VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATICKÝ SYSTÉM AKTUALIZACE OBSAHU CLOUDOVÉHO SYSTÉMU PRO DISTRIBUCI SOFTWAROVÝCH KOMPONENT

THE AUTOMATIC UPDATE MECHANISM OF SOFTWARE PACKAGES IN CLOUD DISTRIBUTION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TOMÁŠ WILLASCHEK**
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: **Willaschek Tomáš**
Program: Informační technologie
Název: **Automatický systém aktualizace obsahu cloudového systému pro distribuci softwarových komponent**
The Automatic Update Mechanism of Software Packages in Cloud Distribution System
Kategorie: Databáze

Zadání:

1. Seznamte se s interními nástroji pro generování konzistentních sestav programových komponent a s nástrojem pro kontinuální integraci Atlassian Bamboo.
2. Na základě získaných znalostí navrhněte skript pro předzpracování popisu konzistentní sestavy programových komponent pro proces nasazování těchto sestav na distribuční systém.
3. Navrhněte formát předzpracovaného popisu konzistentní sestavy softwarových komponent.
4. Navrhněte a realizujte akceptační test sestavy programových komponent pro nasazeni na distribuční systém.
5. Navrhněte a pomocí skriptovacího jazyka Python a nástroje pro kontinuální integraci Bamboo realizujte automat procesu naplnění databáze distribučního systému programových komponent o informace o nově přidávané sestavě programových komponent.
6. Vyhodnoťte efektivitu aktualizace programových komponent po nasazení realizovaného řešení.

Literatura:

- Dle pokynů vedoucího a odborného konzultanta.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

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 práce se týká webové služby, která slouží k distribuci programových komponent zákazníkům. Firma NXP je výrobcem polovodičových součástek. K těmto součástkám dodává programové komponenty, usnadňující uživatelům vývoj aplikací pro tyto součástky. Cílem práce je vytvořit proces nasazující sadu softwarových komponent na tento distribuční web a sadu testů ověřující funkčnost tohoto procesu. Součástí řešení byla analýza původního procesu, se záměrem předcházet známým problémům a postavit tak spolehlivější a efektivnější řešení.

Abstract

This bachelor thesis deals with web services that are used to distribute software components. NXP is a manufacturer of semiconductor devices. NXP also produces software component and tools for development of embedded software applications for those semiconductor devices. The aim of this work is to create a process that will deploy the software components on the distribution website and to create tests to verify functionality of this process. The solution includes an analysis of the original process, with the intention of preventing known problems and thus building a more reliable and efficient solution.

Klíčová slova

nasazení, distribuce softwarových komponent, automatizace, distribuční web

Keywords

deployment, software component distribution, automatization, distribution web

Citace

WILLASCHEK, Tomáš. *Automatický systém aktualizace obsahu cloudového systému pro distribuci softwarových komponent.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Adam Crha,

Automatický systém aktualizace obsahu cloudového systému pro distribuci softwarových komponent

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Adama Crhy Další informace mi poskytl pan Ing. Petr Kraus. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Willaschek
20. května 2019

Poděkování

Rád bych poděkoval svému konzultantovi, kterým je pan Ing. Petr Kraus, za skvělé vedení a odborné rady, týkající se právě této práce. Dále bych rád poděkoval svému vedoucímu, kterým je pan Ing. Adam Crha, za skvělé rady týkající se obsahu a vzhledu práce.

Obsah

1 Úvod **3**

2 Systém distribuce SDK **5**
2.1 Součásti procesoru 6
2.2 Unifikace dat 7
2.3 Distribuce dat 8

3 SDK Generátor **10**
3.1 Historie generátoru 11
3.2 Generátor v současnosti 12
3.2.1 Formáty dat 13
3.2.2 Fáze generátoru 14

4 Modul Webdata **16**
4.1 Implementace struktury pro vytvoření XML souboru 17
4.2 Vytvoření konfiguračního souboru pro web 19
4.3 Validace 20

5 Akceptační testy **22**
5.1 Realizace testů 22

6 Atlassian Bamboo **24**
6.1 Funkce Bamboo při vývoji generátoru 25
6.2 Role Bamboo při nasazení na produkční web 25
6.2.1 Nevýhody Bamboo v tomto projektu 25

7 Nasazení softwarových komponent na produkční web **26**
7.1 Starý proces nasazování 27
7.2 Nový proces 27
7.2.1 Přípravný proces 28
7.2.2 Proces nasazení 28
7.2.3 Post proces 29
7.2.4 Implementace dílčích částí 29
7.3 Rozšíření 30

8 Efektivita nového procesu **31**

9 Závěr **33**

Literatura **34**

Kapitola 1

Úvod

V dnešní době jsou populární chytrá zařízení, která zvládnou bez lidského zásahu reagovat na nejrůznější situace, chovají se za každých okolností bezpečně a intuitivně. Systému, který běží na těchto zařízeních a vykonává požadovanou činnost, je nazýváno vestavěný. Na trhu je dnes možné najít mnoho firem, které vyrábějí součástky, na kterých poté tyto systémy fungují. Jednou z těchto firem je také NXP Semiconductors. Tato firma vyrábí polovodičové součástky a aby je mohla úspěšně prodávat, musí k nim dodávat také základní programové vybavení, které obsahuje programy a data k inicializaci a základní konfiguraci těchto součástek. Dále k nim dodává i nízkoúrovňové ovladače periferních modulů a poté vyšší vrstvy, kterým se říká middleware. Tento celek je základem pro vznik systémového vývojového nástroje (SDK), který obsahuje veškerý základní software a manuály pro zprovoznění jednoho konkrétního zařízení. Obrázek 1.1 zachycuje celý tento systém, rozložený do jednotlivých hardwarových a hlavně softwarových vrstev, které jsou doplněny o aplikační kód, vytvořený samotným uživatelem. Tato práce je věnována bezpečnému nasazení těchto softwarových dat na distribuční systém, kde si uživatel přehledně volí obsah konkrétního SDK balíku, který mu je po sléze doručen.

Kapitola 2 je zaměřena na popis systému jako celku. Vysvětluje co to jsou data, jak vznikají a jakým způsobem se následně dostanou na produkční systém, skrze který si uživatel konfiguruje obsah SDK balíků, které mu jsou následně doručeny.

V kapitole 3 jsou vysvětleny základní a důležité pojmy související právě s SDK generátorem, daty a SDK balíky. Dále je zde popsán starý proces generování SDK balíků, jeho nevýhody a důvody vzniku nového generátoru. V poslední části jsou popsány obecně fáze generátoru, které při spuštění v korektním pořadí vytváří požadovaný SDK balík.

V kapitole 4 je vysvětlen důvod vzniku a využití souboru, který obsahuje základní informace o datech, která mají být nasazena na distribuční systém a ze kterých jsou následně sestavovány SDK balíky. V této kapitole je také obecně popsána struktura, díky které může být tento soubor jednoduše vytvořen v požadovaném formátu. Poslední část kapitoly je věnována validaci vzniklého souboru.

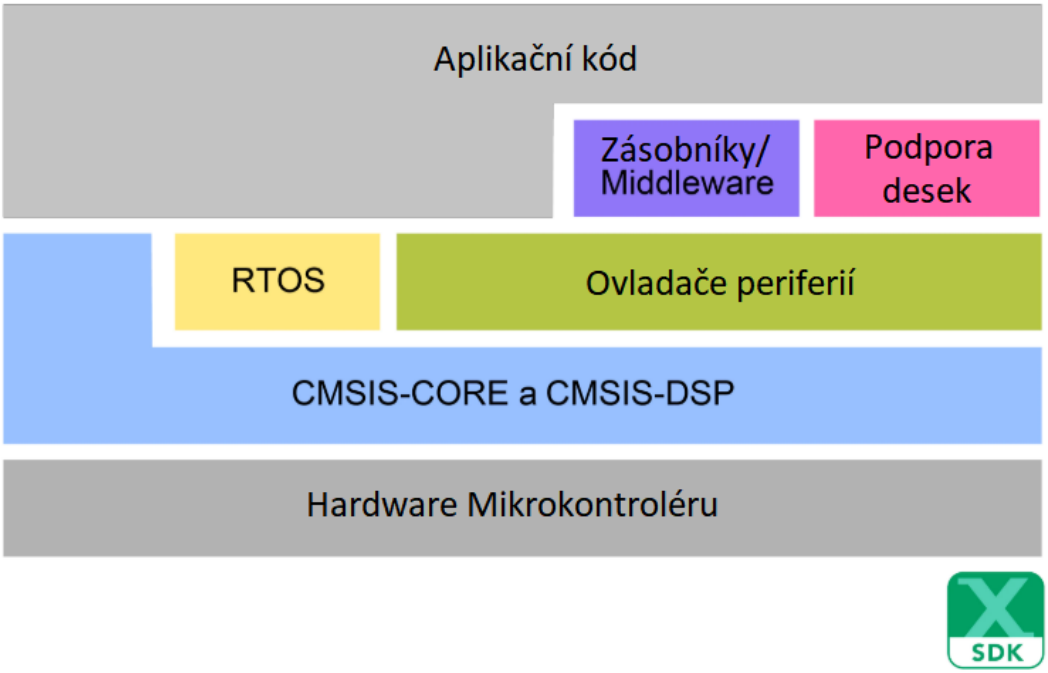
V kapitole 5 je popsán důvod, proč je nutností mít akceptační testy. Jsou zde také popsány jednotlivé dílčí testy, které jsou navrženy za účelem základní validace dat tak, aby byla akceptovatelná. V poslední řadě je zde zmíněn důvod, proč tyto testy zatím nejsou plně nasazeny a proč na ně zatím není brán takový zřetel.

V kapitole 6 jsou popsány produkty firmy Atlassian a jejich použití obecně, ale i použití v tomto projektu. Dále je zde část věnována právě produktu Bamboo, díky němuž lze jednoduše realizovat proces nasazení. V této části jsou zmíněny výhody i nevýhody samotného

produktu a také jiné produkty, které jsou právě v kombinaci s Bamboo mocným nástrojem pro nasazení softwaru.

Poslední části zpracování dat, tedy když jsou připravena a je potřeba je distribuovat, je věnována kapitola 7. V této části je popsán nový a efektivnější způsob, jak aktualizovat obsah produkční databáze, ale také starý způsob a chyby v něm. Součástí této kapitoly jsou také možná vylepšení tohoto procesu jako celku

V závěru práce, v kapitole 8, je popsána efektivita celého nově vytvořeného procesu. Jsou zde popsány vylepšení a jejich dopad jak pro generátor, tak pro samotný proces nasazení dat na distribuční systém.



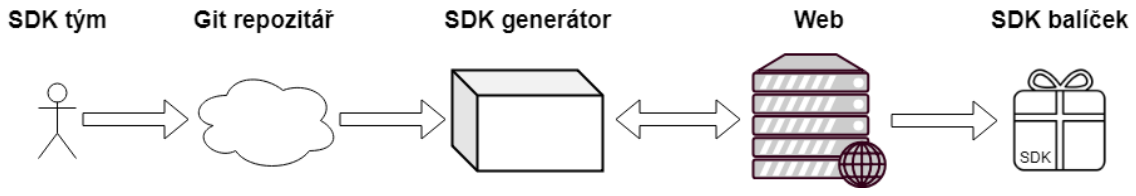
Obrázek 1.1: Obsah struktury SDK balíku. CMSIS-core a CMSIS-DSP – knihovny a hlavičkové soubory pro procesor. RTOS [20] – základní operační systém. Ovladače periferií – základní programy a funkce pro běh. Zásobníky/middleware – software rozšiřující funkčnost. Podpora desek – programy a knihovny pro konfigurace periferií a pinů desek. Aplikační kód – příkladové projekty. Převzato z [18].

Kapitola 2

Systém distribuce SDK

Vytvoření jednoho SDK balíku je komplexní proces, který se dá shrnout do tří hlavních kroků. Pro vizualizaci jsou tyto kroky zachyceny podrobněji na obrázku 2.1. Jak lze vidět v prvním kroku, tým lidí vytváří manuály, knihovny, příkladové projekty, podpůrné aplikace a mnoho dalších kódů, které v té chvíli ještě nejsou specifické pro daný procesor, ale jsou rozděleny do částí, které jsou specifické například pro celou rodinu procesorů a hlavně se jedná o co nejobecnější šablony. Tyto jednotlivé části se ukládají do repozitářů a to hlavně z důvodu snadnější správy a možnosti verzování. K těmto souborům je potřeba vytvářet další popisné soubory, které specifikují, jaké části se mají spojit dohromady, aby výsledný celek dával smysl.

Ve druhém kroku se tyto části, tedy příkladové projekty, knihovny a další kódy unifikují na míru jednotlivým procesorům. Tato unifikace nebo také generování, znamená spojení až několik souborů, které popisují části specifické pro daný procesor, například knihovní makra pro přístup do registrů či velikosti pamětí. Tímto vznikají soubory již specifické pro jeden procesor, které jsou již v takovém stavu, ve kterém se nacházejí v SDK balíku. Toto generování obstarává SDK generátor, na jehož vstupu jsou surová data a na výstupu například kompletní SDK balík. Obsah těchto dat se týká převážně procesorů, ale mohou se zde nacházet i kódy pro vývojové desky. Vývojová deska je deska, na které se fyzicky nachází jeden či více konkrétních procesorů a přináší možnost používání rozhraní (například USB) namísto klasických pinů, jelikož procesor samotný těmito rozhraními nedisponuje. Tyto data specifická pro desky se týkají právě periferií, které se na konkrétní desce fyzicky nacházejí. Při tomto generování vzniká i popisující soubor, který obsahuje důležitá data, podle kterých si uživatel vybírá konkrétní zařízení, pro který chce vyvíjet svou aplikaci. Tento soubor slouží jako zdroj dat pro distribuční web.



Obrázek 2.1: Proces vzniku softwarových komponent, přes jeho zpracování, s následným doručením ve formě SDK balíku koncovým zákazníkům

Distribuční web je aplikace, která pomáhá uživateli s výběrem konkrétních komponent, které potřebuje pro svůj projekt, ale také s výběrem hardwarové platformy, která všechny

tyto komponenty podporuje a disponuje procesorem, který rovněž splňuje jeho požadavky. Dostupné hardwarové platformy jsou procesory, vývojové desky a kity. Kity jsou vývojové desky rozšířené o jakýsi slot, do kterého se dá jednoduše připojit jiná deska, která rozšiřuje funkčnost té původní. Těmto rozšiřujícím deskám se nazývá shield. Tento shield už nedisponuje plnohodnotným procesorem, slouží pouze a jen k rozšíření funkcionality původní desky. Tento celek desky a shieldu je pak nazýván kitem. Velmi důležitou funkcí webu, je generování SDK balíků, tedy zprostředkování předání dat generátoru, který tento balík vytvoří a následnému doručení tohoto balíku koncovému uživateli. Toto generování probíhá na základě informací, které vybírá uživatel, tedy konkrétní zařízení a další softwarové aplikace, které jsou procesorem či deskou podpořeny.

2.1 Součásti procesoru

K úspěšnému prodeji procesoru je potřeba mnohem víc, než jen navrhnout a vytvořit kus křemíku obdélníkového tvaru s mnoha nožičkami. Potencionální zákazník, který si chce vytvořit vlastní aplikaci neví, jak konkrétní součástka funguje, proto je potřeba detailně popsat každý registr, které hodnoty kam zapsat, jak získat potřebná data, tedy jak celý procesor funguje. Za těmito účely je ke každému procesoru manuál, který může být rozsahu i několika set stran. Často lze nalézt manuál pro konkrétní rodinu těchto součástek. Neznamená to, že jsou procesory totožné, ale že principiálně fungují stejně, například mají stejně pojmenované registry, jejich počet a systém jejich použití. Tyto procesory se často mohou lišit jen ve velikosti pamětí či frekvencí.

Zákazník, který si zakoupí tuto součástku a dostane k ní manuál, ale zdaleka ještě nemá vyhráno. Není úplně jednoduché, napsat si vlastní aplikaci jen s manuálem. Ke každému procesoru se proto vytvářejí příkladové projekty, které demonstrují použití jednotlivých částí procesoru jako celku, jeho zprovoznění, ale i komunikaci a využití periferií, které se vyskytují na konkrétním kitu či samotné desce, která je tímto procesorem osazena. Zpravidla se vytvářejí i příkladové projekty, které demonstrují, jak pracovat s jednotlivými softwarovými komponentami (middleware), které jsou procesorem podporovány a jsou na něm otestovány.

Middleware slouží především k jednoduchému rozšíření funkcionality jak desky, tak i procesoru. Jedná se buď o samotnou softwarovou komponentu, nebo o komponentu, která potřebuje ke svému fungování přítomnost nějakého hardwaru. Obě tyto varianty přinášejí uživateli možná rozšíření pro vývoj a běh jejich aplikace bez toho, aniž by uživatel muset požadovaný software či hardware implementovat a testovat. Tento software je dodávám jako volitelný pro jednotlivé procesory, na kterých byl předem otestován a spolu s ním jsou dodány i příkladové projekty, které demonstrují jeho použití. Klasickým příkladem takové softwarové komponenty je souborový systém, konkrétně například FatFS [1]. Tato komponenta je specifická právě pro vestavěné systémy a je navržená tak, aby fungovala na co největším možném počtu procesorů s minimem úprav. Přináší do vestavěných systémů možnost základní práce se soubory, jako je otevírání, editace a zavírání, ale také pokročilejší funkcionalitu, jako je vytváření adresářových struktur.

Pří použití těchto komponent je důležité brát v potaz jejich hardwarové požadavky. Existují komponenty, které vyžadují minimum paměti, ale také existují takové, které vyžadují více zdrojů. Není zaručeno, že každý procesor zvládne provoz několika náročnějších komponent najednou.

Poslední nezbytnou částí, která vzniká spolu s procesorem, jsou knihovny, které obsahují nejrůznější masky, konstanty a deklarace funkcí, značně usnadňující vývoj samotného

softwaru. Konstanty v knihovně pro procesor nejčastěji nahrazují adresy skrze které se přistupuje k jednotlivým registrům. V knihovnách pro komponenty se vyskytují konstanty, skrze které se dá přistupovat k daným komponentám a pracovat s nimi. Celkově zlepšují čitelnost kódu, který se s jejich použitím podobá více textu než změti nesrozumitelných znaků. Masky se zpravidla definují pro jednotlivé příznaky[1] procesoru, buď pro získání jejich hodnoty a nebo pro nastavení hodnoty na konkrétní místo v registru. Každá maska je pojmenovaná jako konstanta. Poslední důležitou částí knihoven jsou deklarace funkcí, které jsou využity při práci s platformou či komponentou. Tyto funkce vytvářejí týmy, které detailně znají daný prvek a proto bývá zpravidla bezpečnější je použít, než navrhovat vlastní.

Všechny výše zmíněné části jako konstanty a masky jsou detailně popsané v dokumentaci, která se na ně odkazuje a pomocí nich popisuje, jak pracovat například s jednotlivými registry nebo jak do nich uložit konkrétní hodnotu.

2.2 Unifikace dat

Data, které vytvoří SDK týmy jsou použitelná jen z části. Dají se použít manuály, které jsou specifické pro procesor nebo jejich celou rodinu, protože tyto manuály se zpravidla neskládají z více souborů. Tyto návody jak pracovat s procesory, často čítají i několika set stran, které popisují konkrétní případy užití funkcí a softwarových komponent dané rodiny. Pokud se nějaký procesor liší, je v manuálu uveden případ použití právě pro něj. Těchto výjimek nebývá mnoho, proto je jejich dovysvětlení dostačující a není nutné vytvářet manuál, který bude specifický pro jedinou výpočetní jednotku. Unifikace se týká převážně zdrojových kódů. Soubory s tímto obsahem jsou logicky rozděleny na části společné a unikátní a při tomto procesu se z každé části vygeneruje z co nejobecnějšího popisu vždy jeden soubor, který obsahuje všechny potřebné informace a je specifický vždy pro konkrétní procesor. Pro toto generování je potřeba znát, které části tvoří logický celek, a proto je nutné disponovat soubory, které obsahují tyto informace. Výsledkem tohoto procesu jsou soubory, které jsou specifické pro jeden procesor. Nachází se mezi nimi právě knihovny s konstantami potřebnýma pro vývoj, ale také příkladové projekty.

V tomto procesu je určitě na místě soubory třídit do adresářových struktur, bez kterých by výsledek byl absolutně zmatečný a to i pro osobu zaobírající se detailně procesem transformace nebo vytváření těchto souborů. Dalo by se to přirovnat k otevření koše typického uživatele na osobním počítači, kde se nachází mnoho souborů, o kterých ani sám uživatel neví, že je kdy použil. Soubory jsou proto setříděny podle hardwarové platformy, kde jsou vždy ve stejném adresáři soubory společné pro jeden procesor či desku, dále podle částí specifických pro komponenty, manuály či jinou dokumentaci a mnoho dalších kategorií a podkategorií, kde každá má specifický význam. Příkladový adresář určený pro konkrétní procesor a pojmenovaný jeho jménem obsahuje zdrojové kódy pro zprovoznění a běh tohoto čipu, ale také soubory specifické pro konkrétní vývojová studia, které obsahují základní nastavení tohoto studia a načtení základní struktury projektu. V adresáři určeném pro desky a kity se nacházejí primárně příkladové projekty, které demonstrují užití periferií kitu či desky osazené konkrétním procesorem a komponenty, které jsou pro danou sestavu otestovány. Adresář pro komponentu obsahuje zdrojové kódy k jejímu zprovoznění, pří-

[1] Příznak – jeden či více bitů v registru, jejichž kombinace značí specifickou situaci nebo nastavuje důležité parametry

kladové projekty a dokumentaci, která se často může odkazovat i na stránky konkrétního výrobce.

Poslední část, která se generuje a souvisí s procesem unifikace, tvoří soubory, které nemusí být součástí SDK balíku. Znamená to tedy, že se tyto soubory nemusejí vůbec fyzicky dostat ke koncovým uživatelům, ale toto je nečiní méně důležitými. Právě naopak. Část tohoto obsahu jsou licence jak pro sadu těchto dat jako celku, tak pro některé komponenty, které vyžadují přítomnost samostatné licence. Tyto části jsou z právního hlediska velmi důležité a nesmí být opomenuty. Například s licencemi pracuje distribuční web, který je podle potřeby zobrazuje uživatelům a nutí je k jejich odsouhlasení, pokud chtějí získat svůj SDK balík. Další část je tvořena soubory, popisující již vygenerované soubory, například která komponenta vyžaduje licenci, čí který procesor patří logicky ke které desce. Bez těchto souborů by bylo možné sestavit SDK balík pouze v případě, že by generátor obstarával jak unifikaci souborů, tak i finální zabalení dat s následným vytvořením SDK balíku a to vše jako jeden celý proces. V realitě je ovšem tento proces přizpůsoben a rozdělen do určitých kroků, které jsou detailněji vysvětleny v kapitole 3, a to proto, aby bylo výsledné generování rychlejší. V případě, že se vytváří pouze unifikované soubory a ne celé SDK balíky, se vytváří i konfigurační soubor, který slouží pouze pro potřeby distribučního webu a obsahuje všechna potřebná data o aktuálně generovaných souborech, které tento web přehledně zobrazuje uživatelům a jinak s nimi pracuje. Poslední nedílnou částí jsou manifesty, které spojují SDK balíky a MCUXpresso IDE [17] tak, že s nimi toto vývojové prostředí umí pracovat a rozpozná jejich obsah. Pojem manifest je vysvětlen dále.

Neúspěch této unifikace značí buď chybu v datech, na kterou SDK generátor, zprostředkovávající tento proces, správně upozorní, nebo chybu v samotném generátoru. Unifikace je navržena tak, aby při jejím běhu probíhala validace dat takovým způsobem, že výsledek bude obsahovat minimum chyb. Pokud takový proces selže, značí to potencionální problém. Tento problém mohl být do kódu zavlečen nově, nebo se může jednat o odhalení chyby, které si nikdo delší dobu nevšiml. Chyby nejsou nic ojedinělého, jedná se totiž o obrovský proces, který slučuje mnoho týmů a zdrojových kódu do jednoho celku, avšak tyto chyby často mohou brzdit celý vývoj. Aby se předcházelo takovým chybám v době, kdy se dokončují práce na konkrétních datech, která budou následně nasazena na distribuční web, spouští se takové generování (unifikace) denně, protože vývojáři využívají různé zkratky pro vytváření konkrétních částí kódu a dat. Kdyby nevyužívali těchto zkratek, vývoj by trvat příliš dlouho. Jako příklad je možno uvést poslední SDK release[2], jehož generování trvá na obyčejném počítači až 20 hodin. Při chybě v generátoru je vypsán klasický traceback[3]. Toto generování sleduje více pověřených osob, které podnikají patřičné kroky k tomu, aby další generování proběhlo bez chyb.

2.3 Distribuce dat

Co s připravenými daty? Jak je šířit mezi zákazníky? Jakou část z tohoto celku skutečně zákazník potřebuje? To jsou hlavní otázky, které jsou zodpovězeny v této části.

Jakmile jsou data připravena, je potřeba je rozšířit mezi zákazníky a to ideálně co nejrychleji. Za tímto účelem je vytvořen distribuční web. Tento web získává informace o těchto datech ze souboru, který vzniká při jejich generování. Soubor svým obsahem definuje chování webu a to tak, že obsahuje popis veškerých potřebných vazeb, které je nutné respekto-

[2]Release – Milník, při kterém SDK tým vydává nová data či revize a jiné opravy k dispozici pro externí uživatele

[3]Traceback – původ vzniku chyby, tedy které funkce a v jakém pořadí byly zavolány před jejím vznikem

vat při konfiguraci obsahu SDK balíku, ale také konkrétní informace o jednotlivých částech tohoto celku, tedy důležité údaje o procesorech, dostupné komponenty pro konkrétní procesory a mnoho dalších, které jsou relevantní pro uživatele konfigurujícího si obsah SDK balíku. Tento celek nese pracovní název superset. Detailnější definice tohoto pojmu je vysvětlena v kapitole 3.

Celý superset je spolu s tímto popisným souborem nahrán na distribuční server a obsahem popisujícího souboru je aktualizován obsah databáze tohoto webu. Za pomocí této databáze web poté zobrazuje jednotlivá data přehledně uživatelům a dává jim možnost aplikovat na ně filtry či různé pohledy, které pomáhají znalým, ale především neznalým uživatelům vybrat si hardware, na kterém chtějí vyvíjet svůj projekt. Pro každý tento hardware si uživatel dále vybírá z množství komponent, které jsou nezbytné pro správnou funkčnost jeho projektu a z množství podpořených vývojových studií, ve kterých chce projekt vyvíjet.

Jakmile má uživatel nakonfigurovaný obsah svého SDK balíku, web vytvoří ze specifikovaných informací požadavek, který předá SDK generátoru. Generátor je součástí každého supersetu a to z toho důvodu, protože je vytvořen na míru aktuální verzi dat. Tento generátor na základě specifikovaných informací následně shromáždí všechna potřebná data a vytvoří z nich SDK balík. Z tohoto postupu lze jistě poznat, že SDK balík obsahuje pouze konkrétní část dat, kterou uživatel chce. Nebylo by logické ani efektivní posílat mu všechna data, když z nich použije jen nepatrný zlomek. Děje se tomu tak z mnoha důvodů. Jeden z nejdůležitějších je ten, že část těchto dat může být vytvořená na míru pro konkrétní zákazníky a mají být dostupná jen a pouze pro ně. Druhým podstatným důvodem je velikost. Dříve, když ještě nebyl vysokorychlostní internet standardem v každé domácnosti, bylo velmi obtížné a zdlouhavé stáhnout si sadu dat, jejíž velikost se pohybovala v řádech gigabytů. Toto byl i jeden z důvodů, proč vznikl právě tento distribuční web, ruku v ruce s SDK generátorem.

Jak již bylo zmíněno, jedná se o složitý a velký proces, ve kterém se nacházejí chyby, které je potřeba opravovat. Pokud se najde taková chyba v datech po té, co už byla distribuována, je nutné ji opravit. Taková oprava se provádí buď do dalšího SDK releasu, nebo se vytvoří fix verze a to v závislosti na tom, jak je chyba kritická. Pokud jsou data pro jednotlivé části releasu nevalidní, je nutné suspendovat jejich distribuci. Za tímto účelem funguje administrátorská část webu, která umožňuje jednoduchou správu těchto dat. Základní funkčností je třídění uživatelů a hardwarových platforem do skupin, díky čemuž se na web mohou nahrát i data pro specifické zákazníky, kteří tímto dostanou výlučný přístup. Administrátorská část také umožňuje zneaktivnění dat, u kterých byla objevena kritická chyba, což mohou být komponenty, hardwarové platformy, nebo i celé releasy.

Kapitola 3

SDK Generátor

SDK generátor byl doposud jakousi černou skříňkou, které upravovala korektní vstup na korektní výstup. Jelikož je tato černá skříňka nezbytnou částí celku a část této práce je její součástí, je na místě ji popsat více detailněji.

Pro bližší seznámení se s tímto procesem je potřeba dovysvětlit pojmy, které již mohly být zmíněny a úzce s generátorem souvisí. Tyto pojmy spadají do množiny výstupů generátoru a jsou to SDK a superset.

Superset

Superset je množina souborů, které jsou součástí jednoho releasu. Tuto množinu je nejprve potřeba sestavit, tedy vygenerovat všechny potřebné souboru do stavu, v jakém se vyskytují v SDK balících a vytvořit k nim další nezbytné soubory. Při tomto sestavování, je potřeba zachovat konzistenci dat, tedy zaručit, že jsou všechny vazby mezi daty splněny. Superset, respektive všechny jeho dílčí části, které jsou specifické pro každou hardwarovou platformu, jsou dále testovány a posléze, jako jeden celek, umístěny na web. Prostřednictvím webu jsou z této množiny sestavovány SDK balíky, jejichž obsah si specifikují uživatelé.

Superset se vytváří v této podobě hlavně ze dvou důvodů. Prvním z nich je rychlost, jakou se z něj generují SDK balíky. V dnešní době se očekává, že zákazník, který chce svůj balík, na něj nebude čekat půl dne, ale dostane jej v řádech jednotek minut. Toto generování ze surových dat trvá i na výkonných strojích mnohdy více než deset minut, ale ze supersetu je možné vytvořit výsledný balík již v jednotkách minut. Při generování z této před připravené množiny, dochází také k markantní úspoře času při vytváření balíků, které se liší pouze v obsahu komponent, tedy hardwarová platforma je shodná. V tomto případě se spojují do jednoho celku již vytvořená data pro hardwarovou platformu, doplněná o komponenty. V případě surových dat je nezbytné vytvářet data pro hardwarovou platformu pro každý balík vždy od začátku. Druhým důvodem pro tento postup je validace dat, která probíhá s vytvářením supersetu velice důkladně, čímž jsou odhalovány chyby a zabraňuje se tím jejích šíření.

Soubory v supersetu jsou logicky setříděny do adresářů, což celkově zvyšuje přehlednost a orientaci v těchto datech. Při vytváření SDK balíku právě z této struktury je zapotřebí jen několika málo popisujících souborů, které definují potřebné vazby mezi platformami, komponentami a ostatními daty. Tyto soubory vznikají spolu se supersetem.

Superset je dále jakýmsi milníkem SDK týmů, které jeho vytvořením začínají vyvíjet nové kódy, testovat limity procesorů a vylepšovat je. Často je tento pojem v tomto

kontextu nahrazen releasem, protože právě release je dodáván v podobě tohoto supersetu. Pokud je v supersetu nějaká chyba, tak je nutné ji samozřejmě opravit, avšak menší chyby se opravují do vývojových větví[1] a vydají se až s dalším releasem, protože oprava takové malé chyby by obnášela příliš mnoho administrace. Kvůli větším chybám se vytváří opravy buď pro celý superset, nebo jen pro jeho konkrétní části, to záleží na druhu a rozsahu chyby.

SDK

SDK, tedy systémový vývojový nástroj nebo software development kit, je sada všech potřebných souborů, programů, ovladačů a knihoven, které jsou nutné pro zprovoznění projektu ve vývojovém prostředí (IDE[2]), nebo bez něj. Dále jsou obsaženy zdrojové kódy požadovaných hardwarových platforem a vybraných komponent. Tyto kódy jsou navíc doplněny o příkladové projekty, které usnadňují vývoj softwaru jak pro hardwarovou platformu, tak pro komponentu. Aby IDE dokázalo zpracovat tento obsah, je zapotřebí nějakého popisu. Souboru, který popisuje tento obsah, je nazýváno manifest. IDE tento soubor načte a díky jeho obsahu uživateli přehledně zobrazí, co všechno může s daným SDK dělat, jaké komponenty a hardware může použít, odkazuje na on-line zdroje s popisem, specifikacemi a dalšími údaji o výrobku. K SDK je přiložena i základní dokumentace, jak spustit projekt například v MCUXpresso IDE a dále popis výrobku obsahující známé problémy, podpořené komponenty a jiné důležité texty.

Adresářové struktura je prakticky totožná jako u supersetu, jen obsahuje podstatně méně souborů a to pouze ty, které souvisí s konkrétní hardwarovou platformou, její dostupné či nezbytné hardwarové a softwarové komponenty, uživatelem zvolené komponenty a relevantní příkladové projekty.

Ke každému SDK je možné stáhnout si i samostatný balíček, který obsahuje pouze dokumentaci. V tomto balíčku se nachází dokumentace k procesoru, ale i k těm komponentám, se kterými není práce tak snadná a nestačí jen popis v kódu.

3.1 Historie generátoru

Původní SDK generátor neměl takové široké spektrum použití. Generoval z maximálně obecných šablon instance pro jednotlivé procesory a vývojové prostředí, které byly doplněny o příkladové projekty. Tento superset byl distribuován jako celek, což přinášelo řadu problémů. Hlavním z nich je velikost celého supersetu. Tento celek, i přes to, že je nějak komprimovaný, může nabývat na velikosti v řádech gigabytů. Z dnešního pohledu se jedná o zanedbatelnou velikost, avšak tyto procesy vznikaly i deset let nazpět. Je důležité připomenout, že v té době nebylo standardem mít vysokorychlostní, stabilní internet. Často se tedy mohlo stát, že stahování po několika hodinách skončilo chybou a bylo nutné daná data stahovat zase od začátku.

Nedostatky tohoto řešení daly vzniknout distribučnímu webu, který podpořil distribuci jednotlivých SDK balíků, nepřímo umožnil restrikci částí supersetu, přidává možnost definovat závislosti mezi komponentami a umožňuje přidání a zobrazení nových informací. Tento web dále umožňuje sběr dat, která určují další směr vývoje. Jedná se převážně o data typu

[1] Větev – hlavní část verzovacích systémů, umožňuje vyvíjet kód mimo jeho hlavní část a také zkvalitňuje kontrolu nově vytvořeného kódu

[2] IDE – prostředí zaměřené na konkrétní jazyk, obsahuje editor kódu, kompilátor, debugger a další potřebné nástroje pro práci s daným jazykem

jaká platforma a které komponenty jsou mezi uživateli oblíbené. Současně s tímto webem vznikla druhá část SDK generátoru, přesněji další samostatný generátor, který zpracovával výstup původního. Tento generátor umožňovat právě vytváření SDK balíků s přesně daným obsahem, tedy kódy pro konkrétní platformu a uživatelem definovaný set komponent.

Toto řešení se dvěma generátory bylo ovšem velmi složité. První generátor vytvářel supersety, tedy základ SDK a příkladové projekty a druhý poté připravil tyto data pro nasazení na web a vytvořil k nim manifesty. Rovněž byl dvojí systém zápisu informací, pro každý generátor jeden. V celém tomto řešení bylo velmi obtížné hledat chyby a celý proces udržovat, a to zvláště proto, protože o první generátor se staral SDK tým a o druhý tým, který měl na starost právě web. Webový tým byl tímto nucen řešit problémy, které souvisely s vestavěným softwarem, na které neměl dostatek lidí, informací a ani to tak nebylo zamýšleno, protože tento tým má zajišťovat pouze distribuci.

Díky tomu, že existují dva generátory, o které se navíc starají různé týmy, pronikají do procesu nasazení a generování SDK balíků chyby. Je tomu tak z toho důvodu, protože SDK týmy se zároveň s daty starají pouze o první z generátorů, tedy po aktualizaci druhého generátoru nejsou jeho kódy a výstupy testovány tak důkladně, jak by bylo zapotřebí. Tyto chyby se většinou objeví až při spuštění samotného procesu nasazení, nebo dokonce až při sestavování SDK balíků, což už většinou bývá pozdě a v závislosti na závažnosti chyby, je potřeba je buď opravit co nejdřív, nebo až v dalším vydání. Tato oprava je také závislá na umístění chyby. Pokud se nachází v generátoru, webový tým ji po vyjasnění s SDK týmem může opravit, ale pokud je v datech, je potřeba tyto data opravit a dále informovat uživatele, kteří již vlastní balík s touto chybou, že ji obsahuje. Chyby se velmi často vyskytovaly právě v části generátoru, která vytvářela manifesty.

3.2 Generátor v současnosti

Aby byly odstraněny výše zmíněné problémy a došlo k výraznému usnadnění celého tohoto procesu, byl druhý generátor předán do rukou SDK týmu. Tento tým vytváří data, se kterými generátor pracuje, tudíž je oprava chyb o mnoho snadnější a rychlejší. Tímto předáním vzniká celý nový proces, kde oba generátory vznikají úplně od začátku, ale s o to dokonalejším návrhem, který odstraňuje všechny známe problémy starého generátoru, jako například dvojí systém popisování dat, výrazně usnadňuje celý proces a umožňuje mnoho nových funkcí. Tento nový návrh slučuje oba dva staré generátory do jednoho, který podporuje všechny stávající funkce, tedy umí generovat instance pro jednotlivé procesory, příkladové projekty, ale také manifesty. Sloučena je tímto také tvorba supersetu a to tak, že superset nově obsahuje i manifesty, které se vytvářely až při nasazení na web.

S tímto sloučením přichází také nový problém a to jak aktualizovat obsah webové databáze. Vytváření napojení generátoru přímo na web by nebylo efektivní ani jednoduché a to už z toho důvodu, že tyto procesy se nacházejí v oddělených repozitářích, tedy mají jinou strategii vývoje a jsou napsány v jiných jazycích. Z tohoto důvodu je vytvořen konfigurační soubor, který obsahuje všechna potřebná data, se kterými pracuje právě web. Tento soubor je také nově součástí supersetu a tvoří bránu právě mezi generátorem a webem. Správnému návrhu a vytvoření tohoto souboru je věnována kapitola 4.

Se sloučením generátorů přichází i další procesy pro zkvalitnění vývoje a distribuce. Generátor se každý den spouští s cílem vytvořit superset z aktuálně vyvíjených dat. Tento proces odhaluje chyby jak v generátoru, tak v datech velice rychle, což přináší i rychlejší opravu. Samozřejmě se jedná o automatizovaný stav, kdy se toto spouštění jednou nastaví, a poté už se jen upravují vývojové větvě repozitáře, což znamená minimální administraci.

Další ocenitelnou funkcionalitou je možnost vygenerovat si superset či SDK balík navzdory tomu, že data obsahují chyby. Samozřejmě tato funkcionalita není dostupná v případě, že požadovaná data jsou pro tento proces klíčová. V takovém případě končí generování chybou, jako obvykle. Tato funkcionalita umožňuje nahlédnout do vygenerovaných souborů a také zobrazuje více chyb, které se v datech nachází. Více chyb z důvodu, že v generátoru jsou speciální funkce pro načítání dat, a pokud tato funkcionalita není aktivována, proces je ukončen s první nalezenou chybou.

Poslední důležitou inovací je možnost zvolení si vstupu s výstupu. Starý generátor buď vytvářel z dat superset, nebo pracoval s ním samotným a generoval z něj SDK balíky. Nový generátor podporuje stejné vstupy, tedy surová data nebo superset, ale dále podporuje jak vytvoření supersetu, tak samotného SDK balíku, kde balík může být v komprimované podobě, tedy jak je doručován zákazníkům, nebo v rozbalené, což usnadňuje další vývoj. Dalším z možných výstupů je standard CMSIS [7], který byl vyvinut firmou Keil a liší od standardního SDK balíku, který je zmiňován v této práci. Součástí této inovace je i vylepšení konfiguračního souboru, který slouží jako zdroj informací, které má generátor zpracovávat.

3.2.1 Formáty dat

Metadata [4], tedy data popisující data, ze kterých probíhal starý proces nasazení, byla uložena ve formátu XML. Tento formát byl dobrý při začátcích tohoto projektu, tedy distribuce softwaru k mikročipům. Postupně s přibývajícími možnostmi však velikost těchto popisujících souborů roste, jak v počtu řádků, tak na disku a soubory se stávají pro člověka prakticky nečitelné. Je tomu tak z toho důvodu, že formáty typu XML se vytváří pomocí párových značek, mezi kterými se vyskytuje hodnota. Jelikož tyto značky musí být pojmenované, data mezi nimi nejsou pro lidské oko tak nápadná. Příkladový vzhled této struktury popisující část procesoru je zachycena v příkladu 3.1. Tato struktura obsahuje pouze základní informace a to jméno procesoru, velikosti pamětí, frekvenci a desky, na kterých je tento procesor testován a dodáván.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <devices>
    <device full_name="MK64FN1M0xxx12" name="MK64F12" frequency_mhz="120">
      <core name="Cortex-M4F"/>
      <total_memory ram_size_kb="256" flash_size_kb="1024"/>
      <evaluation_boards>
        <evaluation_board name="FRDM-K64F"/>
        <evaluation_board name="TWR-K64F120M"/>
        <evaluation_board name="HEXIWEAR"/>
      </evaluation_boards>
    </device>
  </devices>
</metadata>
```

Algoritmus 3.1: Příklad XML metadat pro popis procesoru

Z důvodu nevyhovujícího formátu metadat bylo nutné zvolit jiný. Nová data jsou tedy vytvářena ve formátu YAML [5], který již neobsahuje zmíněné párové značky, ale popisuje data ve tvaru klíče a hodnoty, kdy ke každému klíči je přiřazena nějaká hodnota. Tento

formát byl zvolen proto, protože první generátor vytvářející superset obsahoval data právě v tomto formátu, tudíž došlo k jeho sjednocení. Metadata se tímto stávají o dost čitelnější, jak je vidět na příkladu 3.2, který popisuje totožná data, jako předchozí příklad. Nevýhodou tohoto formátu je nemožnost validace dle schematu, avšak tuto nevýhodu snadno odmazává generátor, který tuto validaci provádí pomocí propracovaného načítání hodnot z těchto dat. Výhodou je jistě lepší čitelnost, ale také menší velikost výsledného souboru obsahující tento popis. Nový generátor je vytvořen v jazyce Ruby [13], který má širokou podporu právě pro formát YAML, a proto je práce s ním snadná.

```
device.hardware_data:
  section-type: manifest_content
  contents:
    devices:
    - id: MK64FN1M0xxx12
      full_name: MK64FN1M0xxx12
      name: MK64F12
      frequency_mhz: 120
      ram_size_kb: 256
      flash_size_kb: 1024
      core:
      - name: Cortex-M4F
      evaluation_boards:
      - FRDM-K64F
      - TWR-K64F120M
      - HEXIWEAR
```

Algoritmus 3.2: Příklad YAML metadat pro popis procesoru

3.2.2 Fáze generátoru

Způsob zpracování dat novým generátorem je rozdělen do čtyř fází, které jsou podrobněji rozvedeny dále. Obecně se dá říci, že první tři z nich jsou významné pro vznik supersetu a čtvrtá pro vznik SDK balíku. Je to dáno tím, že v prvních třech fázích se data generují a vytvářejí se k nim další potřebná. Pokud je tento postup aplikován pouze na část množiny dat, které spolu logicky souvisejí, vzniká tímto superset pro jednu konkrétní hardwarovou platformu. Čtvrtá fáze pouze upraví data do určité podoby, což obnáší i jejich komprimaci a vytvoření souboru, který je distribuován zákazníkům. Tato fáze není nutná pro superset, protože ten se distribuuje v nezpracované formě. Je tomu tak z toho důvodu, protože webová aplikace z něj následně sestavuje balíčky a nebylo by efektivní tuto operaci provádět nad komprimovanými daty.

Fáze 1

V této části se provádí příprava dat pro jejich následnou transformaci. Prvním krokem je pročištění cílové lokace pro případ, že už se zde nějaká data nacházejí, aby nedocházelo ke konfliktům. Následně se data kopírují, nejčastěji z Git repozitářů, do této cílové lokace. Při tomto procesu se data konvertují do takové podoby, aby se se všemi dalo pracovat stejně. Tyto rozdíly mohou vznikat například změnou v metadatech, kdy se jeden element začne

popisovat novým způsobem z důvodu rozšíření elementu o další informace nebo lepšího návrhu. Děje se tomu proto, aby se v generátoru nemusely podporovat oba způsoby zpracování dat.

Fáze 2

Jakmile jsou data zpracovatelná stejným procesem, začíná práce s nimi. Generátor prochází metadata a hledá nově odkazované soubory, které je potřeba ještě nakopírovat do cílové lokace, aby při dalších procesech nedocházelo k chybám spojeným s neexistujícími soubory. Následuje složité řešení a vyřešení závislostí mezi softwarovými komponentami, jež mohou být závislé na jednotlivých hardwarových platformách, mezi sebou, ale také na příkladových projektech, které jsou z tohoto pohledu brány za softwarové komponenty. Pokud nějaké z těchto závislostí nejsou splněny, není možné pokračovat v tvorbě supersetu, jelikož by celý projekt vytvořený s touto chybou byl nefunkční a distribuce takových dat je nežádoucí. Součástí této fáze je také unifikace dat pro jednotlivé platformy. Ta probíhá spolu s řešením závislostí mezi komponentami, kdy se popisy těchto komponent kopírují do specifických souborů.

Fáze 3

Po vyřešení závislostí komponent a částečné validitě dat nastává chvíle vytvořit manifesty, tedy popisující soubory pro MCUXpresso IDE. Při jejich vytváření je použita většina z metadat, a proto s tímto procesem probíhá i validace, po níž už se v metadatech i datech může nacházet pouze minimum chyb. Vytvářením manifestů začíná jakési první generování SDK balíků, protože tyto manifesty se generují pro každou hardwarovou platformu, tedy pro každý SDK balík. Toto generování se provádí dopředu, protože validuje data do hloubky, protože je zdlouhavé a protože vygenerované soubory jsou stálé, tedy jejich obsah se nemění v závislosti na obsahu jednotlivých SDK balíku, ke kterým jsou přiloženy, tudíž není nutné prodlužovat generování balíku touto akcí. Jakmile jsou manifesty vytvořené, nastává chvíle pro vytvoření souboru, který je stěžejní pro tuto práci. Jedná se o soubor s pracovním názvem '`webdata.xml`'. Jak je možné zjistit už z názvu, jedná se o soubor, který popisuje tyto data pro webovou aplikaci. Vznik tohoto souboru je podmíněn vytvářením supersetu, protože pokud je vytvářen jen SDK balík, není tohoto popisu zapotřebí. Detailní popis vzniku tohoto souboru je uveden v kapitole 4.

Fáze 4

Tato část generátoru, jak již bylo zmíněno, slouží pouze pro poslední úpravy a zabalení souboru do výsledného SDK balíku. V rámci posledních úprav se provádí konverze konců řádků z linuxových na takové, které podporuje operační systém, pro který je balík vytvářen. Pokud by byla tato část vynechána, mohlo by dojít k dysfunkci mnoha souborů a celý SDK balík by byl tímto nefunkční jen z toho důvodu, že cílový operační systém neví, jak s takovými soubory zacházet. Po této konverzi jsou soubory rozděleny dvě části, a to zdrojové kódy a dokumentaci. Tyto dvě části jsou zkomprimovány a zabaleny buď ve formátu `.zip` nebo `.tgz`, v závislosti na platformě. Tímto vzniknou dva soubory, kde první z nich obsahuje plnou dokumentaci k SDK balíku a jeho obsahu a druhý převážně zdrojové kódy. Tyto dva balíky už jsou ve finálním stavu, ve kterém se doručují zákazníkovi.

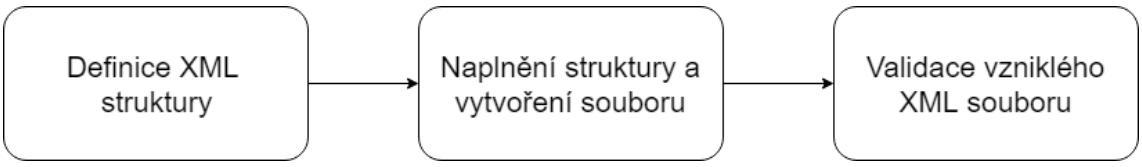
Kapitola 4

Modul Webdata

Tento modul, je součástí třetí fáze generátoru a vytváří XML soubor s daty, která jsou nezbytná pro nasazení supersetu na web. Formát XML byl zvolen primárně z důvodu, že lze validovat proti XML schematu, tedy není potřeba vytvářet testy. V tomto případě nevadí, že je tento formát méně čitelný, protože soubor slouží pouze ke strojovému zpracování.

Hlavní funkcionalitou tohoto procesu je seskupit data, která se nacházejí na více místech, případně detekovat a nahlásit jejich absenci, poté na ně aplikovat základní validace, které je možné užít v této fázi a následně je uložit do XML souboru. Konfigurační XML soubor pro web je dále nutné validovat oproti XML schematu, čímž se kontrolují například datové typy, rozsah hodnot některých dat či jejich délka, aby tato data bylo možné do databáze vůbec vložit.

Proces, při kterém je vytvořen tento soubor, je zachycen na obrázku 4.1. První část tohoto procesu, tedy definice struktury, se provádí pouze při návrhu celého modulu, při pozdějším užití jsou do této struktury zaznamenávány pouze změny v obsahu, který má být dodatečně zobrazen na webu, ovšem tato editace není tak častá. Struktura je mimo tato nová rozšíření dále neměnná. Druhá část procesu zajišťuje prezenci dat, která jsou následně vložena do struktury, ze které je možné vytvořit požadovaný XML soubor. Na tento soubor je následně aplikována validace v podobně XML schematu.



Obrázek 4.1: Jednotlivé fáze vzniku popisujícího souboru

Základním kamenem této práce je právě struktura, protože popisuje veškerá data supersetu, která jsou nezbytná pro fungování samotného webu, a proto nesmí být opomenuta žádná část jejího obsahu. Mezi tyto data patří informace o jednotlivých platformách, například frekvence a paměti procesorů, závislosti na jiných platformách, tedy že procesor vždy vyžaduje desku, informace které komponenty jsou pro danou platformu dostupné, ale také závislosti mezi komponentami, tedy které komponenty jsou vyžadovány k funkčnosti jiné. Součástí těchto informací jsou také i cesty k souborům, se kterými web pracuje. Jsou jimi například obrázky platforem, ale také licence a dokumentace. Soubor, který bude za pomocí této struktury vytvořen s každým supersetem, slouží jako brána mezi SDK generátorem a distribučním webem, kde právě tento web aktualizuje obsah své databáze

na základě obsahu tohoto souboru. Tento obsah, který je nezbytný pro proces nasazení a správné fungování webu, byl získán analýzou struktury databáze a také analýzou starého procesu nasazení.

4.1 Implementace struktury pro vytvoření XML souboru

Pro vytváření XML souborů v Ruby sice jsou dostupné gemy[1], ale naneštěstí pro tyto účely je většina z nich nevyhovující. Z tohoto důvodu byl pro řešení tohoto úkolu použit gem `nokogiri-happymapper` [2], který umožňuje definovat strukturu XML za pomocí tříd se speciálními atributy. Toto řešení obsahuje sice o poznání více kódu, než použití některé z již vytvořených knihoven, protože je nutné definovat samostatnou třídu pro každý XML element obsahující jiné elementy a atributy, ale zato dává programátorovi svobodu tvorby a možnost detailní specifikace, například datových typů, výsledné XML struktury. Definice části této struktury, která je použita ve finálním řešení, je zobrazena v příkladu 4.2. Při naplnění právě této struktury daty vzniká XML obsah, který může vypadat například jako v příkladu 4.1.

Celý kód příkladu 4.2 byl zabalen do modulu a to především z toho důvodu, že se jedná o rozsáhlý projekt, ve kterém se nacházejí třídy se stejnými názvy. Díky těmto modulům rozpozná vývojové prostředí správnou třídu a validuje správně syntaxi dílčích metod bez nutnosti spouštět celý program. Uvnitř modulu byly definovány třídy, kde každá třída z pravidla odpovídá jednomu výslednému typu elementu, který se bude nacházet ve výsledném souboru. V každé z těchto tříd byly následně definovány atributy za pomocí klíčového slova `attribute` a elementy, které jsou reprezentovány klíčovými slovy `element` (definuje jednoduchý element s různými zanořenými elementy) nebo `has_many` (definuje element, jehož obsah je tvořen elementy stejného typu/názvu, tedy polem). Některé z těchto prvků byly dále rozšířeny o parametr `tag`, jehož užití lez vidět například ve třídě `Device`. Tento parametr definuje název elementu ve výsledném XML souboru. Parametr je nutné používat v případech, kdy definice obsahuje například podtržítko, které tato knihovna defaultně ignoruje.

Po definici základních prvků každého elementu, byly vytvořeny metody pro uložení hodnot do těchto prvků. Aktualizace hodnoty atributu, respektive elementu je realizována přiřazením hodnoty, respektive inicializované třídy a v případě elementu `has_many` vložením inicializované třídy do pole. Za účelem zjednodušení vkládání specifických hodnot do zanořených jednoduchých elementů, například s jedním atributem, byly vytvořeny metody, jež inicializují třídu konkrétní hodnotou, například ve třídě `EvalBoards` a její metodě `add_board`.

Neinicializované elementy a atributy se ve výsledném XML souboru neobjeví. Díky této vlastnosti nebylo nutné definovat speciální třídu pro každý element, který není komplikovaný, tedy obsahuje třeba jen několik málo atributů. U komplikovaných elementů tato nutnost zůstává z důvodu přehlednosti. Pro takovéto elementy byla vytvořena jedna speciální třída, která dovoluje vytvořit element jen s definovanými názvy atributů, kde konstruktor očekává proměnný počet parametrů ve tvaru klíče a hodnoty. Klíč musí být v rozsahu povolených názvů atributů a jeho hodnotou je následně inicializován konkrétní atribut. Toto řešení výrazně zkracuje délku kódu a zvyšuje jeho přehlednost, protože se v něm nevyskytují duplicitní třídy, které se liší jen v názvu třídy samotné a definovaných atributech.

[1] Gem – pojmenování pro knihovnu v Ruby, kterou je možné doinstalovat a rozšířit tím funkčnost daného jazyka.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<device name="MK64F12" frequency="120">
  <evaluation_boards>
    <board name="HEXIWEAR"/>
    <board name="FRDM-K64F"/>
  </evaluation_boards>
</device>
```

Algoritmus 4.1: Výsledek při naplnění předchozí struktury

```ruby
module Webdata
  class Board
    attribute :name, String

    def initialize(name)
      @name = name
    end
  end

  class EvalBoards
    has_many :boards, Webdata::Board, tag: 'board'

    def initialize
      @boards = []
    end

    def add_board(name)
      @boards.push Webdata::Board.new(name)
    end
  end

  class Device
    attribute :name, String
    attribute :frequency, Integer
    element :eval_boards, Webdata::EvalBoards, tag: 'evaluation_boards'

    def initialize(name, frequency)
      @name = name
      @frequency = frequency

      @eval_boards = Webdata::EvalBoards.new
    end

    def add_eval_board(board_name)
      @eval_boards.add_board name
    end
  end
end
```

Algoritmus 4.2: Příklad definice struktury XML v Ruby

4.2 Vytvoření konfiguračního souboru pro web

Jakmile je definována struktura XML, je možné začít, skrze metody jednotlivých tříd, vkládat do této struktury data. Tyto data jsou načítána z YAMLů a samotné načítání je realizováno pomocí dvou funkcí `fetch` a `dig`. Funkce `fetch` reprezentuje načtení hodnoty, na kterou je nahlíženo jako na povinnou, tedy pokud není obsažena v datech, celý proces je ukončen chybou. Funkce `dig` načítá hodnoty, na které je nahlíženo jako na nepovinné. Obě z těchto funkcí tímto provádějí základní validaci prezence hodnoty a validují datový typ hodnoty. Dále se také jedná o jediný možný a stanovený způsob, jak načítat hodnoty z dat, aby byla zajištěna důkladnost samotné validace.

Naplnění samotné struktury bylo po jejím správném sestavení velmi jednoduché. Správným sestavením je myšleno například propagování funkce `add_board` do třídy `Device` a její metody `add_evaluation_board` v příkladu 4.2. S touto zkratkou lze přidávat evaluační desku přímo k zařízení bez nutnosti přístupu k třídním proměnným. Vytvoření a naplnění struktury 4.1 je zachyceno v příkladu 4.3.

Překážkou při tomto plnění bylo to, že data jsou rozsáhlá a rozdělená do více souborů, což značně ztěžovalo vyjádření některých vazeb. Jednou z takových vazeb je závislost příkladových projektů na komponentách. V tomto případě bylo nutné hledat pouze komponenty, které jsou specifické pro web, jenže příkladové projekty na těchto komponentách závislé nejsou. Pro najití těchto komponent bylo nutné projít celý závislostní strom, ošetřit kruhové závislosti a najít jen požadované komponenty. Tento úkol je poměrně jednoduchý, avšak komplikované na něm je to, že k projití těchto závislostí je nutné používat některé z již vytvořených funkcí na načítání dat, aby byla omezena duplicita v kódu. Tyto funkce jsou převzaty z procesu generování manifestu, avšak jsou navrženy právě pro tento proces, tedy nemusí fungovat dokonale pro jiný. Výsledkem při použití právě jedné z těchto funkcí byl nízký počet hledaných komponent, což obnášelo dlouhé hledání chyby s její nápravou.

Celkově je řešení popisu komponent nevhodné právě pro tento modul, jelikož závislosti potřebné pro web nejsou definované přímo v elementu komponent, nýbrž jsou zanořené. Tento popis je ovšem nezbytný pro definici závislostí přes celou sadu komponent pro jeden procesor, tedy je nutné jej nějak překonat. K najití potřebných závislostí bylo nutné také projít celý závislostní strom, stejně jako tomu bylo v předchozím případě.

```
def create_device
  device = Webdata::Device.new('MK64F12', 120)
  device.add_evaluation_board 'HEXIWEAR'
  device.add_evaluation_board 'FRDM-K64F'
end
```

Algoritmus 4.3: Příklad plnění struktury definované v bodě 4.2

Aby byl kód jednoduchý a byly zajištěny základní pravidla, byl pro tento projekt použit analyzátor kódu RuboCop [10], který nutí vývojáře dodržovat stanovená pravidla. Jejich dodržování je důležité, protože při vytváření projektu byla tato pravidla stanovena a bylo domluveno, že projekt nebude obsahovat žádné části, které by je porušovaly. Tato pravidla přináší mnohé komplikace, protože omezují například délku názvu funkce, tudíž není možné některé z nich pojmenovat naprosto jednoznačně. Dalším z pravidel je právě omezení složitosti kódu či maximální délky funkce. Kombinace těchto dvou pravidel neumožňuje řešit výše zmíněné problémy s komponentami jiným způsobem, než rekurzivně, což může značně ztížit hledání chyb v kódu. Samotné pravidlo složitosti funkce často neumožňovalo

jiný postup, než funkci rozdělit do více jednodušších funkcí, i přes to, že bylo na první pohled zřejmé, co daná funkce dělá. Je tomu tak z toho důvodu, že každý použitý cyklus či bezpečný operátor, které se v Ruby hojně používají, přidávají funkci na složitosti.

4.3 Validace

Po vytvoření XML souboru, který obsahuje validní a ideálně neprázdné elementy naplněné částmi metadat je nutné jej validovat znova. Tato validace probíhá za účely odhalením dalších chyb, které bylo zbytečné nebo neefektivní kontrolovat v kódu (například odkazování se na neexistující element), ale také délky hodnot a jejich datové typy, aby tyto hodnoty bylo možné vložit do databáze bez nutnosti její modifikace. Za tímto účelem bylo vytvořeno XML schema [3]. Toto schema umožňuje kontrolovat, zda jsou přítomny vyžadované elementy, zda jsou správně zanořeny či zda nejsou přítomny jiné, které v konfiguračním souboru vůbec být nemají. Tyto validace lze aplikovat i na atributy jednotlivých elementů. U atributů je dále možné kontrolovat datový typ, množinu hodnot, které může nabývat, jeho délku nebo třeba zda odpovídá regulárnímu výrazu. Tato validace je přítomna jednak z důvodu správnosti hodnot, ale hlavně z důvodu, aby výsledný XML popis dodržoval stanovenou strukturu, pro kterou jsou vytvořeny skripty realizující proces nasazení.

Část XML schema, použitého ve finálním řešení, které definuje možný vzhled příkladu 4.1, je zobrazena v příkladu 4.4. Při vytváření tohoto schema byl kladen důraz na minimální duplicitu, proto jsou ve finálním řešení často použity elementy `complexType` a `simpleType`. Tyto elementy umožňují definovat datový typ, na který se vytvořený element nebo atribut následně odkazuje. Typ `simpleType` byl použit pro definici obsahu atributů, kde omezuje rozsah hodnot, kterých může daný atribut nabývat. Typ `complexType` byl využit pro definici obsahu elementů, kde jasně specifikuje, jaké elementy a atributy může tento element obsahovat. U obsažených atributů lze tímto type dále definovat, zda jsou volitelné či vyžadované a u elementů jejich počet a pořadí. Použití těchto typů značně redukuje duplicitu v kódu, protože definice jednotlivých atributů a elementů se odkazují právě na tyto vytvořené typy. V tomto řešení byla dále využita i možnost dědičnosti, protože například hardwarové platformy obsahují řadu shodných atributů. Tyto atributy byly definovány jedním typem, který byl použit jako rodič dalšího typu, jež definoval další atributy specifické pro konkrétní platformu.

Samotná struktura pro tvorbu XML byla vytvářena spíše paralelně s tímto schematem, některé části dokonce až po jeho dokončení. Je tomu tak z toho důvodu, protože obecně platí pravidlo, že každá definice elementu v tomto schematu odpovídá jedné třídě v kódu, tedy nejprve byl specifikován formát XML a až následně byl přenesen do kódu, který soubor splňující tento formát vytváří. Vzniklé schema se strukturou, která jej reflektuje v kódu, vytvořilo již zmíněnou pomyslnou bránu mezi webem a SDK generátorem. Skrze tuto bránu neprojdou žádná nevalidní data, proto pokud je při plnění struktury odhalena chyba, celý proces je nekompromisně ukončen a pokud je vytvořen výstupní soubor, který nesplňuje strukturu definovanou XML schematem, je označen jako invalidní.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="PlatformName">
    <xs:restriction base="xs:string">
      <xs:maxLength value="50"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="BoardType">
    <xs:attribute name="name" type="PlatformName" use="required">
  </xs:complexType>

  <xs:element name="device">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="evaluation_boards">
          <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
              <xs:element name="board" type"BoardType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="PlatformName" use="required"/>
      <xs:attribute name="frequency" type="xs:positiveInteger"
                    use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Algoritmus 4.4: Příklad XML validačního schematu

Kapitola 5

Akceptační testy

Akceptační testy jsou druhou částí pomyslné brány mezi SDK generátorem a webem. První část této brány je popsána v kapitole 4.3. Zatímco XML schema je jakási společná část, která se musí dohadovat mezi oběma týmy tak, aby byly uspokojeny všechny požadavky, tak akceptační testy jsou v režii jen a pouze týmu, který vyvíjí webovou aplikace. Toto obnáší jednu komplikaci a to takovou, že tyto požadavky musejí být v ideálním případě jednoduché a hlavně splnitelné. Pokud by tato kritéria splněna nebyla, obnášelo by to jisté komplikace v podobě odporu k jejich užívání a možné snaze tyto testy obcházet.

Jelikož je samotný SDK generátor ještě v plenkách, není nutné s těmito testy spěchat, avšak je dobré již mít základní strukturu a pouze nahlížet na výsledky těchto testů s nadhledem. Veškeré odhalené chyby je dále nutné hlásit patřičným lidem a vyvíjet určitý tlak na vývojáře zodpovědné za odstranění těchto chyb, aby tomu tak učinili co možná nejrychleji. Jakmile budou všechny problémy odstraněny, nebo budou domluvené jisté výjimky, akceptační testy vytvoří pevnou hranici, přes kterou se nedostane žádný nevalidní release.

5.1 Realizace testů

I přes to, že data jsou důkladně validována SDK generátorem, je stále nutností tyto testy mít. Generátor tyto chyby sice zapisuje do souboru, ale pokud generování supersetu není ukončeno chybou, tak se s největší pravděpodobností do tohoto souboru s chybami nikdo nepodívá. Z tohoto důvodu byl první test navržen tak, že vyžaduje přítomnost samotných logů v supersetu. Tento test však pevně vyžaduje přítomnost těchto záznamů pouze ze třetí fáze, která je vytváří automaticky. První a druhá fáze generátoru, pokud neobsahuje žádné chyby či varování, žádný záznam negeneruje. S tímto testem úzce souvisí i druhý test, který byl navržen tak, aby kontroloval přítomnost chyb v těchto záznamech. V tuto chvíli je aktivována pouze část testu, která kontroluje záznamy ze třetí fáze, protože záznamy z první a druhé fáze nemají sjednocený formát, tudíž je není možné s jistotou strojově zpracovávat. Kontrola třetí fáze byla realizována tak, že je vyhledán sumář v tomto záznamu, který obsahuje celkový počet chyb z této fáze. Poslední z navržených testů kontroluje přítomnost konfiguračního XML souboru, bez kterého by celý tento proces nedával smysl a superset by byl naprosto zbytečný, protože by nebylo možné jej nasadit. Tyto zmíněné akceptační testy jsou realizovány pomocí modulu pytest [15].

Jednou z podmínek akceptace supersetu je také validita konfiguračního XML souboru, ale ta je již kontrolována po vytvoření tohoto souboru v SDK generátoru. K tomuto testu byl vznesen návrh, pro duplikování této validace před samotným procesem nasazení, který čeká

na schválení. Poslední podmínka akceptace je kontrola závislosti mezi deskou a procesorem, kdy je nutné, aby každá deska, která se již nachází v databázi, měla nastaven jeden a ten samý procesor jako defalutní. Tato vazba je pevná, tedy jakmile je jednou uložena do databáze, je nutné ji respektovat. Tento test bohužel není možné realizovat pomocí samostatného testovacího skriptu, protože není známo, pro kterou z instancí je konkrétní superset určen a dále není zaručeno internetové připojení a přístup k těmto datům pro kohokoliv, kdo má přístup k tomuto testu. Data, kterých se toto testování týká, jsou proto testována až při samotném procesu nasazení. Případné chyby jsou nahlášeny pověřeným osobám a superset s takovou chybou není akceptován.

Vytvořené testy byly vloženy do nového repozitáře, avšak zatím nejsou použity. Je tomu tak z toho důvodu, že toto řešení dosud nebylo schváleno SDK týmem. Testy jsou aktuálně vytvořené do té míry, že k jejich integraci bude potřeba minimum úprav, například nastavení správné cesty v kódu. Řešení testů, které bylo navrženo, uvažuje tento repozitář jako submodul SDK generátoru s tím, že součástí tohoto submodulu bude i XML schema. Toto XML schema bude nově verzováno, kde jeho verze bude součástí názvu, například `webdata_v1.0.xsd`. Aktuální verze vygenerovaného XML souboru bude obsažena uvnitř něj. Díky této změně bude moct být tento submodul i součástí webu a bude možné konfigurační soubor validovat podruhé, což zobrazí možné chyby lidem, jež mají na starost samotné nasazení.

Kapitola 6

Atlassian Bamboo

Atlassian Bamboo [9] je produkt firmy Atlassian [8] a slouží pro sestavování, testování, distribuci či nasazení vlastního softwaru. Služba funguje tak, že na vzdáleném serveru běží agent, který vykonává požadavky specifikované z hlavního serveru. Hlavní server má kontrolu nad všemi běžícími agenty a je z něj možné jim přiřazovat práci, pozastavovat ji, či spouštět nějakou jinou. Jednotlivé práce lze ukládat do šablon pro pozdější použití. V šabloně je mnohdy specifikován pracovní adresář, tedy místo vykonávání specifikovaného kódu a často i větev repozitáře, se kterou má kód dané práce pracovat. Tato šablona se dá dále větvit na jednotlivé úkoly, které na sebe mohou logicky navazovat, provádět se paralelně, nebo na sobě mohou být dokonce i závislé, tedy když jedna selže, další se nebude provádět. Na jednotlivé šablony lze nastavit i přístupová práva, čímž je zajištěno, že na produkčním serveru nemůže tuto šablonu spustit nikdo, kdo by to neměl dělat.

Kód jednotlivých úkolů se může různit. Bamboo podporuje několik prostředí, kde každé vyžaduje jinou syntaxi kódu. Jedním ze základních a často používaných je prostředí pro Docker [12], kde je nutné psát kód syntaxí jakou používá `Dockerfile`. Dalším používaným prostředím je Linuxový Shell. Toto prostředí dává programátorovi mnohem větší volnost, jelikož je možné využívat veškeré funkce, které podporuje příkazová řádka daného server. Je možné pomocí tohoto prostředí spouštět různé skripty, aplikace či odesílat e-maily.

Bamboo má také širokou podporu pro testování. V tomto případě je potřeba přidat do testů rozšíření, které po jejich skončení vytvoří soubor s celkovými výsledky. Bamboo realizuje spuštění testů a po jejich dokončení aktivuje prostředí, které na vstupu očekává cestu k souboru s výsledky, které následně přehledně zpracuje a zobrazí uživateli. Takto zpracované testy je možné umístit do karantény, v případě že nefungují, nebo jen najít zdroj chyby přehledně v logu. Pokud testy neprochází, je možná integrace s dalšími produkty Atlassian, například vytvořit tiket na opravení tohoto testu v tiketovacím systému Atlassian Jira.

Jelikož se jedná o produkt firmy, která vyrábí produkty pro usnadnění práce, je tato integrace na místě. Bamboo zvládá primárně komunikovat se systémem Atlassian Jira, který slouží pro vytváření takzvaných tiketů, které specifikují, co má který zaměstnanec vyvíjet či opravovat, jakou to má prioritu a do kdy má být daný tiket vyřešen. Dalším podpořenou aplikací je Bitbucket, také od Atlassianu, která přehledně zobrazuje kód v repozitářích, se všemi větvemi a změnami. Při této integraci je mnohdy v Bamboo vidět, která konkrétní osoba udělala v repozitáři změny, kvůli kterým například neprošly úspěšně testy.

6.1 Funkce Bamboo při vývoji generátoru

Bamboo slouží dodnes i při vývoji SDK generátoru. S každým vytvořeným pull requestem[1] se spouští sada regresních testů, které mají za cíl otestovat stávající a nově vytvořený kód, jestli fungují jako celek. Dále se testují jednotlivé moduly, zda jsou plně funkční. V případě že tyto testy neskončí úspěšně, na daný pull request nelze provést akce merge.

6.2 Role Bamboo při nasazení na produkční web

Nasazení sady softwarových komponent na produkční web, tedy nahrání dat o supersetu do databáze se také děje za pomoci Bamboo. Podle literatury by ten to proces měl vypadat tak, že na každém serveru se bude nacházet jeden Bamboo agent, který provede potřebné úkony. V reálném světě to takhle naneštěstí nefunguje. Nejedná se zde jen o produkční server, který se v tomto případě skládá z více než šesti fyzických strojů, ale také o testovací servery, které jsou přítomny pro testování nedokončených a tím pádem ještě nevalidních releasů. Tyto servery se také skládají z více fyzických strojů. Při dodržování jistých doporučení by bylo využívání Bamboo velice nákladné, a proto je proces optimalizován tak, aby ho zvládl vykonat jeden agent. Za těmito účely byly vytvořeny knihovny nebo podpůrné procesy, které zvládají obsloužit více strojů jak sériově, tak i paralelně. Jedny z nejznámějších jsou Ansible [19] a Fabric [14]. Obě tyto knihovny jsou speciálně vytvořeny za účelem jednoduchého použití a bezagentového nasazování. V tomto případě je z Bamboo spuštěn právě takovýto skript, který vykoná práci na všech strojích. Je nutno dodat, že nad šablonami Bamboo nejdou provádět restrikce jako v samotném repozitáři, proto tyto skripty jsou zařazeny do repozitářů a podléhají standardnímu vývojovému procesu. Bamboo při spouštění slouží spíše jako jakýsi monitor těchto služeb, tedy uchovává stav tohoto procesu, jestli uspěl či nikoliv a hlavně logy. Dále slouží k propojení s repozitářem Bitbucket a tiketovacím systémem Jira tak, že všechny záznamy o konkrétním nasazení jsou na jednom místě.

Vzhledem k tomu, že Bamboo umožňuje také ukládání jednotlivých šablon, je této výhody plně využito. Nasazení dat na více instancí tímto způsobem by obnášelo editace šablony při každém nasazení, protože musí být zajištěno, aby se data dostala na správné servery. Proto existuje šablona pro produkční instanci, testovací instanci a další.

6.2.1 Nevýhody Bamboo v tomto projektu

Bamboo samotné umožňuje třídění šablon obecně na obyčejné a nasazovací. Obyčejné šablony se mohou spouštět v závislosti samy na sobě, tedy když je jedna úspěšně ukončena, v návaznosti na ní se spustí další. Nasazovací šablony toto neumožňují, tedy když se úspěšně nahraje na server kód, který zobrazuje webovou stránku, nebo nový release, taky nelze vytvořit vazbu na obyčejnou šablonu, které realizuje tyto testy. Testy nejsou součástí nasazení z toho důvodu, že pokud nějaký selže, celý proces nasazení je vysvícen červeně, což značí selhání. Toto by mohlo být velmi zmatečné a na první pohled značit, že celý proces selhal.

[1] Pull request, neboli code review je akce, kdy jiná osoba kontroluje změny v kódu při slučování více větví

Kapitola 7

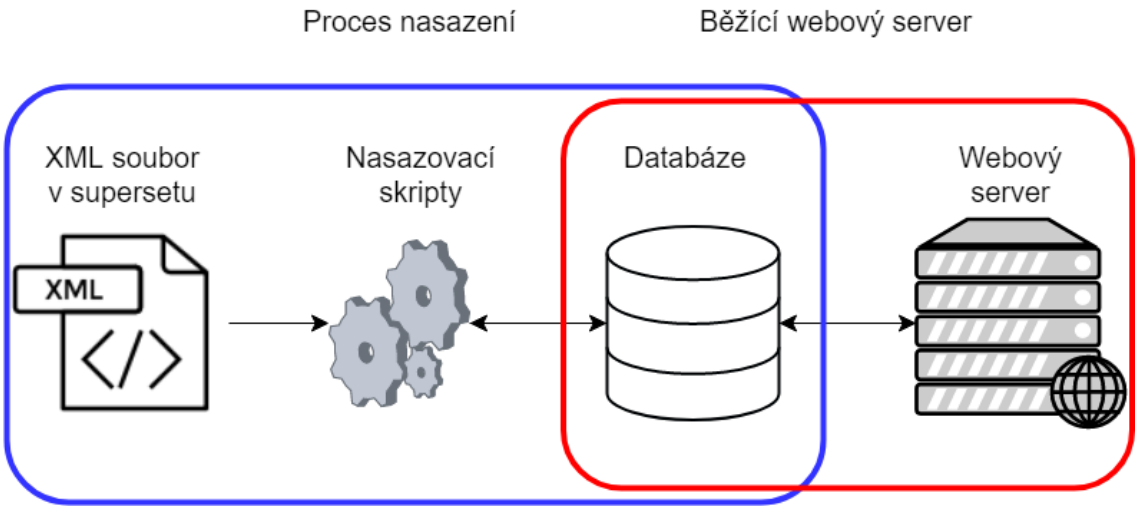
Nasazení softwarových komponent na produkční web

Aby web samotný mohl správně fungovat, je potřeba mít korektně sestavenou a naplněnou databázi. Tato databáze je důkladně navržena při vzniku, ale i přes to je nezbytné občas provádět změny, aby bylo možné uložit nová data či vazby mezi nimi. Jelikož webová aplikace má přístup pouze k této databázi, musí být všechna důležitá data zde, protože susperset samotný, či jiný popis je pro tuto aplikaci nesrozumitelný a není efektivní jej popisovat jinak, než právě takto.

I přes to, že na vstupu této části procesu by už měla být jen a pouze validní data, je zde stále potřeba postupovat bezpečně a data nadále kontrolovat. Validaci je nutné provádět hlavně z důvodu testování nasazení dat, která mohou být v této fázi ještě nekompletní či některé části mohou zcela scházet, ale také z důvodu otestování elementů, které není možné otestovat při vytváření konfiguračního souboru nebo následnou validací XML schematem. Samotná validace XML schematem většinu chyb odhalí, avšak proces pouze označí konfigurační soubor jako invalidní a chyby se zaznamenají do logu, který v této fázi vývoje zatím není zpracováván strojově. Důvody proč tomu tak je, byly zmíněny v kapitole 5. Pokud je konfigurační soubor nevalidní, celý proces se sice provede, z důvodu aby zobrazil další chyby, ale změny se nereflektují do databáze.

Je nutné brát v potaz, že kvůli bezpečnosti a minimalizaci chyb je vývoj tohoto procesu rozdělen na instance, kde první je testovací, následována před-produkční a na konci je samotná produkční instance. Díky tomuto procesu je nutné mít kontrolu nad verzí samotné části, která obstarává nasazení. Není možné nasadit data vygenerovaná novým SDK generátorem na produkční instanci, pokud ona samotná nedisponuje novou verzí procesu nasazení. Toto rozdělení vývoje není kvůli nasazení, ale kvůli webové aplikaci samotné, kterou je nutné ladit více, než její ostatní části, jako je například tato.

Samotný proces nasazení musí také fungovat bezpečně a nesmí měnit obsah databáze do té doby, dokud nebude jisté, že je tato změna bezpečná, protože jak lze vidět z obrázku 7.1, jak proces nasazení, tak webový server potřebují ke své funkčnosti společnou databázi a oba procesy z ní chtějí číst a zároveň do ní zapisovat současně. Pokud by k takové situaci došlo, mohly by být uživatelům zobrazeny v lepším případě špatná data, ale v těch horších dokonce i chyba, protože data nemusí mít splněny všechny vazby na ostatní data, které web vyžaduje. Této situaci částečně zabraňuje uchovávání dat pro web v mezipamětích, avšak v těchto mezipamětích nejsou uložená všechna data.



Obrázek 7.1: Zdroje procesu nasazení oproti zdrojům samotného webového serveru

7.1 Starý proces nasazování

Starý proces nasazení, tedy aktualizace a rozšíření obsahu databáze, je prováděn již zmíněným druhým SDK generátorem, pro který jsou speciálně vytvořená metadata. Tento proces se prováděl s již vygenerovaným supersetem, tedy s daty, která by měla být kompletní, validní a měla by obsahovat informace pro nasazení na web. Bohužel součástí tohoto procesu je také generování manifestů, které vyžaduje další specifická data, stejně jako nasazení na web. První generátor nevaliduje tyto data jako celek, který je nezbytný pro tento proces, tudíž pokud je zapomene pověřená osoba aktualizovat, data pro tento proces chybí a superset nemůže být nasazen na distribuční web. Těmhle chybám sice zabraňují akceptační testy, ale ty nejsou vždy aktuální vůči změnám jak na webu, tak v manifestu.

Za použití tohoto řešení je nezbytné načítat do paměti množství, s tímto procesem naprosto nesouvisejících informací, které jsou důležité pro generování manifestů, ale né pro aktualizaci obsahu databáze. Tento proces dále samotnou aktualizaci obsahu databáze neřeší za pomocí sezení, jak je doporučeno a dokonce i zvykem, ale ukládá tyto hodnoty přímo. V tomto případě, pokud je odhalena chyba, jak v generování manifestu, tak při aktualizaci obsahu databáze, tento proces končí chybou a obsah databáze může být již pozměněn, což způsobuje nucený výpadek pro obnovení databáze ze zálohy, což je nežádoucí.

7.2 Nový proces

Nedokonalosti popsané v předchozí kapitole byly dalším z důvodů vzniku nového SDK generátoru, který přesouvá starost o softwarová data na SDK týmy, které se tímto budou nově starat i o generování manifestů. Tento proces však nově vyžaduje vytvoření konfiguračního souboru, který bude obsahovat veškerá potřebná data pro aktualizaci obsahu databáze, aby se proces nasazení nemusel měnit s každou změnou v metadatech a nemusel s těmito daty vůbec pracovat.

Díky tomuto souboru se veškerá potřebná data nacházejí na jednom místě, tudíž je potřeba je jen validovat a navíc jim přizpůsobovat tento proces, což obnáší mnohém méně práce než předchozí řešení. Již zmíněnou pomyslnou bránou mezi SDK generátorem a webem je

právě tento soubor. Díky němu už samotný proces nasazení nepotřebuje rozumět supersetu v takovém rozsahu, nýbrž jen tomuto souboru, což velmi zjednodušuje celý tento proces i jeho správu.

Nový proces nasazení obsahuje tedy jen základní struktury pro načtení dat z konfiguračního souboru a dále funkce, které systematicky aktualizují obsah databáze. S načítáním a ukládáním těchto dat probíhá i poslední část validace, která odhaluje poslední chyby. Tento proces ale není pouze o aktualizaci dat v databázi, je zapotřebí také provést určitě operace na disku, umístit superset fyzicky na distribuční server a jiné přípravy a procesy po ukončení aktualizace databáze, aby byl celý web funkční a data byla dostupná. Tyto procesy jsou popsány v následujících kapitolách.

7.2.1 Přípravný proces

Z důvodu velikostí jednotlivých supersetů, které dosahují i řádů gigabytů, není možné jich uchovávat více než několik desítek na jednom serveru. Z tohoto důvodu jsou právě tyto supersety komprimovány do formátu SquashFS [6], čímž je jejich velikost mnohonásobně menší. Komprimace probíhá v prvním kroku, ještě před nahráním dat na server. Po vytvoření tohoto formátu, se superset nahraje na distribuční server do požadované lokace. Tento komprimovaný formát dále stačí jen přimontovat k operačnímu systému tak, aby z nich uměl operační systém, potažmo webová aplikaci číst. Po tomto procesu je dále nutné vytvořit adresáře na disku, do který se budou ukládat soubory spojené s jednotlivými balíčky a konkrétním supersetem, pro jejich logické třídění. Všechny tyto operace vyžadují nastavení potřebných práv na vytvořených souborech a adresářích.

7.2.2 Proces nasazení

Samotný proces nasazení byl vytvořen ve stejném jazyce, jako webová aplikace, tedy v Pythonu [16] na základě analýzy starého procesu. Jeho vstupem je takzvaný „tag“, který jednoznačně odlišuje supersety, slouží pro orientaci mezi nimi a dává procesu informaci, jaký soubor v požadovaném formátu hledat. Při spuštění tohoto procesu jsou provedeny základní validace, které nezajišťuje XML schema, například odkazy na názvy desek, jež jsou v některých elementech zprostředkovány pomocí názvu desky, nikoliv jejím identifikátorem. Je tomu tak z důvodu zjednodušení generování právě tohoto souboru, protože metadata se v určitých částech odkazují názvem (nejčastěji části pro rozhraní na webu), ale v jiných pomoci identifikátorů. Po provedení této validace následuje část, která systematicky aktualizuje obsah databáze. Při této aktualizaci probíhají validace s již existujícími daty. Jednou z těchto validací je závislost desky na procesoru, která je více popsána v kapitole 5.1. Součástí tohoto procesu bylo implementováno i kopírování souborů na disk a to z toho důvodu, že celý superset již není v nezpracovaném formátu, ale ve formátu SquashFS. Kopírování se týká obrázků platforem, které se zobrazují na webu s každým jejich výběrem a dokumentací, na které se web hojně odkazuje a zprostředkovává přístup k jednotlivým on-line verzím. Jelikož to jsou velmi používané soubory, nebylo by logické například kvůli každému obrázku montovat superset, načíst obrázek a poté jej zase odmontovat, protože tato operace se může provádět i mnohokrát za minutu.

Při vkládání dat do databáze byl kladen důraz na pořadí nahrávání jednotlivých dat. Databáze samotná je tvořena mnoha vazbami mezi jejími tabulkami, a proto je při plnění nutné začít na úrovni jednotlivých elementů a až poté definovat vazby mezi nimi, aby nedocházelo k odkazování neexistujícího prvku.

Jak již bylo zmíněno, celý tento nově implementovaný proces disponuje parametrem, který umožňuje nahrání dat do databáze i přes to, že nejsou validní. Tento parametr byl vytvořen zejména pro první testovací nasazení, protože SDK generátor vznikal od nuly a bylo na něm mnoho práce, tedy data pro tento proces přibývala postupně a pomalu. Stejně, jako bylo mnoho práce na generátoru, bylo nutné vyvíjet i samotný proces pro nasazení, protože i ten je potřeba řádně otestovat, než se dostane do ostrého provozu. Toto testování a vývoj byly prováděny na neúplných datech, přičemž každá taková datová sada obsahovala vždy nějakou novou část, pro kterou bylo možné vyvinou další část tohoto procesu a následně jej otestovat. Tímto způsobem vznikl celý proces nasazení.

Pro načítání dat z konfiguračního souboru byly vytvořeny obdobné funkce `dig` a `fetch`, které mají shodnou funkcionalitu, jako funkce z SDK generátoru, tedy získávají povinnou nebo volitelnou hodnotu a zaznamenávají chybu, pokud tato hodnota není nalezena. Tyto funkce navíc disponují proměnným počtem parametrů, pro možnost načtení konkrétní hodnoty libovolně zanořené. Tato varianta byla zvolena z toho důvodu, protože data z XML souboru jsou načítána knihovnou `xmltodict` [11], pro zjednodušení celého procesu. Tato knihovna poskytuje dostačující řešení, tedy že obsah XML souboru načte do datového typu slovník a právě zmíněny volitelný počet parametrů z tohoto slovníku získá hodnotu, která je uložena pod posledním z klíčů, tedy parametrů. Funkce `fetch` je svázána s výpisem na standardní výstup a při nenalezení požadovaného elementu, je toto oznámeno jako chyba. S každou takovou chybou je dále upravena hodnota proměnné, která definuje validitu dat. Při zpracování celého tohoto souboru, pokud tato proměnná neobsahuje kladnou hodnotu, data se smažou z databázového sezení a celý proces je ukončen jako neúspěšný. V opačném případě jsou data uložena do databáze a stávají se viditelná pro webovou aplikaci.

7.2.3 Post proces

Po nahrání těchto dat do databáze nastává automatizovaná úprava dat, například známé limitace, či opravy dlouhodobých problémů, které ještě nebyly reflektovány do dat. Poté jsou vytvořeny a rozeslány notifikace uživatelům, které informují o možné aktualizaci stávajících balíků. Konečně s touto poslední změnou v databázi, se splachuje samotná mezipaměť a tím webová aplikace získává přístup k novým datům a s prvním dotazem je také zobrazuje.

Poslední úpravy, které není nutné provádět okamžitě a jsou prováděny ručně, se nacházení v administrativní části aplikace. Zde je potřeba správně rozdělit hardwarové platformy a komponenty do skupin, aby se data, jsou určena jen některým uživatelům, nedostala mezi všechny. Společně s tímto ručním zásahem se provádí i manuální aktivování releasu, který se tímto zobrazí externím uživatelům, pro které je určen.

7.2.4 Implementace dílčích částí

V tomto procesu bylo také nutné automatizovat řešení některých problémů. Tyto řešení jsou nad rámec samotného procesu nasazení a byla přidána z toho důvodu, aby byly usnadněny některé procesy v budoucnu a předešlo se množným problémům. V první řadě tomu tak bylo při aktualizace komponent. Některé z nově přidaných komponent se nepatrně lišily ve jméně, například rozdílnými velkými písmeny, což může být jak matoucí pro koncové uživatele, tak nežádoucí pro samotný web, který s těmito jmény pracuje a vytváří díky nim seskupení komponent, které jsou shodné, ale liší se napříč jednotlivými releasy, což pomáhá uživateli při jednom z možných filtrování. Takové komponenty bylo dále nutné detekovat a upravit jejich název tak, aby byl shodný s již nasazenými komponentami.

Další problém související s komponentami byl právě v popisu závislostí mezi nimi. Každá z komponent obsahuje hodnotu, která definuje, zda má být daná komponenta vybraná na webu, nebo ne. Komplikace nastává ve chvíli, kdy jedna z komponent má být defaultně vybraná, ale je závislá na jiné, která nemá být defaultně vybraná. V tomto případě by SDK balík nešel sestavit, protože nejsou splněny závislosti mezi komponentami. Z tohoto důvodu je nutné při aktualizace tabulky s komponentami detekovat takové komponenty, jež mají být defaultně vybrané, ale jsou závislé na komponentě, které ne. Tuto závislou komponentu je po detekci nutno nastavit také jako defaultně vybranou.

V tomto procesu bylo následně nutné vložit do databáze záznam o dokumentaci komponenty, který nebyl správně odkázaný v metadatech, tudíž nebylo možné jej vložit do konfiguračního souboru. Celý proces je vyřešen tak, že pokud je v konfiguračním souboru obsažena komponenta se specifickým jménem, je prohledán obsah adresáře s dokumentací a pokud některá z přítomných složek odpovídá výrazu složeného ze specifického textu a názvu komponenty, je prohlášena za dokumentační adresář právě k této komponentě. Dále je o tomto adresáře vytvořen záznam v databázi.

V konfiguračním souboru bylo dále nutné kontrolovat existenci obrázků v supersetu. Každý popis desky a kitu obsahuje element, který odkazuje na lokaci konkrétního obrázku v supersetu. Samotný SDK generátor však tento odkaz do dat nekontroluje a tak se může stát, že pro nasazení je doručen superset, jehož konfigurační soubor odkazuje obrázky, které se v tomto supersetu fyzicky nenacházejí. Jednotlivé týmy si následně mohou stěžovat, že se na webu k jejich platformám nenacházejí obrázky. V tomto případě je nutné takové odkazy detekovat, aby nedošlo k chybě a nahlásit problém, aby byly vykonány patřičné kroky k dodatečnému nahrání obrázku.

Společně s celý procesem nasazení bylo nutné aktualizovat veškeré funkce pro aktualizaci dat v databázi o podmínku, která umožňovala vložení nového záznamu tímto procesem do databázového sezení. Je tomu tak z důvodu zpětné kompatibility, jelikož některé z SDK týmů stále vyvíjejí data starým způsobem, jež vyžaduje nasazení starým generátorem. Pro tento proces nebylo nutné tyto funkce vytvářet znova, jelikož by se lišily pouze v minimu řádků a vedlo by to k velké duplicitě kódu.

Celý proces nasazení disponuje systémem logování, které zapisuje do souboru a na obrazovku každou změnu, kterou proces provedl, každé varování a také chyby. Tento výstup pomáhá lidem, který s tímto procesem pracují, odhadnou stav procesu a dále podniknout patřičné kroky při zahlášení chyby bez toho, aby bylo nutné tuto chybu složitě hledat.

7.3 Rozšíření

Zatím jediným myslitelným rozšířením, je nasazení na jedno kliknutí, například z administrační části webové aplikace. Tento proces by byl usnadněním pro ty, kteří provádějí právě tuto činnost, protože v tuto chvíli je nutné spustit Bamboo plán, do kterého je potřeba zadat vstupní parametry a poté ještě korektně nastavit přístupová práva na jednotlivé položky supersetu v administrační části webové aplikace. Díky této inovaci by bylo možné zadat potřebné vstupy právě do jednotného webového formuláře, spustit samotný proces a poté jen upravit přístupová práva a to vše bez nutnosti pamatovat si rozhraní Atlassian Bamboo, přepínání mezi záložkami prohlížeče, či kontroly, zde je spuštěn plán, který tento proces provede na požadovanou instanci.

Kapitola 8

Efektivita nového procesu

Celý proces, od práce s daty, až po jejich nasazení, je značně efektivnější, a to hlavně z toho důvodu, že je navržen na základě detailní specifikace, která odstraňuje chyby starého procesu. Celý tento nový proces je založen na validaci, proto občasným spuštěním procesu jako celku lze odhalit mnoho chyb. Současně podpořené parametry jak pro vygenerování invalidního supersetu, tak pro nasazení těchto dat na distribuční webový systém pomáhají testovat jednotlivé části dat bez nutnosti mít je kompletní a správně. Tato funkcionalita umožňuje jednotlivým vývojářům se specializovat na svou část, reálně si ji otestovat a poté data sloučit do jednoho funkčního celku bez toho, aniž by se blokovali mezi sebou navzájem.

Seznam všech vylepšení této práce napříč oběma procesy je následující:

1. Vylepšení SDK generátoru:
   - Zjednodušení celého procesu.
   - Data pro nasazení na jednom místě – v konfiguračním souboru.
   - Vylepšení validace dat pro nasazení.
   - Jednodušší přímý kód (modul zabývající se konkrétním problémem).
2. Vylepšení nasazení softwarových komponent na web:
   - Nový návrh pokrývající staré chyby.
   - Jednodušší správa celého procesu.
   - Násobně rychlejší.
   - Ukládání aktualizovaných dat do sezení.
   - Možnost testovacího nasazení nevalidních dat.
   - Vylepšení validace.

Vylepšení validace dat pro nasazení
Pro validaci byly vytvořeny specifické funkce, které při načítání hodnot ze sjednoceného formátu dat nepropustí žádnou chybu bez toho, aniž by jí zaznamenaly do logu. Tyto funkce jsou použity právě pro načítání hodnot pro konfigurační soubor a pokud při nenalezení požadované hodnoty je celý proces ukončen, tedy soubor se nevygeneruje. Dále je použito XML schema, které podruhé a ještě přísněji validuje vzniklý soubor a pokud je nevalidní, je tak také označen. Díky těmto validacím nejsou známé žádné problémy s nevalidními daty od použití nového generátoru v kombinaci s novými nasazovacími skripty.

Vylepšení validace procesu nasazení
V tomto procesu jsou použité obdobné funkce jako na načítání hodnot z metadat, tedy každá hodnota z konfiguračního souboru je načítána tímto způsobem a pokud není nalezena, je zaznamenána chyba a data nejsou nasazena. Od používání tohoto procesu se nedostala na produkční systémy jediná chyba, která by způsobila inkonzistenci, nebo by vyžadovala manuální zásah do databáze.

Násobně rychlejší proces nasazení
Tento bod je způsoben primárně oddělením generování manifestů od procesu nasazení kde dříve celý proces trval 41 minut, ale dnes pouhých 7. Při přičtení času generování manifestu a vytvoření konfiguračního souboru, tedy fáze, která byla vyjmuta z tohoto procesu, je výsledek 18 minut, což je stále výrazně méně než při použití starého řešení. Nutno podotknout, že tato fáze nového procesu byla spuštěna na vývojovém stroji, který nedisponuje RAMDiskem, což činí výsledek jen orientační. Dále také byla spuštěna na datech, jež jsou obsáhlejší, jelikož data pro starší release nejsou kompatibilní s novým generátorem.

Zjednodušení správy procesů
Oba procesy jsou navrženy od začátku a s důrazem na eliminaci známých chyb ve starých procesech. Díky tomuto novému návrhu jsou tyto kódy čitelnější, snazší na údržbu a také se v nich lépe hledají chyby. Díky tomu, že jsou navrženy znovu, jsou novější a využívají nových knihoven a metod, což je činní rychlejšími.

Kapitola 9

Závěr

Cílem práce bylo navržení formátu souboru, který má sloužit jako zdroj informací pro proces nasazení. Spolu s tímto formátem navrhnout proces validace, který redukuje počet chyb ve výsledném formátu na minimum. Dále bylo nutné seznámit se jak se starými SDK generátory, tak s novým a do něj následně integrovat modul, který vytváří soubor, jež splňuje strukturu navrženého formátu, obsahuje validní data a splňuje navržené validace. Po vytvoření tohoto souboru vytvořit proces, na jehož vstupu bude právě tento soubor, a který aktualizuje obsah distribuční databáze informacemi obsaženými v tomto souboru. Tento nově navržený proces musel dále splňovat veškerou funkcionalitu sterého procesu tak, aby po nasazení supersetu byl veškerý jeho obsah dostupný a web plně funkční. V poslední řadě bylo nezbytné analyzovat produkt Bamboo a vytvořit v něm plán, který provede toto nasazení na distribuční web. Po integraci celého tohoto procesu bylo potřeba definovat a vytvořit akceptační testy, přes které se nedostanou žádné nevalidní supersety.

Většina dílčích kroků zadání byla splněna. Byl navržen formát, který propojuje nově vytvořený SDK generátor s distribučním webem. Bylo navrženo XML schema, které definuje pevný obsah souboru, jež obsahuje data v navrženém formátu. Dále byl integrován modul, který vytváří tento soubor, odpovídající definovanému schematu. Navíc bylo potřeba tento modul vytvořit tak, aby byl plně zdokumentovaný a napsán co možná nejvíce čitelně, protože tato část generátoru bude předána k údržbě právě SDK týmu.

Za dobu fungování tohoto modulu, přibližně půl roku, se nedostal k nasazení žádný soubor, který by obsahoval chybu, která by způsobila inkonzistenci databáze. Jakmile byly k dispozici první verze konfiguračního souboru, byla vytvořena sada skriptů, jež tento soubor částečně validovala a jeho obsahem bezpečně aktualizovala distribuční databázi. Při dokončení celého procesu byl vytvořen skript ve aplikaci Fabric, jež dodržuje týmové postupy a tento skript je spouštěn Bamboo plánem, který provádí proces nasazení na konkrétní servery. Délka běhu tohoto skriptu je při použití na nejobjemnějších datech, tedy na kterých jsou testovány rozdíly v předchozí kapitole, přibližně sedm minut, což je téměř šestinásobné urychlení.

Jediná část zadání, která nebyla plně dokončena, jsou akceptační testy. Tato část je aktuálně plně navržená a vyvinutá, tudíž ji stačí jen integrovat, což vyžaduje minimum změň v kódu. Akceptační testy nebyly nasazeny z toho důvodu, že v generátoru se stále nacházejí chyby, které by bránily úspěšnému dokončení těchto testů (nesouvisející s touto prací) a tyto testy nebyly plně schváleny SDK týmem.

Mimo tyto testy jsou oba procesy v produkčním nasazení již od začátku tohoto roku.

Literatura

[1] Anonym: FatFS. [Online; navštíveno 6.5. 2019].
URL http://elm-chan.org/fsw/ff/00index_e.html

[2] Anonym: nokogiri-happymapper. 2018, [Online; navštíveno 6.12. 2018].
URL https://rubygems.org/gems/nokogiri-happymapper/versions/0.5.9

[3] Anonym: XML schéma. 2018, [Online; navštíveno 12.12. 2018].
URL https://www.w3schools.com/xml/schema_intro.asp

[4] Anonym: Metadata. 2019, [Online; navštíveno 12.04. 2019].
URL https://cs.wikipedia.org/wiki/Metadata

[5] Anonym: YAML. 2019, [Online; navštíveno 12.04. 2019].
URL https://cs.wikipedia.org/wiki/YAML

[6] Anonymous: SquashFS. *Linux Format*, , č. 214, 2016: s. 54–54, ISSN 1470-4234.
URL http://search.proquest.com/docview/1835599549/

[7] Arm Ltd.: CMSIS-Pack Documentation. 2019, [Online; navštíveno 10.04. 2019].
URL http://www.keil.com/pack/doc/CMSIS/Pack/html/index.html

[8] Atlassian: Atlassian. 2019, [Online; navštíveno 14.04. 2019].
URL https://cs.atlassian.com/

[9] Atlassian: Atlassian Bamboo. 2019, [Online; navštíveno 14.04. 2019].
URL https://cs.atlassian.com/software/bamboo/

[10] Batsov, B.: RuboCop. 2019, [Online; navštíveno 11.5. 2019].
URL https://rubocop.readthedocs.io/en/stable/

[11] Blech, M.: xmltodict. 2019, [Online; navštíveno 11.5. 2019].
URL https://pypi.org/project/xmltodict/

[12] Docker Inc.: Docker. 2019, [Online; navštíveno 11.5. 2019].
URL https://www.docker.com/

[13] Fitzgerald, M.: *Learning Ruby*. Sebastopol: O'Reilly, první vydání, 2007, ISBN 978-0-596-52986-4.

[14] Forcier, J.: Fabric. 2018, [Online; navštíveno 14.04. 2019].
URL http://www.fabfile.org/

[15] holger krekel: pytest. 2017, [Online; navštíveno 11.5. 2019].
URL https://docs.pytest.org/en/latest/

[16] Lutz, M.: *Programming Python*. programming/Python, Beijing: O'Reilly Media, Čtvrté vydání, 2011, ISBN 978-0-596-15810-1.

[17] NXP Semiconductors: MCUXpresso IDE. 2019, [Online; navštíveno 30.03. 2019]. URL http://www.nxp.com/mcuxpresso/ide

[18] NXP Semiconductors: MCUXpresso SDK. 2019, [Online; navštíveno 5.5. 2019]. URL https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-software-development-kit-sdk:MCUXpresso-SDK

[19] Red Hat: Ansible. 2019, [Online; navštíveno 14.04. 2019]. URL https://www.ansible.com/

[20] Services, A. W.: The FreeRTOS^{TM}Kernel. [Online; navštíveno 5.5. 2019]. URL https://www.freertos.org/