Vysoká škola ekonomická

Fakulta informatiky a statistiky

Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	*LS 2015/2016*
Autoři – jméno, příjmení , xname	*Jana Novotná - xnovj145* *Kateřina Fixová – fixk00*
Téma	*Towards an Agile Software Architecture*
Datum odevzdání	*15.5.2016*

Abstrakt

Tato práce se zabývá srovnáním tradiční a agilní softwarové architektury. V první části je vymezena softwarová architektura. Ve druhé části je přiblížen tradiční přístup k softwarové architektuře, ve třetí části je pak blíže pojednána agilní architektura, která je vysvětlena na příkladu z praxe. V závěru jsou shrnuty oba přístupy.

Klíčová slova

Softwarová architektura, Agilní softwarová architektura, Agilní metodiky, SCRUM, dokumentace agilní metodiky, vývoj software

Obsah

1 Úvod

Softwarová architektura tvoří základ systému. Definuje, z jakých prvků se skládá, jakým způsobem se chová. Protože softwarová architektura tvoří v podstatě kostru systému, je velmi citlivá na změny.

V dnešní době existují dva hlavní přístupy k architektuře, které se významně liší právě v přístupu ke změnám - na jedné straně tradiční přístup založený na rigorózních metodikách (typu vodopádu), na druhé straně jsou metodiky agilní (například SCRUM).

Cílem této práce je vytvoření rámcového přehledu zabývajícího se softwarovou architekturou. Prvním cílem je definovat softwarovou architekturu a následně ji popsat z hlediska tradičních a agilních metodik. Druhým cílem je popsat a přehledně porovnat výhody a nevýhody obou přístupů.

Po přečtení by čtenář měl mít přehled o tom co je to softwarová architektura, jak se liší přístupy k ní, a jaké jsou výhody a nevýhody obou přístupů.

2 Softwarová architektura

Architektura představuje základní kámen, z kterého je budován systém. V této práci se zaměříme nejdříve na tradiční, a poté na agilní přístup modelování, vývoje a rozvoje softwarové architektury.

2.1 Definice softwarové architektury

Prozatím bohužel neexistuje žádná oficiální definice softwarové architektury, proto si zde uvedeme několik příkladů:

„*Softwarová architektura je struktura komponent programu/systému, jejich vzájemné vazby, principy a předpisy určující jejich návrh a vývoj v průběhu času*.“ Institut softwarového inženýrství při Carnegie Mellon University v Pittsburghu

„*Softwarová architektura je soubor rozhodnutí o návrhu, která pokud jsou špatná, mohou způsobit zrušení projektu*.“ Eoin Woods

„*Softwarová architektura se nezabývá pouze strukturou a funkcemi, ale také využitím, výkonem, znovupoužitelností, technologickými omezeními a kompromisy.“* Kruchten Philippe

Ačkoliv se od sebe vzájemně liší, v některých rysech se definice softwarové architektury shodují. Vyplývá z nich, že se jedná o hrubé členění softwaru na subsystémy, komponenty a způsoby jejich vazeb měnící se v rámci času. Většina definic naznačuje, že architektura se týká struktury a chování systému, zabývá se významnými prvky systému, je ovlivněna prostředím a také všemi „stakeholdery“.

Struktura

Struktura je zásadním prvkem architektury – zabývá se přitom nejen jednotlivými prvky, ale i jejich vzájemnými vazbami. Prvek přitom bývá definován vágně, může jím být myšlený například subsystém, knihovna, proces, atd.

Chování

Kromě definování prvků struktury se architektura zabývá také interakcemi mezi jednotlivými prvky. V podstatě tedy popisuje funkcionalitu systému.

Významné prvky

Ačkoli softwarová architektura definuje strukturu a chování systému, nezabývá se definováním všech jejích detailů. Zabývá se pouze prvky, které jsou nějakým způsobem podstatné. Díky tomu poskytuje nadhled na celý systém jako celek.

Kompromis mezi potřebami zainteresovaných stranVšechny osoby zainteresované na projektu mají rozdílné priority (například zákazník je zaměřen na cenu, stabilitu, určení rozvrhu dokončení, vývojář potřebuje hlavně přesně definované požadavky a návrh a čas na vývoj, marketingové uvedení chce dostat systém co nejrychleji na trh a podobně). Tyto priority mohou být často protichůdné. Softwarová architektura tvoří platformu pro komunikaci těchto potřeb a hledání kompromisu.

Okolní prostředí

Kromě zainsteresovaných stran na architekturu systému působí i okolní prostředí. Vliv může mít například první systém, technická omezení, business cíle firmy a podobně.

Kvalitní návrh architektury s formálními pravidly a notací umožňuje hladký přechod od analýzy k implementaci. Se softwarovou architekturou je také nepopiratelně spojen proces řízení celého životního cyklu softwaru. Pro tento účel byly vytvořeny přizpůsobitelné metodiky pro různé týmy a konkrétní potřeby, neboť pokusy používat „jednu šablonu na všechno“ často selhávají. [2][3]

2.2 Typy softwarové architektury

V současné době je možné popsat dva hlavní proudy metodik zabývající se vývojem softwaru, jeho architekturou a celého jeho životního procesu. Jedná se o takzvané rigorózní a agilní metodiky.

3 Tradiční architektura

Tradiční softwarová architektura vychází z rigorózních metodik. Rigorózní metodiky jsou založené na přesvědčení, že všechny procesy je možné popsat, naplánovat, a následně řídit a měřit. Z toho důvodu se snaží detailně popsat všechny činnosti a výstupy. Výsledkem bývají velmi objemné dokumenty, které jsou časově náročné nejen pro napsání, ale i přečtení a pochopení.

3.1 Historie

Metodiky řízení vývoje softwaru se začaly prosazovat přibližně v 80. letech minulého století. Jejich cílem bylo stanovení postupů a pravidel, které by snížily výskyt problémů v průběhu vývoje a tím pomohly k větší úspěšnosti projektů.

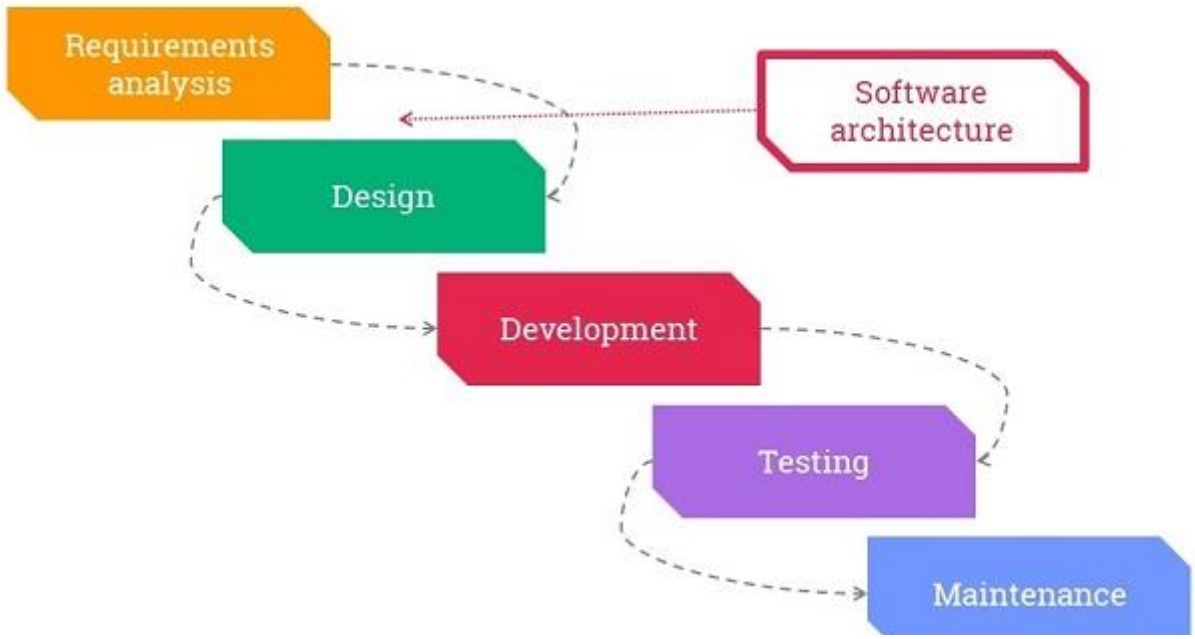
Použití vhodné metodiky pro vývoj software výrazně přispívá k efektivnějšímu průběhu celého vývojového procesu.

3.2 Popis

Rigorózní, neboli tzv. těžké metodiky, se zabývají podrobným popisem celého procesu vývoje software. Základním rysem je propracovaný popis řízení a plánování vývoje softwaru. Dle tohoto pojetí je vývoj software jasně definovaný proces, od kterého není možné se odchýlit. Jsou přesně definovány činnosti, které musí být uskutečněny, a také návaznosti ve kterých musí proběhnout.

Každý projekt má tři základní konstanty, nad kterými je nutné vyvažovat – čas, náklady a rozsah projektu. Některé jsou fixní, jiné se mnou v průběhu projektu měnit. V případě rigorózních metodik je fixní veličinou rozsah projektu, který je specifikován v úvodní části projektu. Dle něj se odhaduje čas a náklady potřebné na realizaci. Ačkoliv to umožňuje dobré řízení projektu, vyplývají z toho i zřejmá negativa, jako například nemožnost reakce na změny, velký objem dokumentace aj.[4]

Rigorózní metodiky jsou obvykle založeny na vodopádovém přístupu, kterému se budeme věnovat v další části. Dále existují rigorózní metodiky, které jsou založené na iterativním nebo inkrementálním vývoji – například Rational Unified Process (RUP), Enterprise Unified Process (EUP) a jiné.[3]



Obrázek 1 - Tradiční vodopádový model [1]

Vodopádový přístup je charakteristický tím, že má několik fází, které mají přesně daný začátek i konec. U každé z fází je také přesně určeno, jaké aktivity musí ve které fázi proběhnout. Fáze na sebe navazují, přičemž každá fáze je přímo závislá na výstupech fáze předcházející.

Na obrázku vidíme vodopád, který obsahuje 5 fází:

- Analýzu požadavků
- Návrh
- Vývoj
- Testování
- Údržba

Práce na softwarové architektuře začíná ve chvíli, kdy je dokončena analýza požadavků. [1]

3.3 Tradiční softwarový architekt

Softwarová architektura je tradičně vytvářena softwarovým architektem. Typicky je to vývojář, který byl do této role povýšen poté, co získal dostatečné technické znalosti a zkušenosti. Je to ssoba s hlubokými technickými znalostmi a zkušenostmi.

Je zodpovědný za

- Analýzu softwarových požadavků
- Provádění technických rozhodnutí (založených na požadavcích)
- Vytvoření návrhu architektury

V závislosti na rozsahu projektu může být architekt pouze jeden, případně může vzniknout celý tým softwarových architektů, kteří na projektech spolupracují. To se obvykle stává pouze u velmi rozsáhlých projektů, které mají dlouhodobé trvání (v rámci několika let). Můžeme se setkat i s opačnou situací, kdy ve firmě není oddělená role architekta. Pak je tato zodpovědnost delegována na seniorní vývojáře.

Tradiční softwarový architekt by měl mít tyto dovednosti:

- Soustředí se na „big picture“

  Softwarový architekt musí přemýšlet o tom, jak bude vyvíjený systém vypadat v budoucnosti, jaké komponenty by do budoucna mohlo být vhodné přidat a jakým způsobem. Musí brát v úvahu další systémy, které by se mohly napojit, a jakým způsobem spolu budou komunikovat.

- Vytváří neměnnou vizi

  Vycházíme z popisu rigorózních metodik – výstupem od softwarového architekta jsou dokumenty, které popisují architekturu systému. Ve chvíli, kdy architekt tyto dokumenty předá do další fáze projektu, není již možné na nich dělat jakékoli změny.

- Pracuje na základě teorie a ne praxe

  Softwarový architekt vytváří architekturu izolovaně od vývojového týmu, a do vývoje se přímo nezapojuje. To může vést ke vzniku tzv. „ivory tower“, architektury které je dokonale promyšlená a popsaná, ale v praxi není příliš použitelná. Za prvé proto, že

developeři mnohokrát nechtějí přijmout architekturu, která byla vytvořena bez přispění jejich znalostí a technologických zkušeností. Za druhé, je to teoretický model, který může narazit na technická omezení, protože architekt nemá dostatek zkušeností s vývojem. Za třetí, „ivory tower“ architektura bývá nedokonalá, protože jeden člověk může stěží promyslet všechny potřeby systému. A za čtvrté, může vzniknout zbytečně rozsáhlá architektura, která bere v úvahu všechny požadavky, se kterými se kdy architekt setkal, a ne pouze aktuální projekt.

- Produkuje plány („blueprints“)

  Kolekce dokumentů a diagramů, které popisují systém, jsou základním výstupem od architekta. Jeho úkolem je popsat systém ze všech možných perspektiv a poskytnou vývojářům kompletní materiál pro vývoj systému. Je totiž možné, že architekt bude později odvolán na jiný projekt a nebude moct spolupracovat s vývojovým týmem – ten proto musí být schopný dohledat vše v dokumentech. [1]

3.4 Výhody a nevýhody

Jak vyplývá z předchozích kapitol, tradiční softwarová architektura má mnoho nevýhod. Mezi ně patří neflexibilita rigorózních metodik, která neumožňuje jakoukoli změnu architektury v průběhu projektu. Další je oddělenost role softwarového architekta, který může být z projektu odvolán po dokončení architektury, a z toho důvodu je v případě nejasností pro vývojáře těžko dosažitelný. Z toho plyne další nevýhoda, kterou je vznik rozsáhlé dokumentace. Ta je časově náročná jak na vytvoření, tak na následné používání. Kvůli oddělenosti architekta od vývojářů také hrozí vznik „ivory tower“, které mohou znemožnit úspěšné ukončení projektu. Další významnou nevýhodou je, že zákazník není zahrnut do procesu, a produkt vidí až na úplném konci projektu. [5]

Ačkoliv má tradiční softwarová architektura mnoho nevýhod, nelze říct, že by nebyla vhodná pro žádné projekty. Naopak, pro projekty, kde je zadání jasné a jasně formulovatelné a kde je jisté, že se v průběhu nebude měnit, je tradiční přístup velmi vhodný.

4 Agilní přístup

V této kapitole budou popsány důvody pro vznik agilních metodik, jejich historie a hlavní principy. Dále bude popsán agilní přístup k softwarové architektuře, a jeho výhody a nevýhody. Následovat bude praktický příklad ze zahraničí.

4.1 Historie

V druhé polovině 90. let došlo ke změně ekonomicky technologického prostředí, což způsobilo, že dříve hojně využívané rigorózní metodiky přestaly fungovat. Změny nastaly v technologických možnostech, vybavení, nových nástrojích, dostupnosti výpočetní techniky a hlavně v dynamice celého odvětví. Celkově se zvýšila konkurence výrobců softwaru a nároky na rychlost zavedení. Zákazník požadoval kvalitu, ale nově na ni odmítal dlouho čekat. Dále poměrně často se stávalo, že zákazník ani vlastně nevěděl, co systému a finálního produktu očekává a vyžaduje.

Vzhledem k dříve popsaným důvodům začaly vznikat metodiky, které umožňují rychlejší vývoj softtwaru, jeho průběžnou údržbu a reakci na měnící se podmínky v zadání. První agilní metodiky začaly vznikat nezávisle na sobě a jejich tvůrci měli jisté společné znaky. Patřili ke špičkám ve svém oboru, ale zpravidla postupně pracovali na několika neúspěšných projektech. Neúspěch vedl k špatnému pocitu z jejich práce, a proto začali uvažovat, jaké příčiny k němu vedly. Jako představitele si můžeme uvést např. Alistara Cockburna, Kenta Becka, Warda Cunninghama, Martina Fowlera, Kena Schwabera, Jeffa Sutherlanda a další.[6]

4.2 Agilní manifest

V roce 2001 se někteří z dříve zmíněných sešli v Utahu a došli k překvapivému závěru, že jejich metodiky jsou založeny na velkém množství společných principů. Vytvořili základní zastřešující pilíře, z kterých později vznikl Manifest agilního vývoje.

Mezi hlavní myšlenky patří, že dáváme přednost:

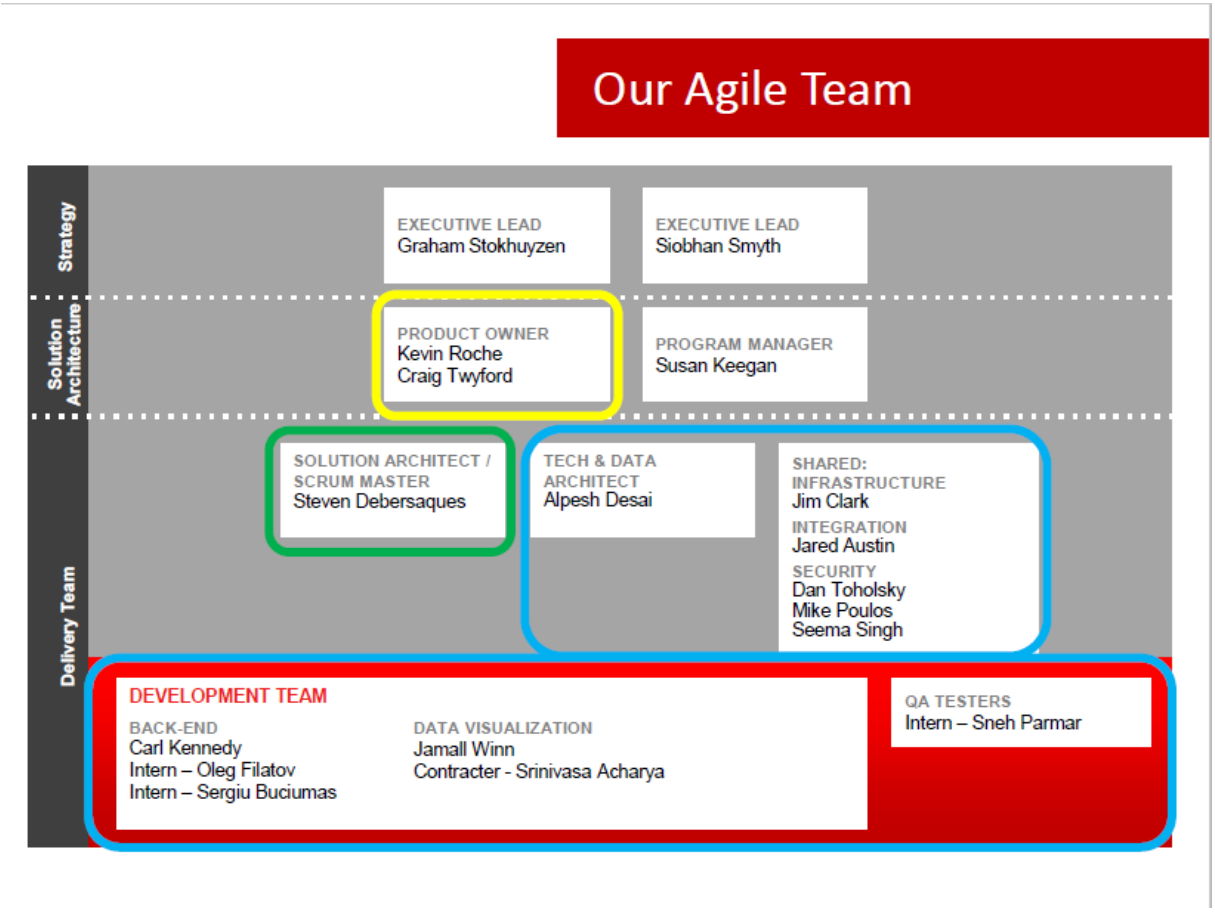
- Individualitám a komunikaci před procesy a nástroji
- Provozuschopnému softwaru před obsaženou dokumentací
- Spolupráci se zákazníkem před sjednáváním kontraktu
- Reakci na změnu před plněním plánu

4.2.1 Hlavní principy agilních metodik

1. Nejvyšší prioritou je včas a kontinuálně dodávat software, který zákazníkům přináší hodnotu. Zákazník má větší užitnou hodnotu ze softwaru, nikoliv z různých diagramů. Fungující software je nutné dodávat často (krátké iterace).
2. Změnu požadavků je možné provést i v pozdějších fázích vývoje, protože tím může zákazník získat konkurenční výhodu. Změny jsou vítané právě díky konkurenční výhodě. Nedělá se nic, co momentálně není potřebné, neboť je pravděpodobné, že se požadavky v budoucnu změní
3. Uživatelé a vývojáři spolupracují denně na projektu. Zákazníci a vývojáři nepřetržitě spolupracují, neboť není možné na začátku projektu podepsat specifikaci požadavků, která bude v průběhu celého vývoje neměnná.
4. Motivovaní jedinci, kteří mají vytvořené podmínky pro práci a mají podporu a důvěru vedení, jsou klíčovým faktorem úspěchu projektu.
5. Nejefektivnějším způsobem přenosu informací v rámci vývojového týmu je osobní komunikace. Porozumění nejlépe dosáhneme pomocí přímé komunikace
6. Primární mírou úspěchu je fungující software. Pouhé naplnění specifikace neznamená úspěch projektu, neboť mohou nastat problémy při implementaci.
7. Agilní procesy předpokládají "zdravý" vývoj. Měly bychom se zaměřit se na udržitelný vývoj projektu. Přesčasy a práce v noci řeší problémy jen krátkodobě, dlouhodobě vedou k snížení produktivity. Dále nutnost práce přesčas značí závažný problém projektu
8. Agilním vývojem vytváříme perfektní technické řešení i návrh. Změny nejsou důkazem nízké kvality, ale právě naopak. Znamenají, že v průběhu celého projektu se pracuje na návrhu, který je začleněn do každodenní náplně práce programátorů.
9. Zásadním požadavkem je jednoduchost řešení, tj. umění maximalizovat množství neudělané práce. Postupuje podle principů "Keep it simple, stupid" a "You ain´t gonna need it". Tzn. volíme nejjednodušší možné řešení a nezabýváme se vytváření práce a výstupů, která nebudou potřebné.
10. Nejlepší architektury, požadavky a návrhy vznikají ze samoorganizujících se týmů. Klademe důraz na kreativitu členů týmů. Podporujeme ji důvěrou a častou komunikací.[1][6][8]

4.3 Praktický příklad agilního týmů Coca-cola z Belgie

V průběhu minulého roku jsem zavítala na přednášku Stevena Debersaquese s názvem Agile project management at CCE. Tematicky byla zaměřena na praktické zkušenosti s agilním vývojem belgického týmu Coca-coly (viz obrázek č.2).



Obrázek 2 - Agile team, Coca-cola [7]

Využijeme tento případ z praxe jako ukázku práce směrem k agilní softwarové architektuře. Jak je možné z obrázku č. 1 vyčíst, náš tým pracuje podle agilní metodiky SCRUM. Jsou v něm definovány role scrum mastera, product ownera, a také solution architekta. Byl nám přidělen nový projekt, proto si projdeme cyklus vývoje zcela od začátku.

4.3.1 Architektura v rámci celého životního cyklu

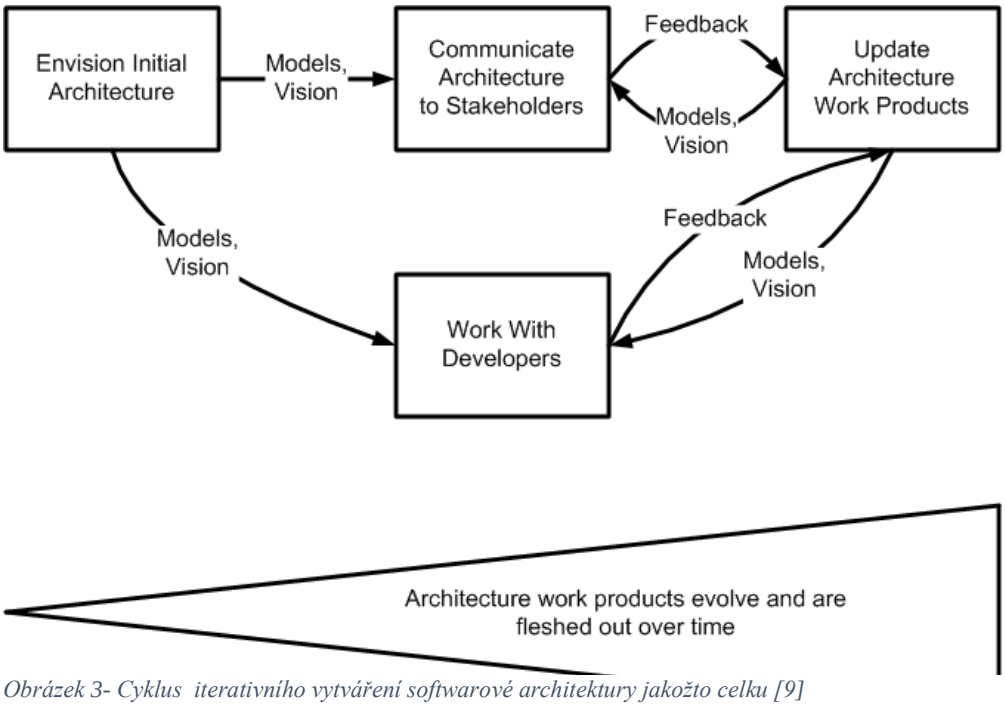
V této části je popsán vývoj softwarové architektury v rámci životního cyklu projektu v našem příkladovém belgickém týmu coca-coly.

4.3.1.1 Sprint #0

Náš agilní tým začíná projekt speciálním sprintem s označením #0. V jeho rámci se řeší základní parametry a kostra projektu, tak, aby následující sprinty byly co nejvíce efektivní tzn. nastavení prostředí, jazyk, platforma, databáze atd. Z hlediska softwarové architektury je vytvářen minimální design, který bude v průběhu následujících sprintů dotvářen a předěláván dle vývoje práce a měnících se požadavků na architekturu. Identifikujeme potencionální technologický směr a technická rizika, kterým bychom mohli čelit. Není zapotřebí řešit detailní architektonická specifika, neboť vytváření softwarové architektury v agilním prostředí je stále probíhající proces, díky čemuž je možné změnit kostru i ve velmi pozdní fázi projektu. V rámci sprintu #0 je nutné dojít k týmovému počátečnímu porozumění, díky němuž bude možné se pohybovat vpřed jakožto tým.

4.3.1.2 Klasické sprinty

Na obrázku č.3 je popsán cyklus postupného iterativního vytváření softwarové architektury jakožto celku. Jak bylo již dříve zmíněno, ve sprintu #0 jsme vytvořili minimální počáteční architekturu (Envision Intial Architecture). Následuje komunikace architektury ke všem stakeholderům a k týmů developerů. Jejich zpětná vazba je začleněna do pracovního procesu, díky čemuž dochází k úpravě vize architektury a jejího grafického vyjádření - modelů. Agilní přístup lze popsat myšlenkou, že “Implementujeme pro dnešek, navrhujeme pro úspěšnou implementaci”. To znamená, že cílem návrhu je umožnit úspěšnou implementaci modulů, které



Obrázek 3- Cyklus iterativního vytváření softwarové architektury jakožto celku [9]

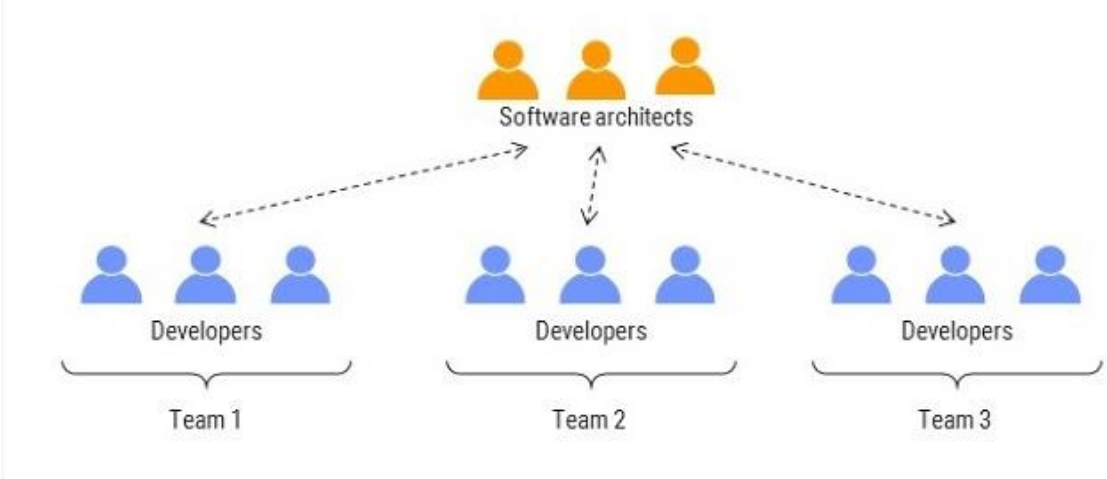
jsou v plánu v rámci aktuální iterace. Vzhledem k častým iteracím s výstupem provozuschopného výrobního produktu získáváme postupně zpětnou vazbu od všech zainteresovaných stran, díky čemuž dochází k pozvolnému vytváření konečné softwarové architektury.

4.3.2 Agilní architekt, Architect owner

Velmi záludná a problematická záležitost spočívá v otázce, kdo vlastně zastává roli architekta? Všichni členové týmu jsou si rovni, proto by logicky architektem měl být každý jednotlivý člen týmu. Vzhledem k tomu, že je architektura vytvářena společně, zvyšuje se její porozumění a akceptace. Dále jsou také developeři ochotni ji pozměnit, pokud se ukáže jako neefektivní.
V praxi ovšem není možné, aby jednotlivý členové měnili architekturu sami od sebe, neboť, by došlo k naprosté anarchii a ztrátě integrity. Navíc ne vždy se dojde k společnému porozumění členů týmu pro její změnu. Proto by každý agilní tým měl mít člena, jež zastává roli architect ownera, často též označovaného názvem agile solution architect (v rámci našeho týmu tuto roli zastává Steven, který je také SCRUM master).

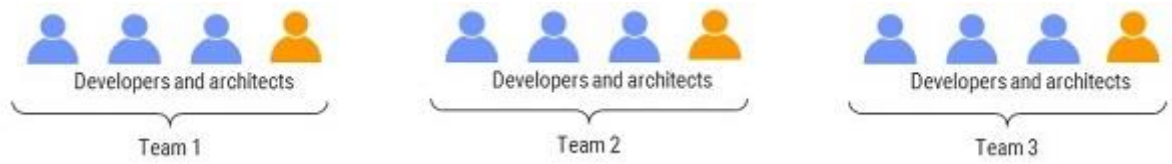
Architect owner je zpravidla velmi technicky zkušená osoba, která zasahuje ne jenom do vývoje architektury, ale také do jeho implementace. Mezi jeho klíčové dovednosti také patří komunikace se všemi zainteresovanými stranami a porozumění byznysu. Zpravuje a zodpovídá za týmovou architekturu (tzn. nejedná se jen o jeho výtvor). Má na starost zjednodušení architektonického modelování, společné porozumění a přijetí architektury a řízení úsilí architektonických změn. Rozhodnutí týkajících se změn by měla být vždy přijímána společně v týmu.

Existuje několik variant umístění architekta v rámci velkých projektů s několika agilními týmy. Na obrázku č. 4 vidíme rozložení, ve kterém agilní softwaroví architekti společně blízce spolupracují, ovšem na úkor vztahu s developery. V takovémto rozpoložení můžeme čelit dvěma hlavním problémům. Zaprvé tým architektů se může stát jakýmsi týmem “poptávajících”, zatímco týmy developerů “výrobci”. Takovýto případ by mohl skončit klasickým vodopádovým přístupem, v kterém developeři nevytvářejí architekturu, ale pouze implementují vizi architekta. Zadruhé tým developerů ztrácí motivaci, neboť nezasahuje do vývoje.

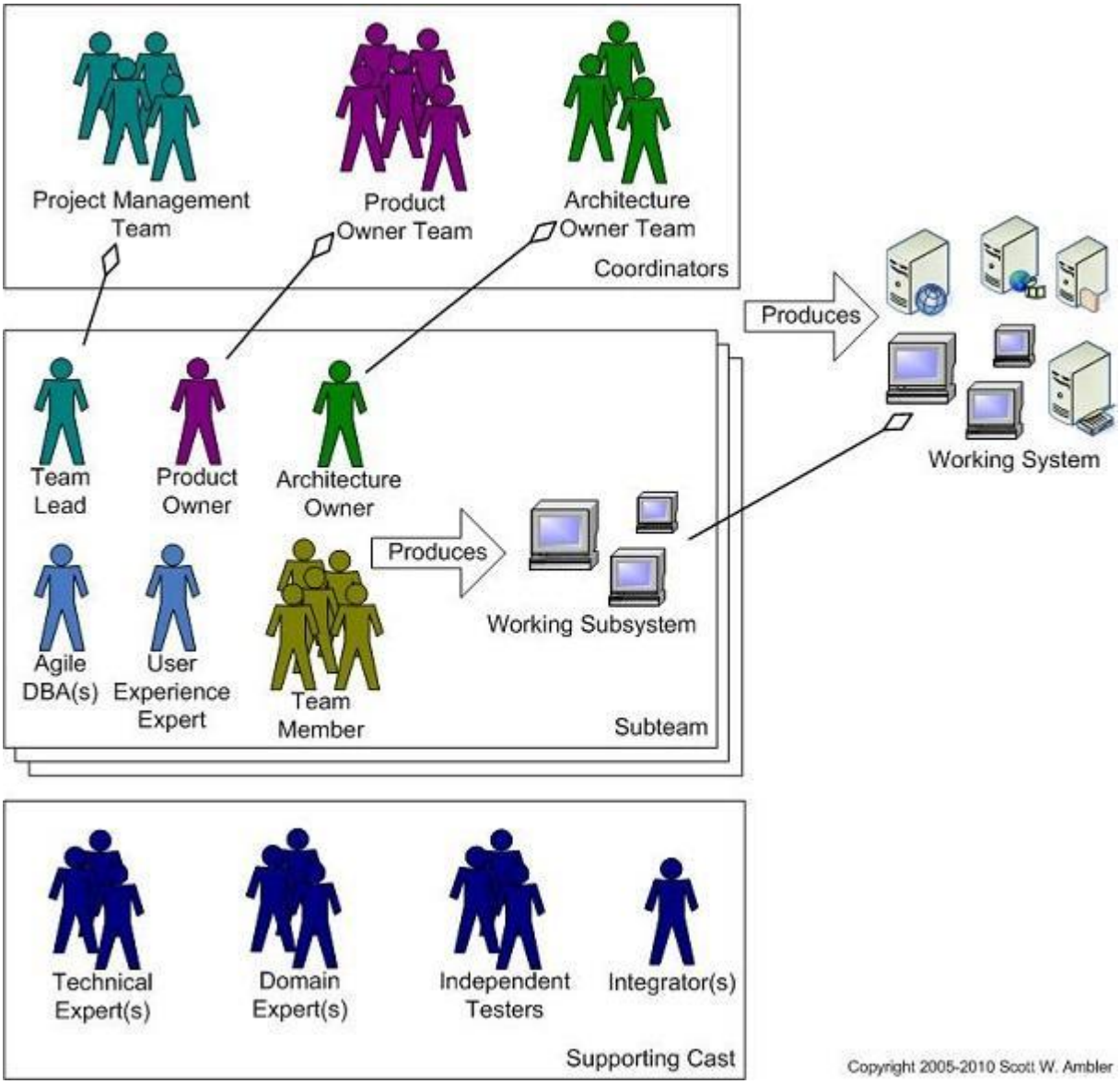


Obrázek 4 – Oddělený tým softwarových architektů pracující s týmy developerů [1]

Naproti tomu struktura zakreslená na obrázcích č.5 a č.6 představuje stav, kdy jsou agilní architekti součástí týmu developerů. Současně rozdělují svůj čas na práci v developerském týmu (zapojují se do implementace a přípravy pracujících výstupů subsytémů) a povinnosti vůči týmu architektů, či architect ownerů, starajících se o pracující celkový systém (udržují rovnováhu mezi vizí "big picture" a současným stavem). [1][9]



Obrázek 5 – Každý tým developerů má vlastního architekta [1]



Obrázek 6 - Organizační struktura velkého agilního týmu [9]

4.4 Dokumentace architektury

Při agilním vývoji platí pravidlo jednoduchosti a maximalizování množství práce, které není zapotřebí dělat. To samé platí i pro dokumentaci. Hlavním komunikačním kanálem agilního týmu je face-to-face komunikace. Dokumentace se využívá v případech, kdy není možné pro architect ownera blízce spolupracovat s developery, či pro snazší porozumění díky grafickému vyjádření. Mezi další důvody patří vytváření dokumentů pro základ analýzy a tvorbu systému a vzdělávání nově příchozích členů týmu, vývojářů, správců systému, atd. Vždy se snažíme o vytvoření co "nejlehčí" varianty dokumentace. Nevytváří se padesátistránkový dokument, pokud lze vše podstatné popsat v pěti stránkách. Nevytváří se pěti stránkový dokument, pokud pro

týmové porozumění postačí diagram atd. Nedokumentují se ty samé aspekty v několika různých formátech, postačí pouze jeden (pokud je zakreslen diagram, nepotřebuje se vytvářet jeho slovný popis v dalším dokumentu). Dokumentace je živý spis, který je potřeba neustále udržovat aktuální se zaznamenání každé změny architektury, neboť vývojáři, potřebují znát aktuální stav k opravě chyby či přidání nové funkce. Při vytváření platí výběr vždy nejjednoduššího nástroje pro zápis (tzn. není potřeba zaznamenávat diagram v složitém case nástroji, pokud pro pochopení stačí jednoduchý náčrtek). [1]

4.5 Výhody a nevýhody agilního přístupu

Mezi výhody agilní softwarové architektury lze zařadit flexibilitu, postupný vývoj a schopnost radikální změny i ve velmi pozdním stádiu projektu. Dále je založena na praktickém vývoji, tedy nejedná se jen o teoretickou vizi. Navíc vzniká díky týmové spolupráci, což vede ke zvýšení motivace a zainteresovanosti jednotlivých členů. Na druhé straně lze mezi nevýhody zahrnout slabší dokumentaci, která může způsobit problémy týmům, které převezmou projekt, aniž by se účastnili jeho dřívějšího vývoje. Dále je tento přístup silně závislý na komunikaci a týmové spolupráci. Bohužel ne všechny týmy a jedinci jsou schopny takto úzce kooperovat.

5 Porovnání tradičního a agilního přístupu

Tradiční softwarová architektura je vytvářena softwarovým architektem odděleně od zbytku týmu. Vzniká v přesně stanovený čas v úvodních fázích projektu, a po dokončení a zdokumentování je neměnná. Oproti tomu agilní architektura je vytvářena všemi členy týmu (se zodpovědností architect ownera), vzniká postupně a vyvíjí se podle potřeb projektu. To umožňuje flexibilitu a také zvýšenou zainteresovanost členů týmu.

Tradiční architekt je zaměřen na budování „big picture“ a zvažování všech možností, kam by se projekt mohl při vývoji posunout. Při agilním vývoji je potřeba nalézt vyvážený stav mezi „big picture“ a současným stavem – nevyvíjet to, co aktuálně nepotřebujeme, ale zároveň vytvořit udržitelnou platformu, na které je možné dále stavět.

Tradiční architektura vzniká na základě teoretických znalostí, architekt nebývá zapojen do samotného vývoje systému a hrozí vznik „ivory towers“. Naproti tomu agilní architektura stojí na „hands on experience“, architekt se přímo zapojuje do vývoje a nevytváří pouze teoretickou vizi.

Spolu s tradiční softwarovou architekturou vzniká veliký objem dokumentace, diagramů a podobně (tzv. „blueprints“), ve kterých je projekt popsaný ze všech možných perspektiv. Agilní projekty oproti tomu mohou trpět slabší dokumentací, která může způsobit komplikace, například týmům přebírajícím rozjetý projekt.

Každý z přístupů k softwarové architektuře má své výhody a nevýhody. Ačkoli ze srovnání vychází na první pohled lépe agilní metodiky, záleží vždy na typu projektu. Vybrat přístup na základně konkrétního projektu je zásadní.

6 Závěr

První cíl práce byl definovat softwarovou architekturu a následně ji popsat z hlediska tradičních a agilních metodik. Jelikož neexistuje oficiální definice softwarové architektury, byla tato popsána vyhledáním společných prvků v existujících definicích. Následoval detailnější popis obou přístupů, který čtenáři poskytl ucelený obraz o přístupech k softwarové architektuře.

Druhým cílem bylo srovnat výhody a nevýhody obou přístupů. Tohoto cíle bylo dosaženo pomocí detailního popisu obou přístupů a jejich porovnáním ve stanovených oblastech.

Po přečtení této práce by čtenář měl mít přehled o rozdílech v tradičním a agilním přístupu k softwarové architektuře. Zároveň by měl být schopen vyhodnotit výhody a nevýhody obou přístupů.

7 Seznam zdrojů

[1] MIHAYLOV, Boyan. Towards an Agile Software Architecture. *InfoQ* [online]. 2015 [cit. 2016-05-15]. Dostupné z: http://www.infoq.com/articles/towards-agile-software-architecture

[2] EELES, Peter. What is a software architecture? *Ibm.com* [online]. 2006 [cit. 2016-05-15]. Dostupné z: [2] http://www.ibm.com/developerworks/rational/library/feb06/eeles/

[3] BUCHALCEVOVÁ, Alena (2009). Metodiky budování informačních systémů. 1. vyd. Praha: Oeconomica. 206 s. ISBN 97880-245-1540-3.

[4] PEJCHAL, ING, Jakub. Agilní a tradiční metodiky v projektovém řízení [online]. Brno, 2015 [cit. 2016-05-15]. Dostupné z: *http://is.muni.cz/th/211842/fi_m/Agilni_a_tradicni_metodiky_v_projektovem_rizeni.pdf*

[5] Agile Architecture: Strategies for Scaling Agile Development. Agile Modeling [online]. [cit. 2016-05-15].

[6] KADLEC, V. Agilní programování : metodiky efektivního vývoje softwaru. 1. vyd. Brno: Computer Press, 2004. 278 p.

[7] DEBESARQUES, Steven. Agile Project mgmt at CCE. Belgie, 2015

[8] BUCHALCEVOVÁ, Alena. Agilní metody, jak dál? Ostrava 28.05.2008 – 30.05.2008. In: Tvorba softwaru 2008. Ostrava : VŠB TU EF, 2008, s. 30–34. ISBN 978-80-248-1765-1.

[9] The Architecture Owner Role: How Architects Fit in on Agile Teams. Agile modeling [online]. [cit. 2016-03-13]. Dostupné z: http://www.agilemodeling.com/

8 Seznam obrázků