícímu uzlu `N`, zatímco $N\#(N)$ ukazuje počet dětí, které nesplňují stejnou podmínku. $Con\#(N)$ implikuje zlomek dětí, které přispívají pozitivně k aktuálním shodám `N`. Abychom zamezili neefektivním shodám, musíme znát alternativní shody pro uzel. Seznam alternativních shod pro uzel `(IAM)` je definován následovně.

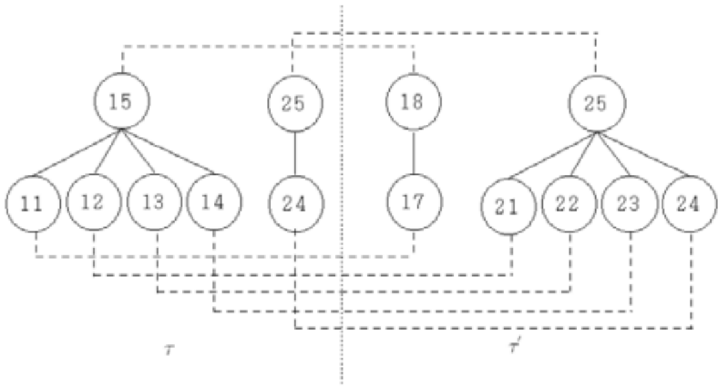
$$IAM(N) = \{< K, \text{počet dětí } C_i > | Label(N) = Label(K) \wedge K \neq M(N) \wedge K = P(M(Ci(N)) \text{ pro některé dítě } C_i(N)\}$$

S `IAM(N)` je možné najít uzel, který bude shodný s uzlem `N`, a také počtem dětí, který pozitivně ovlivňuje. Proces ladění začíná přechodem τ v pořadí. Algoritmus je uveden na obrázku 2.8.

```
For each node N in τ                    // traverse in post-order
 If Con#(N) < 0.5 then {
   Compute lAM (N);
   If fV(N) > P#(N) + P#( fN(N)) then
     {Revoke the current match of N; Revoke the current match of fN(N); Match N with fN(N);}
   }
```

Obrázek 2.8: Algoritmus pro krok 4[17]

Obrázek 2.9 ukazuje části dvou `X-tree` stromů. Odkryté čáry představují existující shody. Tabulky ukazují $P\#$, $N\#$, $Con\#$ pro uzly τ_{15} a τ_{25}. Všimněte si, že $Con\#(\tau_{15}) < 0.5$. Vzhledem k tomu, že z $IAM(\tau_{15})$, $fN(\tau_{15}) = \tau'_{25}$ a $fV(\tau_{15}) = 3 > P\#(\tau_{15}) + P\#(\tau_{25})$, nahradíme aktuální τ_{15} shodu $(\tau_{15} \rightarrow \tau'_{18})$ za novou $(\tau_{15} \rightarrow \tau'_{25})$.



Data of τ_{15} for tuning

$P\#(\tau_{15})$	1
$N\#(\tau_{15})$	3
$Con\#(\tau_{15})$	1/4 = 0.25

Data of τ_{25} for tuning

$P\#(\tau_{25})$	1
$N\#(\tau_{25})$	0
$Con\#(\tau_{25})$	1/1 = 1.00

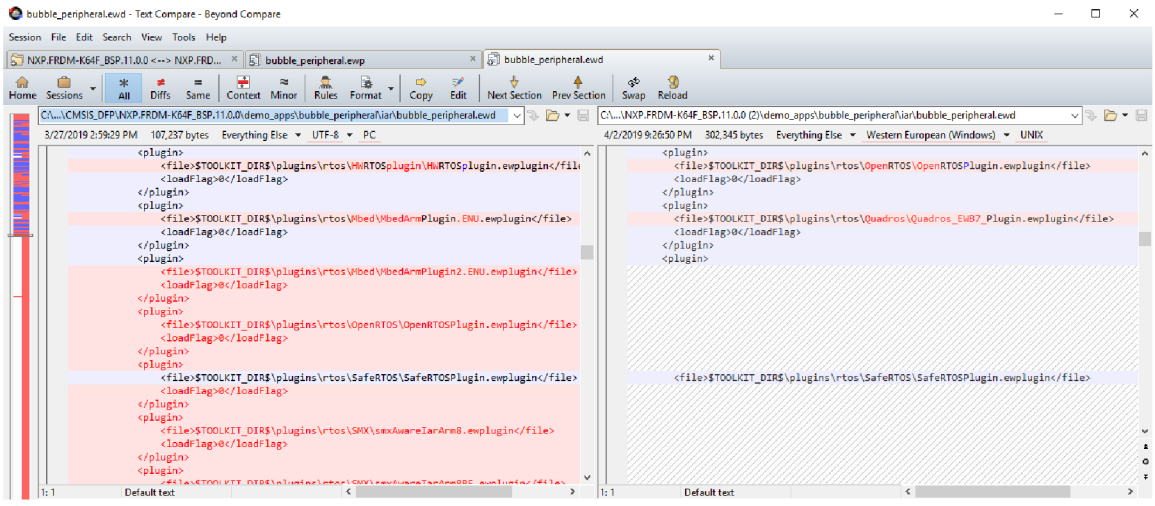
Obrázek 2.9: Příklad kroku 4 (ladění existujících shod)[17]

2.5 Analýza dostupných nástrojů

V současnosti existuje spousta nástrojů, které umožňují porovnávat strukturované soubory `XML` a občas jsou některé sofistikovanější a konfigurovatelnější, než je rozsah této práce.

S netriviálními `XML` soubory si ale většina těchto porovnávačů neporadí. Program `Beyond Compare` jako jeden z mála si dokáže poradit s jakýmkoliv `XML` po vhodném nastavení. Na obrázku 2.10 můžete vidět příklad porovnání pomocí tohoto nástroje. Má to ale i své nevýhody. `Beyond Compare` je placený program a lze jej ovládat pouze přes `GUI` rozhraní.

Mým cílem a zároveň motivací bylo vytvořit `XML` porovnávač pro rozhraní příkazové řádky za účelem integrace do regresních testů a současně, aby si poradil se všemi možnými situacemi, které mohou v `XML` nastat. Neméně důležité je, aby zvládal nadstandardní požadavky podrobně popsané v sekci 2.4.1.



Obrázek 2.10: Porovnání XML souborů pomocí Beyond Compare

Kapitola 3

Návrh automatických regresních testů

V této kapitole jsou popsány požadavky práce. Kapitola je rozdělena do čtyř částí. V první části jsou popsány obecné požadavky na práci. V dalších částech je popsán návrh uživatelského vstupního rozhraní, struktura regresních testů a grafický návrh výstupního rozhraní.

3.1 Požadavky práce

Požadavky na práci můžeme rozdělit do dvou kategorií: požadavky na funkcionalitu a na grafickou reprezentaci dat. Základní požadavky na funkcionalitu byly primárně definované zadavatelem práce, zatímco grafickou reprezentaci dat bylo nutné vymyslet a navrhnout podle zjištěných a požadovaných výsledků do přehledného rozhraní.

V tomto projektu je cílem vytvořit grafické rozhraní pro výsledky porovnání `SDK` a `CMSIS` balíčků a výsledky `IDE` testů balíčků z vývojových prostředí `uVision`, `IAR Workbench` a `MCUXpresso IDE`.

3.2 Uživatelské vstupní rozhraní

Uživatelské vstupní rozhraní je grafické. Všechna nastavení automatizovaných regresních testů se provádí přes `Atlasian Bamboo` 4.2.1. Uživatel se přihlásí přes `Atlasian` účet a vyhledá požadované regresní testy. Struktura regresních testů je popsána v kapitole 3.3.

Při testování `SDK` balíčků se jako první tyto balíčky stahují z webové stránky `MCUXpresso` [24] pomocí takzvaných zadních vrátek. Uživatel si může vybrat mezi 5 produkčními servery, ze kterých se dané balíčky stáhnou 2.3.1. Balíčky, které budou staženy, uživatel nastaví ve vstupní konfiguraci. Blíže popsáno v sekci 3.5.

`CMSIS` balíčky nevyžadují systém zadních vrátek, proto se při testování předává pouze cesta ke složce, kde se testované balíčky nachází.

Při `IDE` testu si uživatel dále může zvolit, v jakém vývojovém prostředí se budou balíčky testovat. Na výběr jsou nyní dostupná testovací prostředí `uVision`, `MCUXpresso IDE` a `IAR Workbench` blíže popsané v sekci 4.3.

U porovnávacího testu je možnost nastavení rozsáhlejší a do budoucna rozšířitelná pro filtrování informací. Je potřeba zvolit dvě složky, ve kterých se budou vyskytovat stejně pojmenované balíčky, které se následně budou párovat podle jména balíčku a následně

porovnávat. Další povinnou informací je zvolit o jaký typ balíčku se jedná, protože pro `CMSIS` a `SDK` jsou ve výstupním rozhraní rozdílné informace.

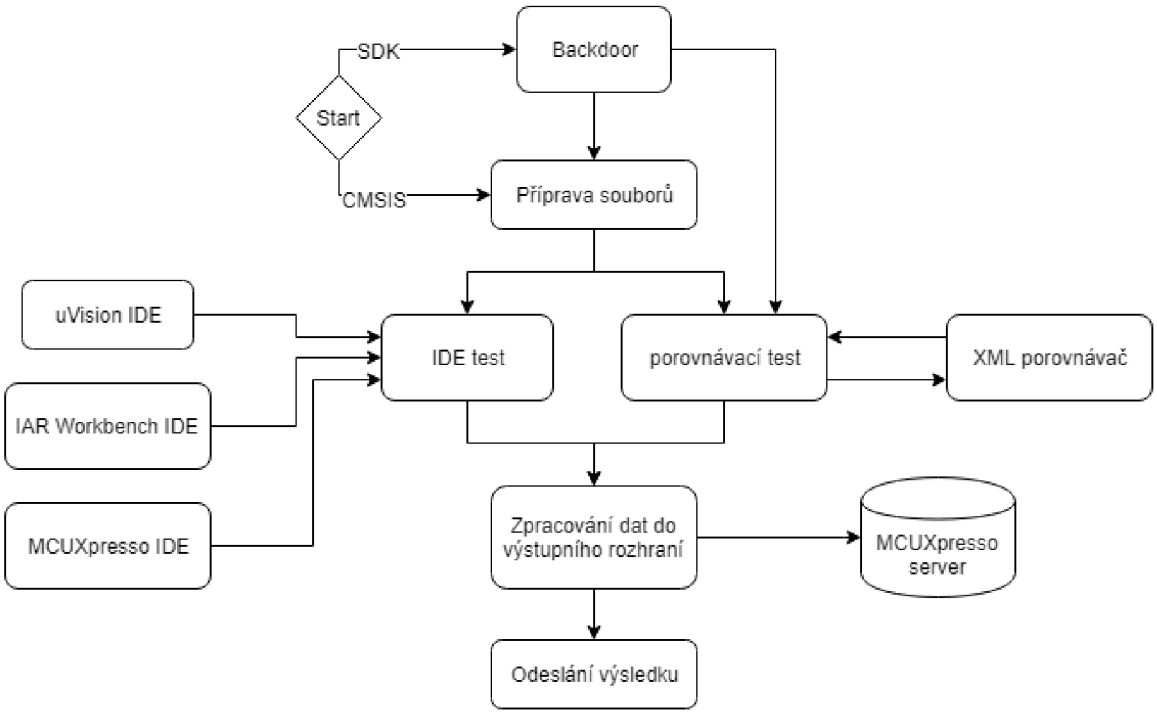
Volitelně lze specifikovat, které atributy a elementy se budou ignorovat při porovnávání strukturovaného souboru založeném na `XML` formátu. Dále lze ignorovat celé soubory na základě jejich jména.

Poslední volitelnou specifikací je ignorování bílých znaků. Při testování bylo zjištěno, že soubory mohou obsahovat rozdílné konce řádků a nemusí se jednat o chybu. Z tohoto důvodu byla zavedena tato specifikace.

3.3 Struktura regresních testů

Regresní testy jsou rozděleny na samostatné projekty, které jsou součástí jedné testovací sady zvlášť pro `CMSIS` a `SDK` balíčky. Projekty jsou nastaveny pomocí triggerů tak, aby se spouštěly po úspěšném ukončení předešlého projektu v daném pořadí. Celý systém automatizovaných regresních testů pro balíčky `SDK` a `CMSIS` je nastaven v nástroji na kontinuální integraci a nasazení `Bamboo` 4.2.1.

Celý koncept struktury regresních testů je navržen tak, aby mohlo současně běžet více testů ve stejnou dobu. K tomuto jsou využiti `Bamboo agenti` 4.2.1. Dále je zamezeno, aby se prováděly duplicitní operace, jako je stahování softwarových balíčků pro každý test zvlášť. Navržení takové vhodné struktury regresních testů před samotnou implementací zkrátí čas doby vývoje i zlepší samotnou optimalizaci regresních testů. Struktura regresních testů pro `CMSIS` a `SDK` balíčky je graficky znázorněna na obrázku 3.1.



Obrázek 3.1: Schéma struktury regresních testů

Při testování `SDK` balíčků se vždy jako první spouští stahování softwarových balíčků z webové stránky `MCUXpresso` [24] pomocí zadních vrátek 3.5. Při neúspěšném stáhnutí

balíčku se zaznamená chyba stažení a testování bude pokračovat bez tohoto balíčku. Při dokončení stahování se stažené balíčky připraví pro následující testy.

Stažené softwarové balíčky se předají IDE testu 3.3.1 a porovnávacímu testu 3.3.2, které se zpracovávají souběžně. Zde se využilo již zmíněných `Bamboo` agentů 4.2.1.

3.3.1 IDE test

`IDE test` používá softwarové balíčky k vytvoření projektů ve vývojovém prostředí. K dispozici jsou vývojová prostředí `uVision` 4.3.1, `MCUXpresso IDE` 4.3.2 a `IAR Workbench` 4.3.3. Vytvořený projekt se přeloží a sestaví a v případě chyby se výsledek zaznamená. Uchovávání dat z testování je podrobněji popsáno v sekci 5.3.

3.3.2 Porovnávací test

Porovnávací test dostává na vstupu dvě sady stejných softwarových balíčků s rozdílnými verzemi. Porovnávají ze vždy balíčky se stejným jménem z každé sady, mezi starou a nově vygenerovanou sadou.

Při porovnávání `SDK` balíčků dostane porovnávací test od `IDE testu` první sadu balíčků. Druhá sada balíčků určená k porovnání se musí stáhnout pomocí zadních vrátek 3.5 z jiného produkčního severu, než sada první, stažená pro `IDE test` 3.3.1.

Využívá se zde `XML` porovnávače, který porovná dva strukturované soubory typu `XML` mezi sebou. Podrobněji vysvětleno v sekci 2.4.1. Výstupem je textový soubor s nalezenými rozdíly.

3.4 Výstupní rozhraní

Zpracování dat z předešlých testů `IDE test` 3.3.1 a porovnávací test 3.3.2 do výstupního rozhraní může probíhat rovněž souběžně a je spuštěno po jejich dokončení. Vstupní data mohou mít velikost i několik gigabajtů rozložených ve více než desetitisících souborech. Proto zde byla důležitá optimalizace a rychlá práce se soubory. K tomuto účelu byly využity regulární výrazy a interpret příkazů `Shell` pro rychlou práci se soubory. Více detailů ohledně implementace je podrobněji popsáno v kapitole 5.

Reprezentace dat hraje důležitou roli při každodenním procházení výsledků regresních testů. Postupné procházení rozdílů každého souboru je někdy nereálné. Je důležité nejdříve informovat čtenáře o souhrnných informacích, týkajících se konkrétního testu. Jako příklad je zde použito výstupní rozhraní porovnávacího testu na obrázku A.1, kde byly použity souhrnné informace:

- Jaké softwarové balíčky byly porovnávané,
- s jakým nastavením se testy spouštěly,
- počet testovaných balíčků a souborů,
- počet rozdílných testovaných balíčcích a souborů,
- souhrn změn pro každý softwarový balíček.

Pro reprezentaci dat byl využit značkovací jazyk `HTML`. Hlavními důvody použití jazyka `HTML` byly:

- Větší možnosti grafické editace v emailovém klientovi,
- jednoduchá definice grafického stylu emailu,
- možnost propojení emailového klienta se síťovým diskem přes hypertextové odkazy,
- zobrazení regresních testů v libovolném webovém prohlížeči.

Vzhled výstupního rozhraní regresních testů není až tolik důležitý oproti struktuře nebo získané informaci z testování. Přesto vzhled výsledků testů, vhodného nastavení barev, odsazení a fontů může zpřehlednit výstupní rozhraní. Na základě získaných informací od uživatelů výstupního rozhraní regresních testů jsem navrhl styl, který jsem aplikoval pro všechny výsledky testů.

Kvůli osobním preferencím každého jednotlivce jsem zvolil použití tmavého vzhledu v příloze na obrázku A.1 a světlého vzhledu v příloze na obrázku A.3.

3.4.1 Výstupní rozhraní porovnávacího testu

Výstupní rozhraní porovnávacího testu se dělí na dvě části. První část je posílána emailem a druhá ukládána na vzdálený server.

První část porovnávacího testu

První část je odeslána jako email přes `LDAP`[1] server firmy NXP na emailovou adresu příjemce. V příloze na obrázku A.1 je možné vidět první část výstupního rozhraní porovnávacího testu `CMSIS` balíčků s vysvětlivkami. V příloze na obrázku A.3 je pak vidět pro `SDK` balíčky.

Struktura se skládá z názvu testu a data spuštění. Následuje informace o porovnávaných sadách softwarových balíčků. U `CMSIS` balíčků se jedná o jejich verzi a u `SDK` balíčků o názvy produkčních serverů, ze kterých byly staženy zadními vrátky 3.5. Dále vás informuje o nastavení porovnávacího testu 3.3 a souhrnných informacích o výsledcích testu.

V případě chyby při stahování softwarového balíčku ze serveru se zobrazí dodatečná informace o nevygenerovaných balíčcích, kterou je možné vidět v příloze na obrázku A.3. Ve zbylé části emailu jsou vypsány všechny porovnávané softwarové balíčky. Ke každému balíčku jsou na levé straně vypsány dostatečné informace k jednoznačné identifikaci porovnávaných balíčků. Pravá část obsahuje počet rozdílných a chybějících souborů a zároveň slouží jako odkaz na druhou část výstupního rozhraní porovnávacího testu pro zobrazení změn, které se týkají onoho softwarového balíčku.

Druhá část porovnávacího testu

Druhá část výstupního rozhraní porovnávacího testu je webová stránka a zůstává uložena na interním severu firmy `NXP`, na kterém je též spouštěn `Bamboo plán` regresních testů. Proto je možné si druhou část zobrazit pouze po připojení do interní sítě firmy `NXP` přes `VPN`[2].

Struktura začíná stejně jako první část a skládá se z názvu testu, data spuštění a porovnávaných sad. Dále se struktura liší. V příloze na obrázku A.2 je možné vidět druhou část výstupního rozhraní porovnávacího testu `CMSIS` balíčků s vysvětlivkami. Pro `SDK` balíčky je struktura totožná.

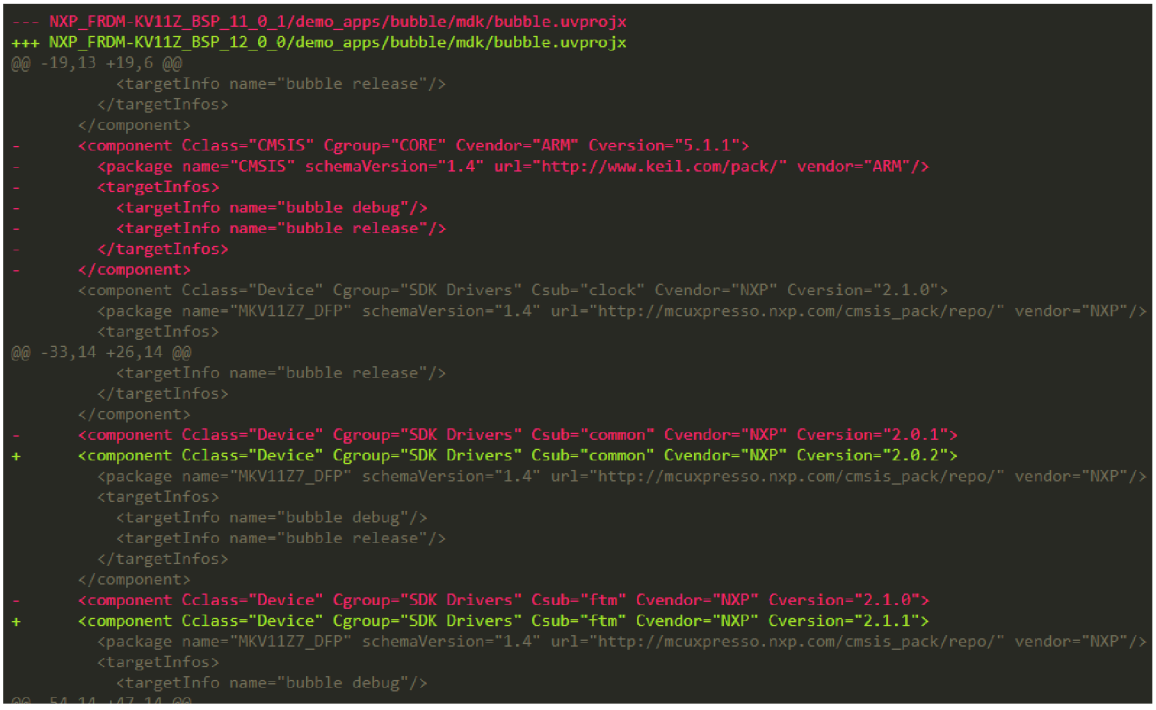
[1] LDAP – Lightweight Directory Access Protocol
[2] VPN – Virtual Private Network

Obrázek 3.2: Výstupní formát porovnaného nestrukturovaného souboru

Za záhlavím se zde nachází stručný popis použití a způsob, jakým lze poznat rozdíly v souborech. Tento popis zde byl zaveden, protože se uživatelé v tomto rozhraní nedokázali zcela vyznat. Časem bylo zjištěno, že pro uživatele je nekomfortní číst dlouhé pokyny použití, proto byly zkráceny a heslovitě v bodech vysvětleny základní orientace v prostředí.

Dále byla zjednodušena reprezentace dat do současné podoby, kterou můžete vidět s vysvětlivkami pro použití v příloze na obrázku A.2.

Rozdílné a chybějící soubory jsou v přehledné tabulce napsány s relativní cestou od kořene softwarového balíčku a velikostí v bajtech. Je možné mezi nimi přepínat tlačítky nebo si je všechny zobrazit v souborové struktuře po kliknutí na název balíčku nad tlačítky.



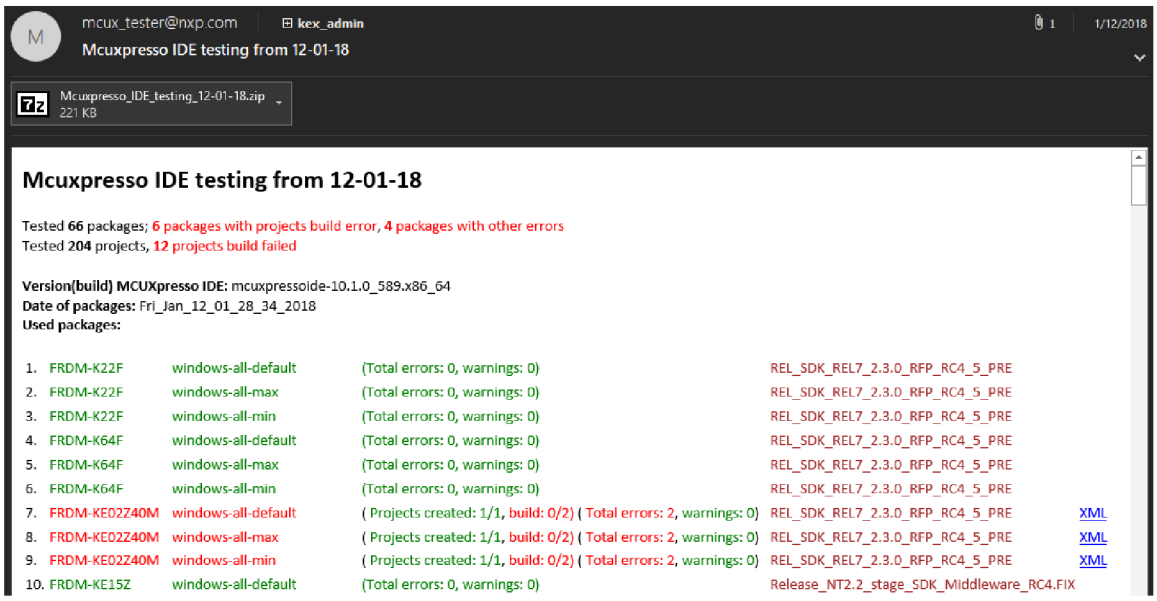
Obrázek 3.3: Výstupní formát porovnaného strukturovaného souboru

Relativní cesty k rozdílným souborům slouží jako odkazy ke zobrazení rozdílů. Soubor s rozdíly mezi dvěma soubory je při procesu zpracování dat do výstupního rozhraní převáděn do HTML jazyka. Bylo zde důležité kvůli čitelnosti barevně rozlišit rozdíly mezi soubory. Vzorem barevného rozložení zde byl textový editor `Sublime` 4.1.2. Ukázku výstupního formátu porovnaného nestrukturovaného souboru je možné vidět na obrázku 3.2.

Při porovnávání `XML` porovnávačem 2.4.1 jsou k rozdílným částem souborů připojeny blízké elementy a podelementy, tak, aby bylo zřejmé, kde se nachází. Ukázku výstupního formátu porovnaného strukturovaného souboru je možné vidět na obrázku 3.3.

3.4.2 Výstupní rozhraní IDE testu

Výstup IDE testu je celý posílán jako email na emailovou adresu příjemce a z bezpečnostních důvodů přes `LDAP` server firmy `NXP`. Na obrázku 3.4 je možné vidět výstupní rozhraní IDE testu `SDK` balíčků. Pro `CMSIS` balíčky je použito stejné rozhraní.



Obrázek 3.4: Výstupní formát IDE testu SDK balíčků

Na začátku výstupního rozhraní se nachází název testu a datum spuštění. Následuje souhrn informací o výsledcích testu, který obsahuje počet:

- Testovaných softwarových balíčků a projektů,
- softwarových balíčků s chybou při překládání jednoho z projektů,
- softwarových balíčků s chybou, která se netýká překládání projektů, ale samotného balíčku,
- chybných překladů projektů,
- softwarových balíčků, které se nevygenerovaly a nestáhly.

Pro `SDK` balíčky je ve výstupním rozhraní navíc informace o použité verzi `MCUXpresso IDE` 4.3.2. Tato přidaná informace je důležitá, protože se při IDE testu 3.3.1 `SDK` balíčků netestují pouze softwarové balíčky, ale také nové verze vývojového prostředí `MCUXpresso IDE` 4.3.2.

Následující seznam použitých softwarových balíčků pro `IDE test`, který obsahuje záznamy s názvem balíčku, konfigurací (podrobněji popsané v sekci 2.3.1), souhrnnou informací o počtu chyb v softwarovém balíčku, verzi použitého generátoru a v případě vyskytnutí chyby i odkaz na manifest soubor softwarového balíčku.

	Package with other errors	Errors
1	Fri_Jan_12_01_28_34_2018-windows-all-LPCXpresso54114_max	ERROR: Unable to find board component for 'lpcxpresso54114_agmp03' board. ERROR: Unable to find board component for 'lpcxpresso54114_agm01' board. ERROR: XDG_RUNTIME_DIR not set in the environment.
2	Fri_Jan_12_01_28_34_2018-windows-all-MKW21D512xxx5_min	ERROR: Unable to find board component for 'usbkw24d512' board. ERROR: XDG_RUNTIME_DIR not set in the environment.
3	Fri_Jan_12_01_28_34_2018-windows-mcuxpresso-LPCXpresso54114_max	ERROR: Unable to find board component for 'lpcxpresso54114_agmp03' board. ERROR: Unable to find board component for 'lpcxpresso54114_agm01' board. ERROR: XDG_RUNTIME_DIR not set in the environment.
4	Fri_Jan_12_01_28_34_2018-windows-mcuxpresso-MKW21D512xxx5_min	ERROR: Unable to find board component for 'usbkw24d512' board. ERROR: XDG_RUNTIME_DIR not set in the environment.

Obrázek 3.5: Formát tabulky pro SDK balíčky s jinou chybou, než je překlad projektu

Pod seznamem použitých balíčků se nachází trojice tabulek. Tabulky byly rozděleny z důvodu přehlednosti výstupního rozhraní a jsou vytvářeny jen v případě, že jsou nezbytné pro zobrazení chybové hlášky.

První tabulka popisuje chyby softwarových balíčků, které nesouvisí s překladem projektů. Na obrázku 3.5 můžete vidět příklad použití.

V druhé tabulce jsou vypsány všechny projekty, které skončily s chybovou hláškou při překladu projektu. Ukázku formátu tabulky s chybovými projekty je možné vidět na obrázku 3.6.

	Project with build error	Package	State
1	Project_MKE02Z64VFM4_frdmke02z40m (Debug)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-KE02Z40M_default	not build
2	Project_MKE02Z64VFM4_frdmke02z40m (Release)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-KE02Z40M_default	not build
3	Project_MKE02Z64VFM4_frdmke02z40m (Debug)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-KE02Z40M_max	not build
4	Project_MKE02Z64VFM4_frdmke02z40m (Release)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-KE02Z40M_max	not build

Obrázek 3.6: Formát tabulky pro SDK balíčky s chybovými projekty

V poslední tabulce jsou vypsány všechny zbývající testované projekty. Tyto projekty se úspěšně přeložily ve vývojovém prostředí bez chyby. Formát tabulky s úspěšně přeloženými projekty je možné vidět na obrázku 3.7.

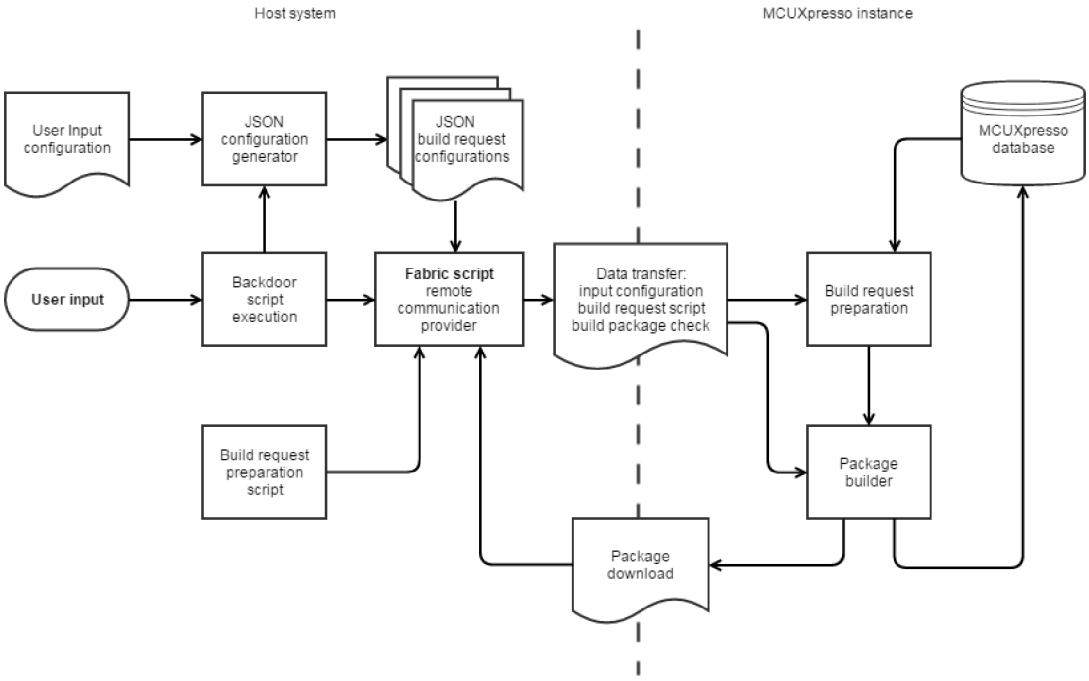
	Project without error	Package	State
1	Project_MK22FN512VDC12_frdmk22f (Debug)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-K22F_default	build success
2	Project_MK22FN512VDC12_frdmk22f (Release)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-K22F_default	build success
3	Project_MK22FN512VDC12_frdmk22f_agmp03 (Debug)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-K22F_max	build success
4	Project_MK22FN512VDC12_frdmk22f_agmp03 (Release)	Fri_Jan_12_01_28_34_2018-windows-all-FRDM-K22F_max	build success

Obrázek 3.7: Formát tabulky pro SDK balíčky s úspěšně přeloženými projekty

3.5 Zadní vrátka

Zadní vrátka, neboli `Backdoor`, konkrétně `SDK builder Backdoor`, je nástroj pro uživatelský přístup ke generování balíčků `MCUXpresso` generátorem bez nutnosti přístupu k webové stránce `MCUXpresso`.

Zadní vrátka slouží k internímu použití pro generování více balíčků s různými kombinacemi konfigurací na jednom nebo více instancích webu `MCUXpresso`. Tato funkce přináší úsporu času při generování balíků pro testovací účely. Uživatel nemusí přistupovat k webu `MCUXpresso` a naklikat si konfigurace na stránkách. Generované balíčky s přístupem `Backdoor` lze snadno stáhnout s tímto skriptem ihned po dokončení generování. Obrázek 3.8 ukazuje hlavní schéma pracovního postupu zadních vrátek.



Obrázek 3.8: Schéma zadních vrátek pro webovou aplikaci MCUXpresso

Hlavní části přístupu zadními vrátky:

Fabric skript

Hlavní částí tohoto nástroje je `Fabric skript`, který umožňuje uživatelům komunikovat po síti se servery instancí `MCUXpresso`. Tento skript nahrazuje funkčnost webového rozhraní `MCUXpresso`. Vstupem je konfigurační soubor s obsahem souvisejícím s konfigurační stránkou `MCUXpresso`. Skript také v průběhu generování umožňuje stáhnout požadovaný balíček z instance `MCUXpresso`. Tento skript může být také použit uživatelem, ale pouze s platným konfiguračním souborem `JSON`. Soubor může být vygenerován konfiguračním generátorem `JSON` podle předdefinované konfigurace uživatelského vstupu.

Generátor JSON konfigurace

Aby bylo možné generovat balíček je potřeba `JSON skript`. K jeho vytvoření slouží generátor `JSON`[3] konfigurací, což je skript, který umožňuje uživateli snadno popsat a generovat konfigurační soubory `JSON`, potřebné pro `Fabric skript`. Hlavním přínosem tohoto skriptu je možnost generovat více konfiguračních souborů `JSON` z jedné uživatelské vstupní konfigurace. Tento skript je součástí spouštěcího skriptu zadních vrátek, který kontroluje a upravuje vstupní konfiguraci do přijatelné podoby. Na obrázku 3.8 se nachází pod názvem `JSON configuration generator`.

Spouštěcí skript zadních vrátek

Spouštěcí skript slouží k vykonávání celého procesu zadních vrátek pro přístupu k instancím `MCUXpresso`. Obsahuje všechny kroky potřebné k provedení přípravy, vytváření a odesílání požadavků na sestavení do instance `MCUXpresso`. K dispozici je také vstupní atribut `-d` pro získání generovaných balíčků na lokální disk z instance `MCUXpresso`. Na obrázku 3.8 se nachází pod názvem `Backdoor execution script`.

Uživatelská vstupní konfigurace

Soubor uživatelské vstupní konfigurace je konfigurací typu `JSON` definující obsah balíčku, který bude rozdělen do jednotlivých konfiguračních souborů, přijatelných pro generátor balíčku v instanci `MCUXpresso`. Obsahuje definici seznamů instancí, desek, procesorů, soustav, volitelných middlewarů, vývojových prostředí a operačních systémů. Na obrázku 3.8 se nachází pod názvem `User input configuration`.

Skript na přípravu požadavku na sestavení

Skript pro přípravu požadavku na sestavení je přesunut do cílové instance, kde se během běhu `Fabric skriptu` používají zadní vrátka. Tento skript poskytuje možnost získat informace z instance databáze k tomu, aby bylo možné vytvořit požadavek na generování balíčku s potřebnými daty. Na obrázku 3.8 se nachází pod názvem `Build request preparation script`.

[3] JSON – JavaScript Object Notation

Kapitola 4

Použité nástroje

Pro testování softwarových balíčků v IDE tesu byla použita vývojová prostředí uVision, MCUXpresso IDE a IAR Workbench. Na vývoj automatizovaných regresních testů byl použit program pro kontinuální integraci Atlassian Bamboo pro automatizaci testovací sady. Implementace testů proběhla ve vývojovém prostředí Jetbrains PyCharm IDE a textovém editoru Sublime.

4.1 Vývojové prostředí

Projekt byl vyvíjen převážně ve vývojovém prostředí PyCharm. Pro účely testování a převážně pro vývoj grafického rozhraní byl využit textový editor Sublime.

Grafické výstupní rozhraní bylo testováno v prostředí email klienta Outlook a webového prohlížeče Chrome. Detailnější popis výstupního rozhraní najdete v kapitole 3.

4.1.1 Jetbrains PyCharm

Implementační část byla vyvíjena v prostředí PyCharm, což je vývojové prostředí pro programovací jazyk Python, který byl vyvinut českou firmou JetBrains [10]. Tento nástroj byl zvolen, protože PyCharm je jediné integrované vývojové prostředí pro jazyk Python, ke kterému škola poskytuje studentskou licenci zdarma [10].

Jednoho z nástrojů PyCharmu, převážně využívaného v tomto projektu, bylo grafické ladění. PyCharm poskytuje možnost ladit procesy tak, že se připojí k danému procesu a přechází do ladícího módu. Je možné vkládat breakpointy, krokovat postup v kódu, sledovat hodnoty proměnných a pohybovat se v rámci zásobníku volání.

Dalšími užitečnými nástroji PyCharmu využitými v této práci jsou analýza kódu, integrovaný jednotkový testovací nástroj a integrace se systémy pro správu verzí.

4.1.2 Sublime

Grafické výstupní rozhraní bylo vyvíjeno v textovém editoru Sublime. Tento nástroj byl zvolen kvůli nenáročnému prostředí a podpoře velkého množství syntaxe různých jazyků a strukturovaných souborů.

Tento textový editor je skvělým nástrojem pro rychlou editaci libovolného textového souboru. Má velice přehledné grafické rozhraní, kterým jsem se nechal inspirovat při navrhování výstupního rozhraní.

4.2 Atlassian nástroje

`Atlassian` je firma, která poskytuje nástroje, které pomáhají se správou projektu a urychlují vývoj software. Hlavním důvodem využití `Atlassian` nástrojů byl systém pro kontinuální integraci `Atlassian Bamboo`, který byl použit pro automatizaci a správu regresních testů.

Všechny `Atlassian` nástroje jsou navzájem propojené. Jako příklad bych uvedl `Jira úkol`, který se odkazuje na revize v `Bitbucketu` a před vytvořením požadavku na sjednocení dvou větví jednoho repozitáře se nad každou revizí spustí automatizované regresní testy na `Bamboo`.

4.2.1 Atlassian Bamboo

Ke kontinuální integraci a nasazení jsem použil nástroj `Bamboo`. Hlavní výhodou oproti konkurenčním nástrojům je nepřetržitá podpora. V případě výpadku nebo vyskytnutí chyby na serveru, firma `Atlassian` začne na opravě pracovat do 10 minut od vyskytnutí. Na `Bamboo` může být spuštěno více sestav paralelně pro rychlejší dokončení. V případě selhání sestavení, poskytuje `Bamboo` analýzu selhání, včetně trasování zásobníku [9].

`Bamboo` poskytuje `REST API`[1], které poskytuje informace o serveru, aktuálním stavu sestavení, atd.

`Bamboo` je rozdělen v hierachickém pořadí do několika částí. První částí jsou testovací sady. Ty tvoří samostatné projekty, které mohou být dále provázány podmínkami pro spuštění. Tyto projekty také obsahují celkovou konfiguraci pro všechny testy z dané sady.

Testovací sady jsou rozděleny na fáze. Důvodem rozdělení na fáze je skutečnost, že občas může mít fáze určitou závislost na předchozí fázi z hlediska určitých artefaktů. Proto také každá fáze musí být dokončena, než přejde na další fázi.

V každé fázi se nachází libovolný počet úloh. Tyto úkoly vykonávají už samotné spuštění testů a klonování repozitářů. Je to nejmenší jednotka práce, kterou agent vykonává.

4.2.2 Atlassian Bitbucket

Pro správu verzí a zálohování zdrojových souborů jsem využil `Bitbucket`. `Bitbucket` jsem použil z důvodu napojení na `Jiru` a `Bamboo` pro zautomatizování regresních testů. Jako verzovací nástroj je zde využit `Git`[2] a repozitář s tímto projektem je uložen na soukromém repozitáři.

4.2.3 Atlassian Jira

Pro řízení projektu, evidenci chyb a problémů při vývoji automatizovaných regresních testů jsem využil nástroj `Jira`. Nástroj mi usnadnil proces vývoje projektu díky neustálému přehledu o stavu a plnění úkolů.

4.3 Testovací prostředí

V této sekci jsou popsána vývojová prostředí, ve kterých se `CMSIS` a `SDK` balíčky testovaly.

Dále základní vlastnosti a schopnosti vývojových prostředí `uVision`, `MCUXpresso IDE` a `IAR Workbench`:

[1] REST API – Representational State Transfer Application Programming Interface

[2] Git – je distribuovaný systém správy verzí, více viz https://git-scm.com/

- Řízení projektů, správa softwarových komponent,
- sestavování projektů, úpravy zdrojových kódů a ladění programů,
- podpora více obrazovek, individuální rozložení oken.

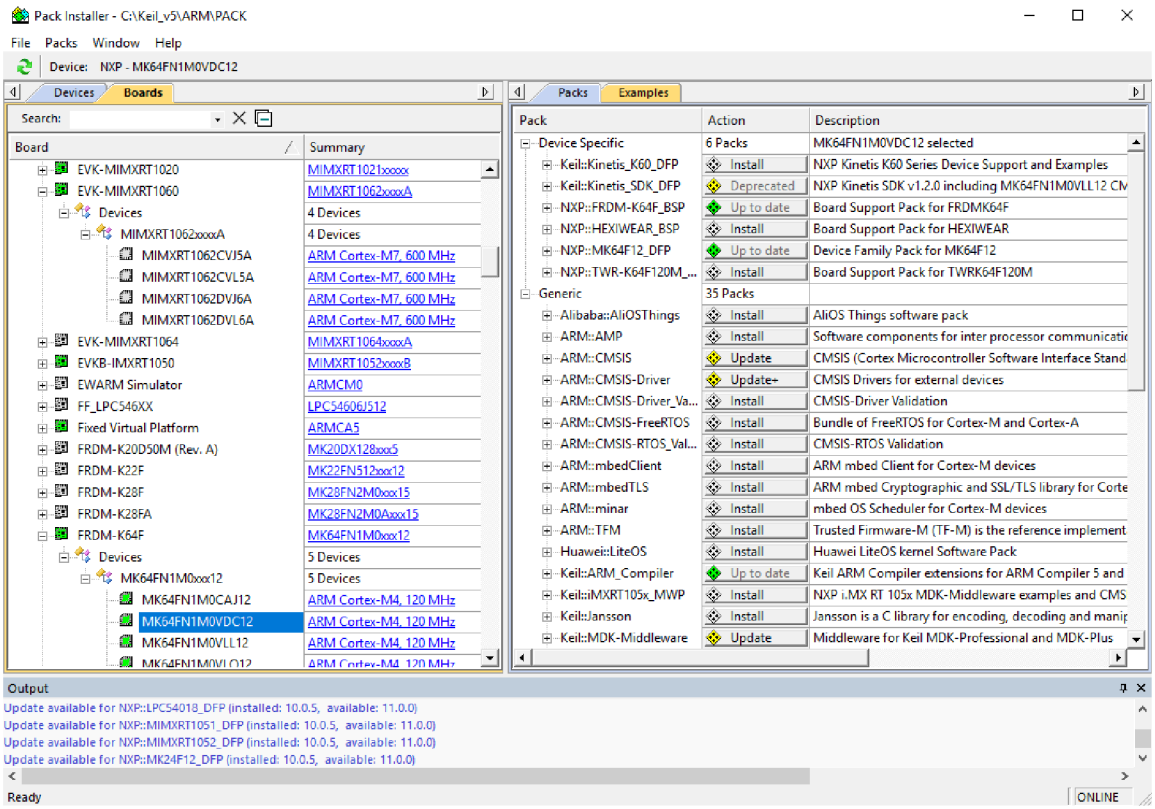
Každé výše zmíněné vývojové prostředí má vlastní správu softwarových komponent. Na obrázku 4.2 můžete vidět správce softwarových komponent. Uživatel si může zvolit jednotlivé komponenty, které jsou kompatibilní s vybraným zařízením. Závislosti softwarových komponent jsou kontrolovány vývojovým prostředím a uživatel je informován validačními zprávami o chybějících závislostech.

Oba typy softwarových balíčků `CMSIS` a `SDK` od firmy `NXP` jsou podporovány ve všech zmíněných vývojových prostředích s vyjímkou `MCUXpresso IDE`, které podporuje pouze `SDK` balíčky.

4.3.1 uVision

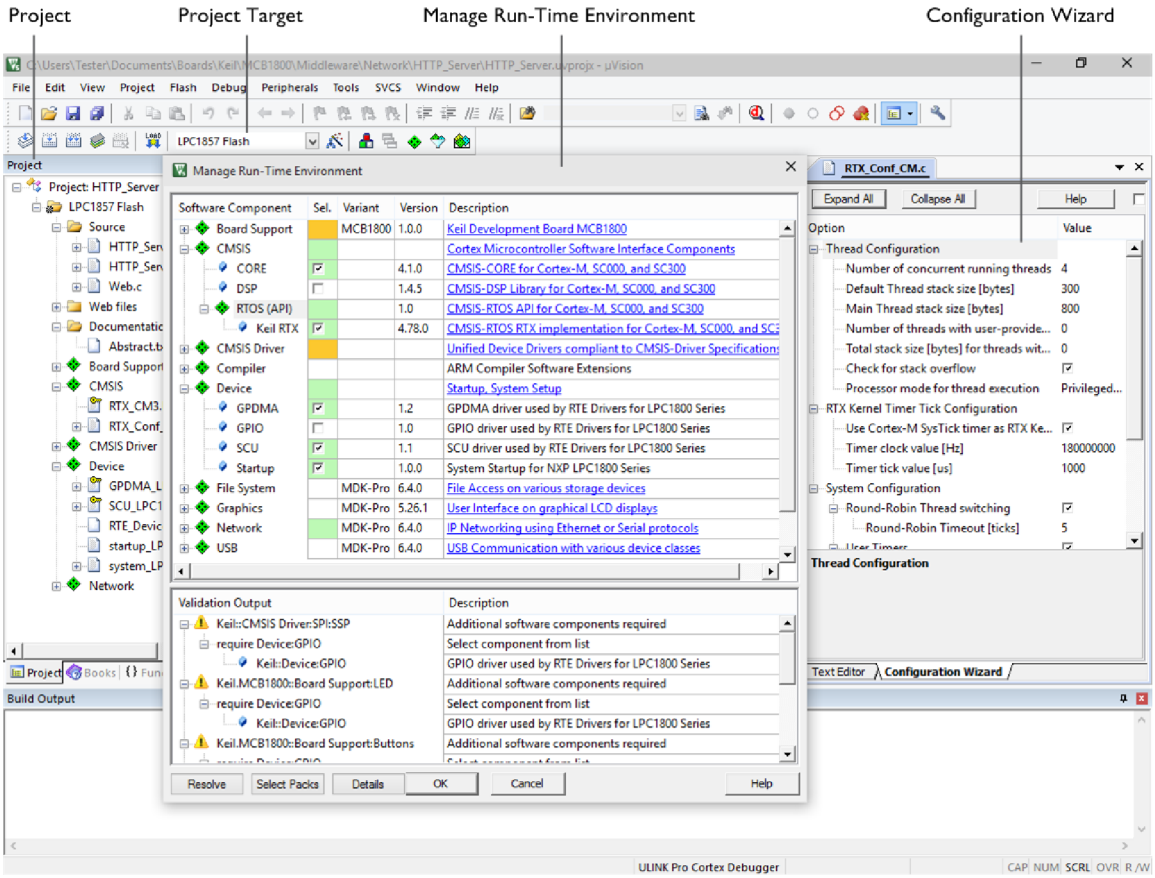
`uVision` je vývojové prostředí vyvíjené původně firmou `Keil`. Od roku 2005 jej dále vyvíjí a vlastní firma `ARM`. Je primárně vytvořen pro tvorbu softwarových aplikací využívající softwarové balíčky `CMSIS`.

Hlavní výhodou tohoto vývojového prostředí je instalátor balíčků na obrázku 4.1, který je propojený se serverem `ARMu` a poskytuje aktuální softwarové `CMSIS` balíčky a v případě vydání nové verze balíčku, nabídne možnost přechodu na novou verzi. [3]



Obrázek 4.1: Instalátor balíčků v uVision

Vývojové prostředí `uVision` je nejlepší volba pro vývoj aplikací pro vestavěné systémy s vyžitím `CMSIS` balíčků. Hlavním důvodem tohoto tvrzení je samotný fakt, že `CMSIS` technologie patří a je dále vyvíjena firmou `ARM` a oproti ostatním vývojovým prostředím podporujícím `CMSIS` balíčky nedochází k chybám souvisejícím s nekompletní implementací této technologie.



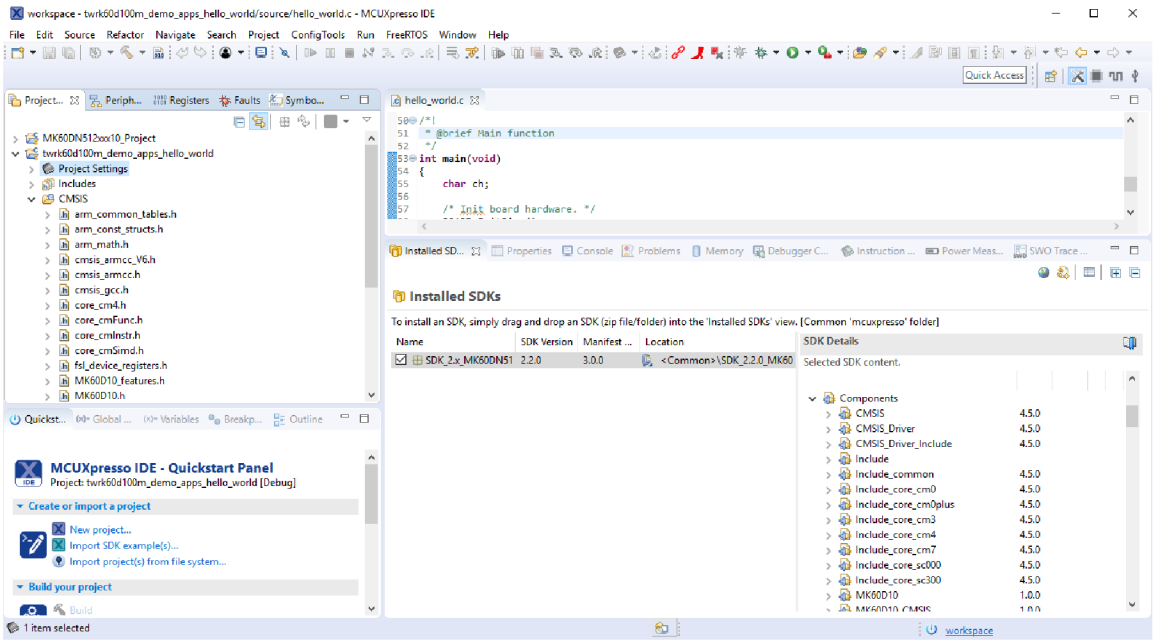
Obrázek 4.2: Správa projektu a softwarových komponent v uVision

4.3.2 MCUXpresso IDE

Vývojové prostředí `MCUXpresso IDE` je nejlepší volbou pro vývoj aplikací pro vestavěné systémy s vyžitím `SDK` balíčků. Hlavním důvodem tohoto tvrzení je samotný fakt, že koncept `SDK` balíčků a samotné vývojové prostředí `MCUXpresso IDE` je vyvíjené firmou `NXP`.

`MCUXpresso IDE` je založené na vývojovém prostředí `Eclipse`, které lze vidět na obrázku 4.3. Toto řešení nese sebou různé nevýhody a omezení. Jediná výhoda tohoto řešení z vývojového hlediska je již předpřipravené vývojové prostředí pro možnou editaci [20].

`MCUXpresso IDE` jako jediné, z dostupných vývojových prostředí je multiplatformní. Jsou podporovány operační systémy `Windows`, `Linux` a `Mac OS X`.

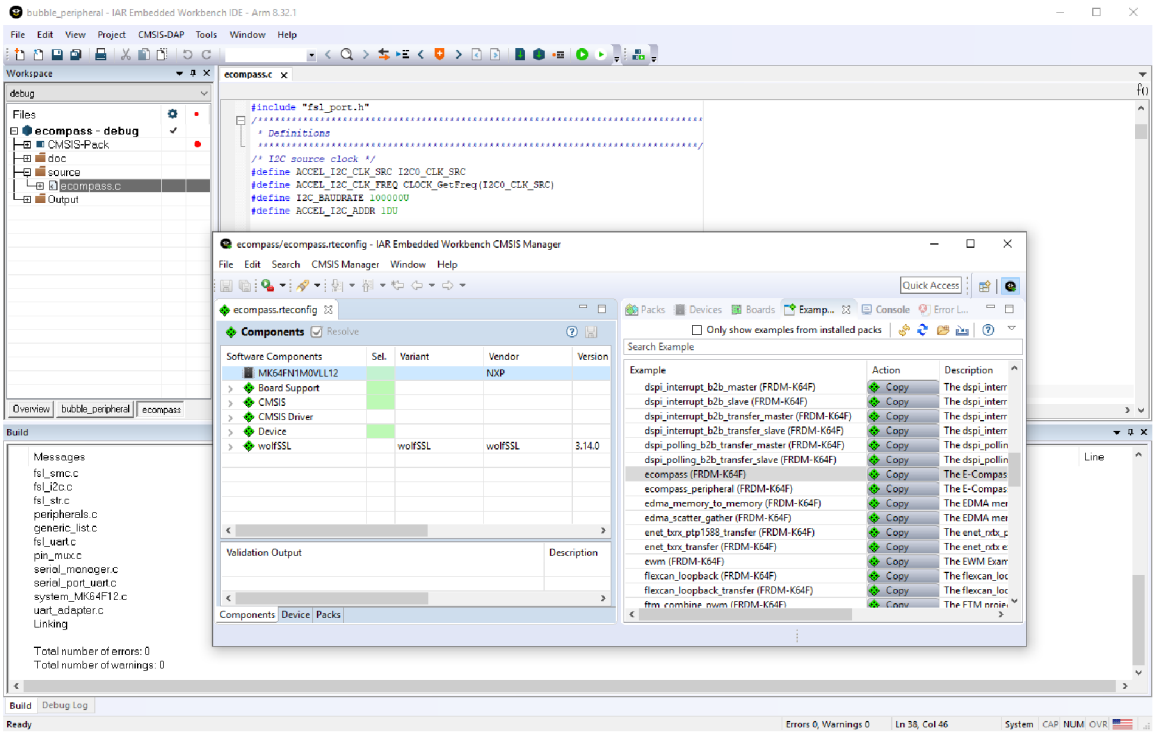


Obrázek 4.3: Správa projektu a softwarových komponent v MCUXpresso IDE

4.3.3 IAR Workbench

`IAR Workbench` je vývojové prostředí s podporou `CMSIS` balíčků. Firma `IAR` vyvíjí kompilátory, ladící nástroje a firmwary, které dává k dispozici právě v prostředí `IAR Workbench` viz. 4.4 obrázek. Podporuje softwarové balíčky založené na `CMSIS-pack` technologii [23].

Firma `IAR` nevyvíjí `CMSIS` balíčky a ani `CMSIS-pack` technologii a podpora pro tyto balíčky byla doplněna do vývojového prostředí nedávno. Důsledkem toho dochází k chybám souvisejícím s nekompletní implementací této technologie a nedostatečnou vyzrálostí `CMSIS pack manager` pluginu pro `Eclipse`, který vyvíjí `ARM`, a který je používán `IAR`.



Obrázek 4.4: Správa projektu a softwarových komponent v IAR Workbench

Kapitola 5

Implementace regresních testů

Tato kapitola pojednává zejména o implementaci jednotlivých částí regresních testů znázorněných na obrázku 3.1. Jelikož byl pro automatizaci regresních testů využit nástroj pro kontinuální integraci a nasazení `Atlassian Bamboo` 4.2.1, tak bude nezbytné si popsat proces fungování automatizace.

Pro realizaci automatizovaných regresních testů, je nutné mít vhodný návrh struktury 3.3 a strukturu pro reprezentaci dat 3.4.1. Struktura regresních testů musí umožňovat automatizaci testů, proto je nutné testy s tímto vědomím vytvářet.

Obecně se snažíme testy pokrýt co největší část testované aplikace, ale u automatizovaných testů se pokrývají především části, u kterých je reálná hrozba výskytu závažných chyb. V této práci byly testovány softwarové balíky ve vývojových prostředích a porovnány starší verze softwarových balíčků s novými.

Dá se říci, že v případě implementace automatizovaných testů se většinou volí vysokoúrovňový interpretovaný nebo skriptovací jazyk. V našem případě byl k implementaci použit jazyk `Python`, `Shell` a pro vizualizaci dat `HTML`.

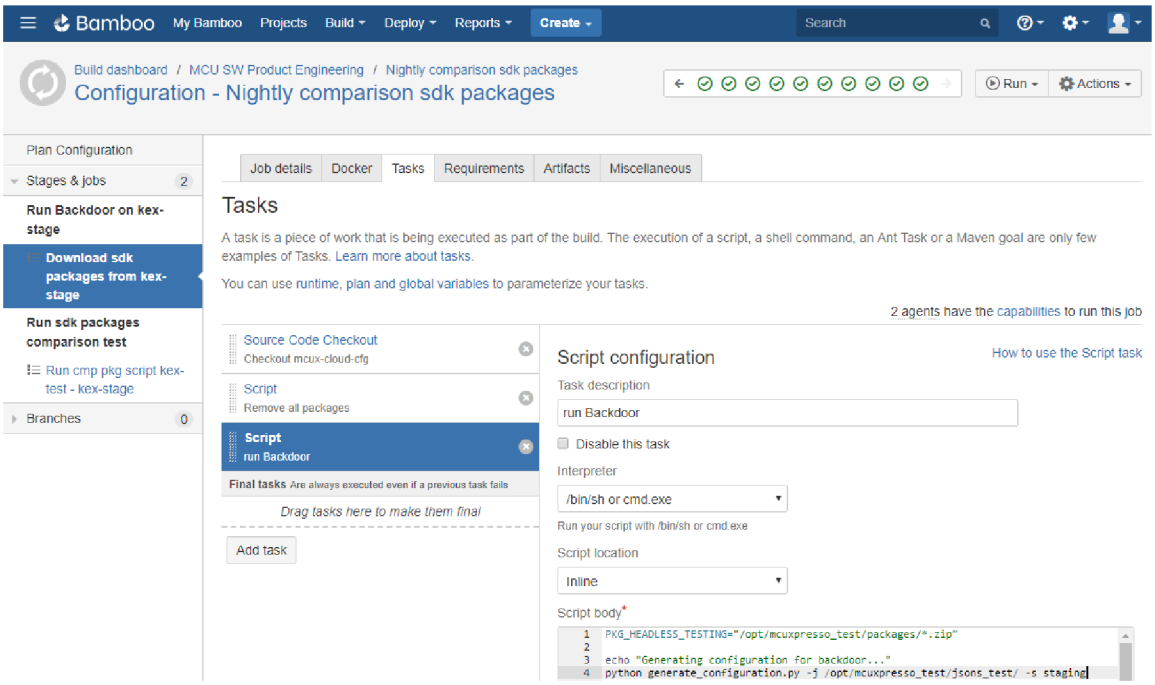
5.1 Automatizace

Automatizace regresních testů byla implementována pomocí nástroje pro kontinuální integraci a nasazení `Atlassian Bamboo`, podrobněji popsaného v sekci 4.2.1. Byly vytvořeny čtyři projekty *CMSIS packs comparison test*, *Nightly comparison sdk packages*, *Mcuxpresso ide headless testing* a *CMSIS packs IDE test*. Dva projekty pro CMSIS balíčky a dva pro SDK balíčky.

Projekty jsou nastaveny pomocí *Trigger configuration*, aby `Bamboo` jednotlivé plány spustil automaticky. Je potřeba nastavit spouštěče, tak, aby určily, jak a kdy bude sestavení spuštěno. Plán spouštění se nastavuje podle navrženého schématu na obrázku 3.1. Plány je možné také spustit kdykoliv ručně [6].

Dalším nastavením jsou `Bamboo agenti`, kterými jsou nastaveny servery, na kterých se budou testy spouštět. Ve všech plánech je nastavena instance `MCUXpresso` serveru `Kex-test` [5].

Jednotlivým plánům jsou nastaveny a přiřazeny repozitáře. Lze přidat jeden nebo více repozitářů, které budou k dispozici pro každou práci v plánu. Repozitáře se nachází na serveru `Bitbucketu` 4.2.2, odkud jsou namapovány na jednotlivé úkoly v plánu. Na obrázku 5.1 je možné vidět příklad jednoho z plánů.



Obrázek 5.1: Noční porovnávací test SDK balíčků

Při vytváření úkolů pro Bamboo plány byl využit interpret příkazů `Shell`. Celá konfigurace Bamboo plánů poté může běžet a spouštět se bez lidského zásahu.

5.2 Proces stahování softwarových balíků

V této sekci je popsán proces stahování softwarových balíků pomocí nástroje `Backdoor`. K implementaci byl použit programovací jazyk `Python` a knihovna `fabric`. Další obecné informace jsou popsány v sekci 3.5.

Uživatel má k dispozici dva způsoby zadání vstupní konfigurace pro softwarové balíčky, kde uživatel specifikuje konfiguraci balíčků a serverů, ze kterých má balíček získat. Prvním je konfigurační soubor s názvem `input_config.py`, kde je pořeba specifikovat následující proměnné:

- **USER** = ””
  - Uživatel, který se chystá sestavit balíček, může tuto proměnnou vyplnit svou e-mailovou adresou nebo přihlašovacím jménem, kterou nastavil na webové stránce `MCUXpresso`.
  - Nastavení uživatelské proměnné umožňuje systému přidávat zabudované balíčky do softwarových úložišť zadaného uživatele v uživatelském rozhraní.
- **SYSTEMS** = [””] - Systémy, kde bude spuštěno generování balíčků.
  - Je navržen jako seznam řetězců oddělených čárkou, všechny konfigurace budou spuštěny na všech uvedených serverech.
- **BOARDS** = [””] - Desky, pro které budou vygenerovány softwarové balíčky.

  - Je navržen jako seznam řetězců oddělených čárkou, takže počet desek v tomto seznamu bude generovat stejný počet balíčků v každém nastavení podle komponent, vývojových nástrojů a operačních systémů.
  - Je možné automaticky vybrat všechny desky z určitého vydaného setu, automaticky vybrat náhodnou desku z posledního vydaného setu nebo je možné automaticky vybrat všechny dostupné desky.

- **DEVICES** = [""] - Zařízení, pro které budou vygenerovány softwarové balíčky.
  - Zařízení budou vygenerována nezávisle na seznamu desek `BOARDS`. Tyto seznamy se nespojují dohromady.
  - Je také navržen jako seznam řetězců oddělených čárkou, takže počet zařízení v tomto seznamu vygeneruje stejný počet balíčků, pokud v seznamu `BOARDS` není vybrána žádná deska.
  - `Backdoor skript` automaticky vybere odpovídající nebo podobné desky pro zařízení `DEVICES`.
  - Je možné automaticky vybrat všechna zařízení z určitého vydaného setu, automaticky vybrat náhodné zařízení z posledního vydaného setu nebo je možné automaticky vybrat všechna dostupná zařízení.

- **KITS** = [""] - Sestavy, pro které budou vygenerovány softwarové balíčky.
  - Je navržen jako seznam řetězců oddělených čárkou, takže počet sad v tomto seznamu generuje stejný počet balíčků v každém nastavení podle komponent, vývojových nástrojů a operačních systémů.
  - Je možné automaticky vybrat všechny sestavy z určitého vydaného setu, automaticky vybrat náhodnou sestavu z posledního vydaného setu, nebo je možné automaticky vybrat všechny dostupné sestavy.

- **TAGS** = [""] - seznam značek `SDK`, které budou použity pro generování požadovaných setů (konfigurací `SDK`).
  - Pro každou značku ze seznamu `TAGS` vygeneruje systém balíčky pro všechny položky v seznamech hardwarových platforem: `KITS`, `BOARDS` a `DEVICES`.

- **COMPONENTS** = [""] - Komponenty, které budou přidány do generovaného balíčku.
  - Je navržen jako seznam seznamů, kde každý vnitřní seznam bude kombinován s jinými konfiguracemi.
  - Počet položek ve vnějším seznamu znamená počet konfigurací a počet položek ve vnitřním seznamu znamená počet komponent, které mají být přidány v jedné konfiguraci.
  - Jsou zde zavedeny již předvolené sady komponent, které lze nastavit pomocí názvu `min` pro minimální, `max` pro maximální a `default` pro výchozí sadu volitelných komponent

- **TOOLCHAINS** = [""] - Vývojové nástroje, pro které budou vygenerovány softwarové balíčky.

  - Je navržen jako seznam řetězců `toolchain`[1], kde každá položka v seznamu bude kombinována s jinými konfiguracemi.

- **OS** = [""] - Operační systémy, pro které bude vytvořen balíček.
  - Také je navržen jako seznam řetězců operačního systému, které budou jednotlivé položky v seznamu kombinovány s jinými konfiguracemi.

- **BUILD_NEW** = [""] - Volitelný parametr. Výchozí hodnota: `False`.
  - Pokud je nastavena na hodnotu `True`, bude vždy generován nový balíček `SDK`.
  - Pokud je nastaven na hodnotu `False` a SDK balíček je již vygenerován z dřívějšího požadavku, tak nebude znovu přegenerován, ale bude stáhnut již existující SDK balíček

Druhý konfigurační soubor `generate_configuration.py` pak nabízí možnost si zvolit konfiguraci pomocí parametrů nebo předat `JSON`[2] soubory vygenerované z předchozích spuštění `Backdoor` skriptu pomocí `makejsons.py`. Výstupem pro vás bude vytvořený soubor `input_configurations.py` s požadovanou konfigurací.

Ke komunikaci mezi `Backdoor` skriptem a serverem, odkud se softwarové balíčky stahují, byla využita vysokoúrovňová `Python` knihovna `fabric` [11]. Slouží pro vzdálené spouštění příkazů `shell` přes `SSH`[3]. Zde byly využity základní funkce `run()`, `cd()`, `get()`, `put()` a nastavení hostů `env`.

Kvůli občasným potížím s konektivitou k internetu a výpadky serverů byly zavedeny časové limity, do kterých se musí každý stahovaný balíček vejít. Limity byly nastaveny vhodným způsobem na základě velikosti a délky generování `MCUXpresso Builder` generátorem. K tomuto účelu byla využita knihovna `time`[16].

5.3 Testování balíčků ve vývojových prostředích

K implementaci `IDE testu`, podrobněji popsaného v sekci 3.3.1 byl použit programovací jazyk `Python`. V rámci mého řešení je test rozdělen na dva skripty.

První skript nese název `cmsis_ide_testing.py` a je určen pro testování CMSIS balíčků. Na vstupu jsou skriptu předány informace o umístění softwarových balíčků, vývojových prostředích, ve kterých se mají balíčky testovat a dále informace o tom, zda mají být předem instalovány `DFP`[4] balíčky do vývojových prostředí pomocí funkce `install_dfp()`.

Hlavní proces testu je funkce `build_examples()`. Zde je využita `Python` knihovna `multiprocessing` [13], pomocí které se předpřipravují testovací procesy ke spuštění funkcí `Process()`. Každý proces se skládá z příkazu pro příkazovou řádku pro spuštění překládání projektu ve vývojovém prostředí. Pro přístup a ovládání vývojového prostředí zde byla využita příkazová řádka, která byla dostupná pro vývojové prostředí `IAR Workbench` [22] i `uVision` [18].

Po nastavení procesů se provede jejich spuštění funkcí `run_parallel_process()` na všech dostupných `theadrech`. Výsledky testů jsou uloženy do souborů a souhrnné informace do `Python` slovníku s formátem:

[1] Toolchain – nástrojová sada
[2] JSON – JavaScript Object Notation
[3] SSH – Secure Shell, zabezpečený komunikační protokol
[4] DFP – Device Family Pack, viz. sekce 2.3.2

```
errors[build|install][pack_name][toolchain][failed|success][project_name]
```

Využití knihovny `multiprocessing` se značně optimalizoval proces testování softwarových balíčků ve vývojovém prostředí. Níže je možné vidět použité řešení:

```
import os
from multiprocessing import Process

numberOfThreads = int(os.environ['NUMBER_OF_PROCESSORS'])

def chunks(l, n):
    for i in range(0, len(l), n):
        yield l[i:i + n]

# @param [Array<Process>] jobs: Array of Process
def run_parallel_process(jobs, pack_name):
    for i in chunks(jobs, numberOfThreads):
        for j in i:
            j.start()
        for j in i:
            j.join()
    working_folder = os.getcwd()
    files = os.listdir(working_folder)
    for file_name in files:
        if file_name.startswith('iar_dict_errors'):
            with open(file_name, 'r') as f:
                new_error = eval(f.read())
                errors['build'][pack_name]['IAR']['failed']\
                    .extend(new_error['build'][pack_name]['IAR']['failed'])
                errors['build'][pack_name]['IAR']['success']\
                    .extend(new_error['build'][pack_name]['IAR']['success'])
            os.remove(file_name)
```

Poslední částí skriptu je reprezentace testovaných dat vytvořením výstupního rozhraní ve funkci `create_email()`. Popis této části je možné přečíst v sekci 3.4.2. Pro odeslání výsledků zde bylo využito `Python` knihovny `MIMEText` [12] ve funkci `send_email()`.

Druhý skript s názvem `mcuxpresso_test.py` je určen pro testování SDK balíčků ve vývojovém prostředí `MCUXpresso IDE`. Funkcionalita tohoto skriptu je, až na pár výjimek, obdobná s prvním skriptem `cmsis_ide_testing.py`.

Prvním rozdílem je, že na vstupu je navíc předáváno umístění instalačního souboru aktuální vývojářské verze `MCUXpresso IDE`. Tento soubor je použit k instalaci novější verze vývojového prostředí `MCUXpresso IDE` s pomocí skriptu `mcux_ide_install`.

Druhým rozdílem je spouštění `MCUXpresso IDE` pomocí `Python` knihovny `subprocess` [14] namísto `multiprocessing` [13]. Důvodem je, že testovaní na `MCUXpresso IDE` probíhá v Unixovém prostředí na operačním systému `Ubuntu` a `IAR Workbench` i `uVision` jsou testovány na operačním systému `Windows`. Blíže popsáno v sekci 4.3.

5.4 Porovnávání souborových struktur

Implementace porovnávacího testu byla jedna z náročnějších částí implementace. Byl zde využit programovací jazyk `Python` a interpret příkazové řádky `Bash`. Řešení implementace je rozděleno do tří částí, přičemž se každá část nachází v jednom souboru.

5.4.1 Proces zpracování a řízení

První část slouží k řízení celého porovnávání softwarového balíčku a je uložena v souboru s názvem `zip_compare.sh`. Soubor slouží jako spouštěcí skript pro porovnání dvou softwarových balíčků. Na vstupu jsou předány informace o umístění softwarových balíčků pomocí parametrů `-zip` a `-zip2`. Dále je třeba specifikovat o jaký typ softwarového balíčku se jedná v parametru `-v`.

Další parametry jsou volitelné a předává se jimi informace:

- `-a <názvy atributů>`: atributy, které mají být u strukturovaných souborů typu `XML` ignorovány.
- `-i <názvy souborů nebo přípon>`: názvy souborů nebo přípon, které nemají být porovnávány.
- `-w`: parametr, kterým se nastavuje, zda se mají ignorovat bílé znaky při porovnávání souborů.

Ve skriptu `zip_compare.sh` nejsou použity funkce, ale posloupnosti složených příkazů. Jako první se nad softwarovými balíčky spustí nástroj `zipcmp` [7], který porovná jména a nekomprimovanou velikost souborů. Pokud `zipcmp` najde jakékoliv rozdíly mezi balíčky, bude porovnávání pokračovat. Nástroj `zipcmp` zde byl použit kvůli optimalizaci celého procesu porovnávání. Umožňuje porovnat celou souborovou strukturu archivu, aniž by ho musel rozbalovat. V případě nenalezení rozdílu je skript ukončen a nemusí rozbalovat a porovnávat každé soubory zvlášť.

Ve zbylé části skriptu `zip_compare.sh` se po rozbalení softwarových balíčků začínají porovnávat jednotlivé soubory, které nástroj `zipcmp` vyhodnotil jako rozdílné.

Textové soubory jsou porovnávány nástrojem `diff`. Zde typicky bývají vždy rozdílné soubory, ovšem v případě použití parametru `-w` může nastat situace, kdy dané soubory budou vyhodnoceny jako identické. V případě nalezení rozdílů nástrojem `diff` se jeho výstup předá funkci `diff_output_to_html()` ze souboru `calculations.py`. Podrobněji popsáno v sekci 5.4.2.

`XML` soubory jsou porovnávány skriptem `xmldiffs.py`, podrobněji popsaného v sekci 5.4.3. `XML` soubor je zde myšlen jako strukturovaný soubor typu `XML` a v softwarových balíčcích se může vyskytovat s těmito příponami: `xml`, `pdsc`, `ewp`, `eww`, `ewd`, `uvmpw` ,`uvoptx`, `uvprojx`. Na základě vyhodnocení skriptem `xmldiffs.py` je výstup, v případě nalezení rozdílů, předán funkci `diff_output_to_html()` ze souboru `calculations.py`. Podrobněji popsáno v sekci 5.4.2.

Výsledky porovnávacího testu jsou předány v souborech na zpracování do výstupního rozhraní. Podrobněji popsáno v sekci 5.5.

5.4.2 Optimalizační metody

Druhá část porovnávacího testu se nachází v souboru `calculations.py`, kde se nachází optimalizační skripty. Při testování regresních testů byly nalezeny optimalizační nedostatky i přes propracovaný návrh. Jednotlivé části kódu implementace, které vykazovaly nadměrnou zátěž na výpočetní výkon, byly přesunuty do souboru `calculations.py` a přeparsovány do jazyka `Python`.

Nachází se zde funkce `diff_output_to_html()`, která převádí výstupní formát nástroje `diff` do jazyka `HTML`. Jako inspirace vzhledu pro tuto konverzi zde byl textový editor Sublime 4.1.2.

Druhá funkce `remove_ignore_files()` odstraňuje názvy souborů, které nemají být porovnávány, z výstupu nástroje `zipcmp`.

5.4.3 XML porovnávač

Porovnávání dvou strukturovaných souborů typu `XML` provádí skript s názvem `xmldiffs.py`. Implementace je inspirována algoritmem `X-Tree Diff+`, který je popsán v sekci 2.4.2. V implementovaném algoritmu byla použita stejná logika jako v algoritmu `X-Tree Diff+`, ale navíc byla rozšířena o kontrolu duplikátních `X-tree` podstromů. Ukázka jednoho z příkladů je vidět v sekci 2.4.2 na obrázku 2.2.

Vstupem programu jsou dva strukturované soubory typu `XML`. Dalšími volitelnými parametry jsou `-a`, kterým se definují atributy, které mají být při porovnávání ignorovány a `-c`, který sdělí, zda se jedná o projektový soubor pro vývojové prostředí `IAR Workbench`.

Část projektových souborů pro vývojové prostředí `IAR Workbench` je přeformátována, jelikož obsahuje nevalidní `XML` znaky, které jsou specifické pouze pro `IAR Workbench`. Ukázku nevalidní `XML` sekce tohoto souboru je možné vidět na obrázku 5.2.

```
1943    </group>
1944    <group>
1945      <name>doc</name>
1946      <file>
1947        <name>$PROJ_DIR$/../readme.txt</name>
1948      </file>
1949    </group>
1950    <cmsisPackSettings>
1951      <rte>&lt;?xml version="1.0" encoding="UTF-8" standalone="no"?&gt;
1952  &lt;configuration xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"&gt;
1953    &lt;components&gt;
1954      &lt;component Cvariant="frdmk64f" Cversion="1.0.0" Csub="project_template" Cgroup="SDK Project Template" Cclass="Board Support"&gt;
1955        &lt;package url="http://mcuxpresso.nxp.com/cmsis_pack/repo/" version="11.0.0" vendor="NXP" name="FRDM-K64F_BSP"/&gt;
1956        &lt;file category="sourceC" version="1.0.0" attr="config" name="project_template/pin_mux.c"/&gt;
1957        &lt;file category="header" version="1.0.0" attr="config" name="project_template/pin_mux.h"/&gt;
1958        &lt;file category="sourceC" version="1.0.0" attr="config" name="project_template/board.c"/&gt;
1959        &lt;file category="header" version="1.0.0" attr="config" name="project_template/board.h"/&gt;
1960        &lt;file category="sourceC" version="1.0.0" attr="config" name="project_template/clock_config.c"/&gt;
1961        &lt;file category="header" version="1.0.0" attr="config" name="project_template/clock_config.h"/&gt;
1962      &lt;/component&gt;
1963      &lt;component Cversion="2.3.1" Csub="gpio" Cgroup="SDK Drivers" Cclass="Device"&gt;
1964        &lt;package url="http://mcuxpresso.nxp.com/cmsis_pack/repo/" version="11.0.0" vendor="NXP" name="MK64F12_DFP"/&gt;
```

Obrázek 5.2: Ukázka nevalidní XML sekce v projektovém souboru pro IAR Workbench

Hlavní proces porovnávání se nachází ve funkci `xmldiffs()`. Pro parsování `XML` souborů byla použita `Python` knihovna `xml.etree.ElementTree` [15]. Tato knihovna velmi usnadnila práci s `XML` soubory. Voláním knihovní funkce `ElementTree.parse()` byl převeden `XML` soubor do `X-tree` a tím proveden i `krok 0` algoritmu `X-Tree Diff+`.

S využitím dočasných souborů pomocí knihoven `NamedTemporaryFile` a `codecs`, jsou oba `XML` soubory seřazeny podle názvu elementů a atributů ve funkci `write_sorted()`.

Funkce `write_sorted()` operuje s `X-Tree`. Pokud daný uzel `X-Tree` je dítě, potom každému elementu jsou abecedně seřazeny atributy funkcí `node_key()` a nejbližší podelementy voláním funkce `sort(key=node_key)`. Následně se dítě stává rodičem a pro každé dítě tohoto uzlu je rekurzivně volána funkce `write_sorted()`. Pokud daný uzel `X-Tree` je rodič, potom jsou všechny jeho atributy seřazeny abecedně funkcí `node_key()` a následně se uzel uzavře. Výsledkem funkce jsou dva seřazené `X-Tree XML` stromy uložené ve dvou dočasných souborech.

Seřazené `XML` soubory jsou za pomocí `Python` knihovny `subprocess` [14] porovnány nástrojem `diff` s parametry `-w` ignorující bílé znaky a `-u` pro specifický typ zobrazení rozdílů, který je využit v další části algoritmu.

Výsledek porovnání `XML` souborů je znovu prověřen druhým průchodem nad výsledkem první části. Díky přesnému formátu výstupu nástroje `diff` jsou pomocí `Python` listů načteny

výsledky porovnání a znovu porovnány mezi jednotlivými částmi kódu, oddělenými znaky '@@'.

5.5 Zpracování dat

Ke zpracování dat byl využit interpret příkazové řádky Bash a pro vizualizaci značkovací jazyk HTML, kaskádové styly CSS a programovací jazyk JavaScript. Zpracování dat je rozdělené do dvou souborů. Pro SDK balíčky se implementace nachází v souboru report.sh a pro CMSIS balíčky v souboru report_cmsis.sh. Oba skripty mají stejnou funkčnost, ale kvůli rozdílnému složení HTML elementů a obsahu výstupního rozhraní byli rozděleny do dvou částí.

Vzhled výstupního rozhraní je popsán pomocí kaskádových stylů CSS a je zapsán do těla HTML pomocí elementu style. Ve výstupním rozhraní porovnávacího testu konkrétního softwarového balíčku uloženého na síťovém disku je navíc využit programovací jazyk JavaScript k přepínání mezi rozdílnými a chybějícími soubory. Pro výstupní rozhraní odesílané emailem bylo možné využít pouze značky HTML3 a to kvůli omezené podpoře HTML jazyka v programu Outlook. Outlook je primární aplikací, používanou uživateli regresních testů softwarových balíčků, a proto byl použit i pro testování skriptů.

Oba skripty byly vytvořeny jako jednoúčelové skripty pro nasazení do automatizovaných regresních testů softwarových balíčků. Jako vstup dostanou skripty data v podobě souboru Different a souboru <jméno_balíčku>_<cesta_k_souboru>.error pro každý rozdílný soubor. Všechny rozdílné soubory jsou pak popsány v souboru Different.

Skript prochází postupně souborem Different a pro každý balíček a soubor a vytváří si statistiky, přiděluje informace, které vyhledává pomocí regulárních výrazů. Všechny získané informace jsou postupně zapisovány do HTML šablony, která je provázána v celém procesu zpracování.

Výsledkem skriptů report.sh a report_cmsis.sh je odeslání emailu s výsledky testů pomocí nástroje mail a přesunutí všech dat o porovnávaných softwarových balíčcích na síťový disk. Pro přesun souborů je použita základní funkce mv, protože na serveru, na kterém zpracování dat vždy probíhá, je daný síťový disk namapován.

Kapitola 6

Závěr

Cílem mé bakalářské práce bylo vytvořit automatické regresní testy pro softwarové balíčky SDK a CMSIS podle požadavků zadavatele - firmy NXP Semiconductors. Řešení tedy zahrnovalo fáze od stanovení požadavků, analýzy a seznámení se s dostupnými metodami a nástroji přes návrh až po implementaci. K tomu bylo nezbytné nastudovat teoretické znalosti technologií, zejména pak způsoby porovnávání XML souborů a nástroj pro kontinuální integraci a nasazení Atlassian Bamboo.

6.1 Zhodnocení

Tato práce zahrnuje nastudování principů automatizovaného testování, dostupných metod, nástrojů a analýza požadavků firmy NXP. Dále bylo třeba nastudovat technologie, které budou využity. Jedná se o systém pro kontinuální integraci a nasazení Atlassian Bamboo a vývojové prostředí uVision, MCUXpresso IDE a IAR Workbench.

Stěžejním bodem analýzy bylo také nastudování problematiky diferencování XML souborů. K dosažení tohoto cíle bylo třeba provést rešerši dostupných metod a nástrojů.

Po získání nezbytných znalostí a provedení analýzy požadavků bylo možné přistoupit k samotnému návrhu. Výsledkem této fáze bylo tedy vytvoření schématu struktury regresních testů. Grafické znázornění fungování regresních testů přispělo k pochopení celého systému jako celku a k vytvoření optimálního řešení.

V rámci návrhu bylo nezbytné vymyslet způsob získávání softwarových balíčků, který vedl k vytvoření zadních vrátek pro webovou aplikaci MCUXpresso Builder. K tomuto účelu bylo využito nástroje Fabric, který umožnil komunikaci po síti se servery instancí MCUXpresso. Návrh zadních vrátek bylo konzultováno a navrhnuto se zaměstnanci firmy NXP, čímž byla zajištěna validita. Dalším stěžejním bodem při návrhu byla reprezentace dat ve výstupním rozhraní. Pro popis stylu rozhraní zde byl využit značkovací jazyk HTML a pro vizualizaci dat byly využity tabulky.

K implementaci testů softwarových balíčků ve vývojových prostředích, porovnávání XML souborů, zadních vrátek a několika optimalizovaných metod byl využit programovací jazyk Python. Ostatní implementace byla provedena v jazyku Shell s využitím regulárních výrazů pro zajištění nejoptimálnějšího chodu regresních testů. Služba Bitbucket byla využita pro správu verzí kódu, program PyCharm a Sublime sloužil jako vývojové prostředí a systém Atlassian Bamboo pro automatizaci regresních testů.

Kromě nástrojů bylo třeba se seznámit a vybrat vhodný algoritmus pro porovnávání XML souborů. Kvůli složitosti problematiky s porovnáváním XML souborů byl pro implementaci

vybrán už existující algoritmus. Implementace byla inspirována na základě tohoto algoritmu a dále byl doplněn o nedostatky týkající se porovnávání komplexních a rozsáhlých `XML` souborů.

Výsledkem práce jsou tedy automatizované regresní testy pro testování projektů softwarových balíčků `SDK` a `CMSIS` ve vývojových prostředích a pro porovnání mezi novými a předešlými verzemi softwarových balíčků `SDK` a `CMSIS`. Dále pak reprezentace dat z výsledků regresních testů.

6.2 Možnosti dalšího rozšíření

V rámci dalších rozšíření a vylepšení této práce je možné uvažovat o rozšíření `XML` porovnávání o možnosti dalších modifikací při porovnávání změn jako je ignorování podle názvu elementů a hodnot atributů a kombinace mezi nimi. V aktuální implementaci je zatím zakomponován pouze parametr pro ignorování elementu na základě jména atributu.

Dalším užitečným rozšířením, které by mohlo značně optimalizovat a zkrátit dobu testování, je zavedení vstupních parametrů regresních testů pro ignorování konkrétních projektů softwarových balíčků. Současně se testují všechny ukázkové projekty, které se nacházejí v softwarových balíčcích. Někdy je ovšem potřeba znát výsledky pouze konkrétních projektů, nebo naopak vyřadit z porovnávání známé chyby v projektech a zpřehlednit tak výstupní vizualizaci dat.

Další rozšíření, které by mohlo vést ke zvýšení uživatelského komfortu, pak představuje v porovnávacím testu způsob reprezentace výsledných dat, a to podle nalezených chyb. Pokud by se ve více souborech nacházela stejná změna, tak by se mohlo vyhledávat ve změnách v souborech podle konkrétních změn. Do výstupního rozhraní, viz. obrázek A.2, by se přidala další tabulka se změnami. Každá změna by obsahovala seznam souborů, ve kterých se objevuje a v jaké části souboru.

Samotný návrh regresních testů umožňuje do budoucna snadné zavedení nových vývojových prostředí pro testování softwarových balíčků nejen `CMSIS` a `SDK` od firmy `NXP`.

Literatura

[1] ARM: *CMSIS-Pack Documentation*. [Online; navštíveno 25.01.2019].
URL http://arm-software.github.io/CMSIS_5/Pack/html/index.html

[2] ARM: *Cortex Microcontroller Software Interface Standard*. [Online; navštíveno 25.01.2019].
URL https://developer.arm.com/tools-and-software/embedded/cmsis

[3] ARM: *Learning platform for Cortex-M microcontroller users*. [Online; navštíveno 25.01.2019].
URL http://www2.keil.com/mdk5/learn

[4] ARM: *MDK5 Software Packs*. [Online; navštíveno 25.01.2019].
URL http://www.keil.com/dd2/pack/

[5] Atlassian: *Bamboo Best Practice - Using Agents*. [Online; navštíveno 25.01.2019].
URL https://confluence.atlassian.com/bamboo/bamboo-best-practice-using-agents-414189352.html

[6] Atlassian: *Triggering a Bamboo build from Bitbucket Cloud using Webhooks*. [Online; navštíveno 25.01.2019].
URL https://confluence.atlassian.com/bamboo/triggering-a-bamboo-build-from-bitbucket-cloud-using-webhooks-873949130.html

[7] Baron, D.; Klausner, T.: *zipcmp(1) - Linux man page*. [Online; navštíveno 25.01.2019].
URL https://linux.die.net/man/1/zipcmp

[8] Cobéna, G.; Abiteboul, S.; Marian, A.: *Proceedings of the 18th International Conference on Data Engineering*. IEEE, 2002, ISBN 0-7695-1531-2.

[9] Cutter, R.: *Continuous Integration and Bamboo*. [Online; navštíveno 25.01.2019].
URL http://www.blendedperspectives.com/wp-content/uploads/2014/04/Bamboo-and-CI-cutterryan.pdf

[10] Filippov, D.: *PyCharm by JetBrains - Powerful Python and Django IDE*. [Online; navštíveno 25.01.2019].
URL https://confluence.jetbrains.com/display/PYH/PyCharm+IDE+and+Python+Plugin+for+IntelliJ+IDEA

[11] Forcier, J.: *Fabric's documentation*. [Online; navštíveno 25.01.2019].
URL https://docs.fabfile.org/en/1.12.1/

[12] Foundation, P. S.: *Command Line*. [Online; navštíveno 25.01.2019].
URL http://www.keil.com/support/man/docs/uv4/uv4_commandline.htm

[13] Foundation, P. S.: *Process-based "threading" interface.* [Online; navštíveno 25.01.2019].
URL https://docs.python.org/2/library/multiprocessing.html

[14] Foundation, P. S.: *Subprocess management.* [Online; navštíveno 25.01.2019].
URL https://docs.python.org/2.7/library/subprocess.html

[15] Foundation, P. S.: *The ElementTree XML API.* [Online; navštíveno 25.01.2019].
URL https://docs.python.org/2/library/xml.etree.elementtree.html#module-xml.etree.ElementTree

[16] Foundation, P. S.: *Time access and conversions.* [Online; navštíveno 25.01.2019].
URL https://docs.python.org/2.7/library/time.html

[17] Lee, S. K.; Kim, D. A.: *X-Tree Diff+: Efficient Change Detection Algorithm in XML Documents.* Division of Information and Computer Science, Dankook University, 2005, ISBN 978-3-540-36679-9.

[18] Limited, A.: *Creating email and MIME objects from scratch.* [Online; navštíveno 25.01.2019].
URL https://docs.python.org/2/library/email.mime.html

[19] Maler, E.; Sperberg-McQueen, C. M.; Paoli, J.; aj.: *Extensible Markup Language (XML) 1.0 (Fifth Edition).* [Online; navštíveno 25.01.2019].
URL https://www.w3.org/TR/xml/

[20] NXP: *MCUXpresso Integrated Development Environment (IDE).* [Online; navštíveno 25.01.2019].
URL https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE?tab=Documentation_Tab

[21] Sha, E.; Han, S.-K.; Xu, C.-Z.; aj.: *Embedded and Ubiquitous Computing.* Springer-Verlag Berlin Heidelberg, 2006, ISBN 978-3-540-36681-2.

[22] Systems, I.: *Build from the command line.* [Online; navštíveno 25.01.2019].
URL https://www.iar.com/support/tech-notes/general/build-from-the-command-line/

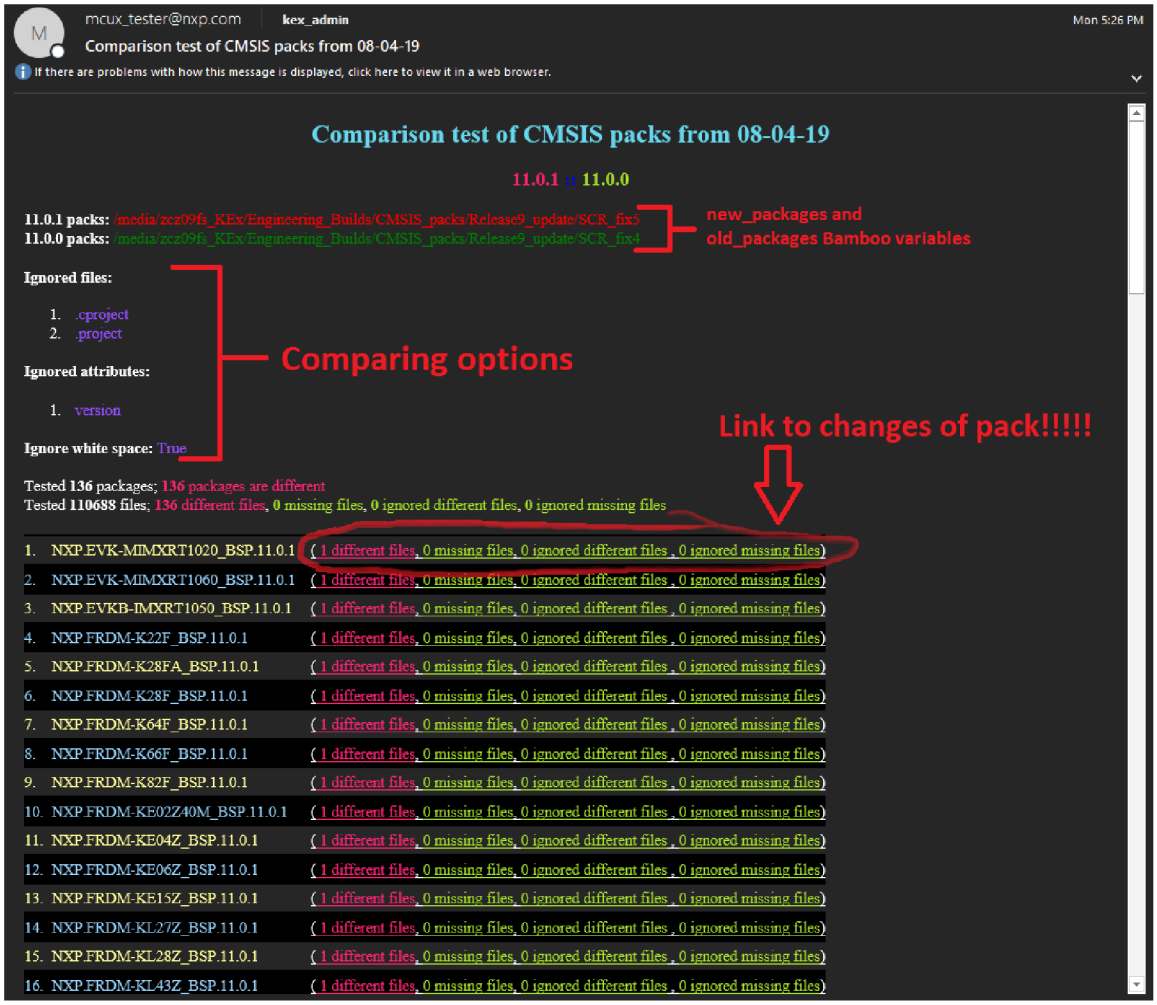
[23] Systems, I.: *IDE Project Management and Building Guide.* [Online; navštíveno 25.01.2019].
URL http://ftp.iar.se/WWWfiles/arm/webic/doc/EWARM_IDEGuide.ENU.pdf

[24] team, N. P. E.: *MCUXpresso SDK Builder.* [Online; navštíveno 25.01.2019].
URL https://mcuxpresso.nxp.com/en/welcome

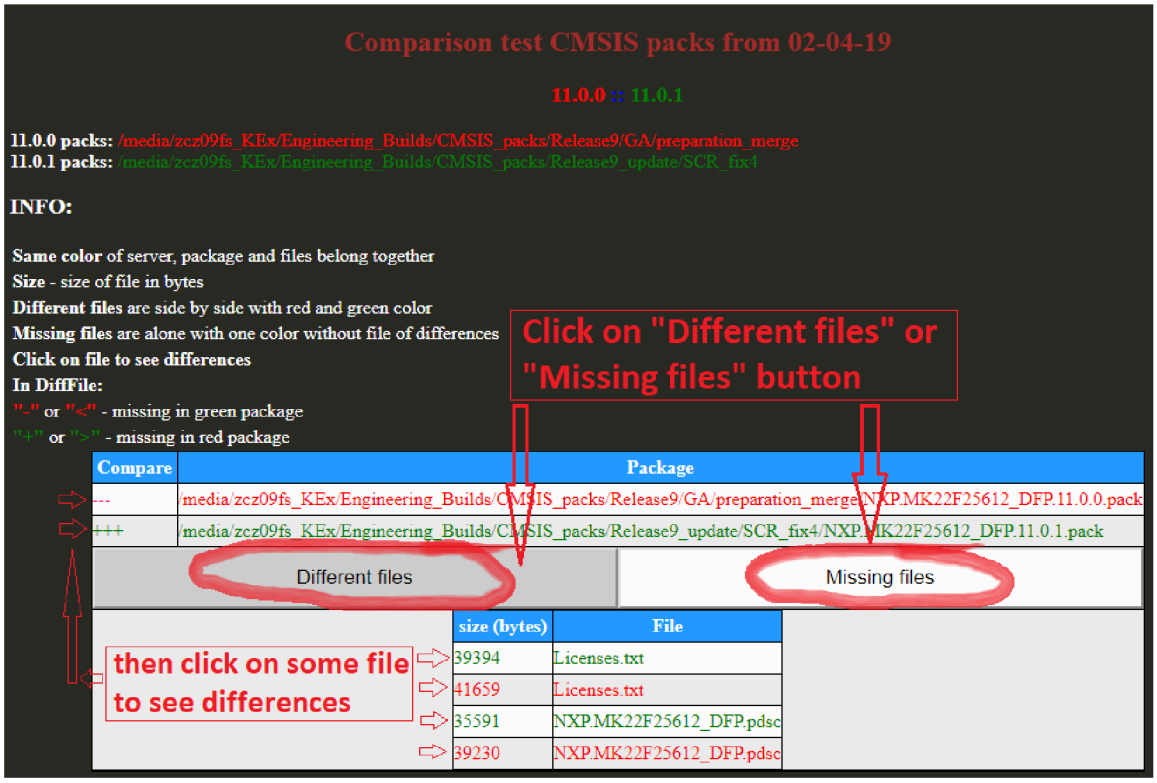
Příloha A

Výstupní rozhraní porovnávacího testu

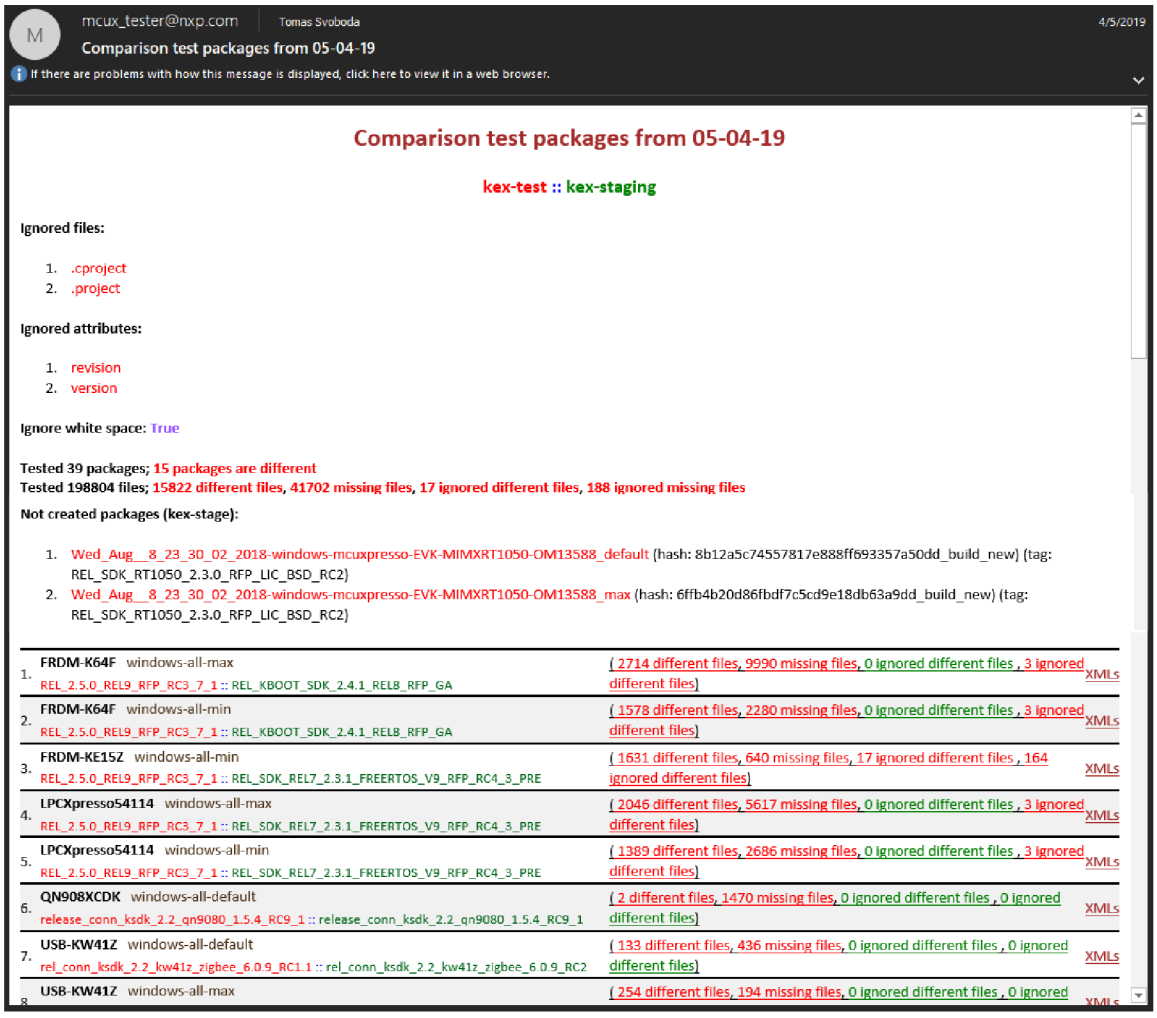
V této příloze najdete ukázky výstupního rozhraní porovnávacího testu pro CMSIS a SDK balíčky.



Obrázek A.1: Výstupní rozhraní porovnávacího testu CMSIS balíčků v emailu



Obrázek A.2: Výstupní rozhraní porovnávacího testu konkrétního softwarového balíčku



Obrázek A.3: Výstupní rozhraní porovnávacího testu SDK balíčků v emailu