Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Vliv komponentového vývoje na vývojový proces softwaru

Bc. Vojtěch Dušek

© 2021 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Vojtěch Dušek

Systémové inženýrství a informatika
Informatika

Název práce

Vliv komponentového vývoje na vývojový proces softwaru

Název anglicky

The influence of component-based development on the software development process

---

Cíle práce

Hlavním cílem diplomové práce je analýza komponentového přístupu při vývoji frontendu aplikace z hlediska procesu vývoje software a následné vyhodnocení výhod a nevýhod tohoto přístupu v porovnání se standardním přístupem k vývoji softwarových aplikací.
Dílčím cílem práce je vývoj konkrétní komponentové knihovny ve frameworku React Native a komplexní charakteristika a zhodnocení způsobu tvorby komponent v kontextu frameworku React Native.

Metodika

Metodika teoretické části diplomové práce vychází ze studia odborných informačních zdrojů vztahujících se ke zvolenému tématu. Na základně analýzy odborných zdrojů budou formovány vlivy komponentové tvorby na vývojový proces software.

V praktické části práce bude provedena implementace frontendové komponentové knihovny ve frameworku React Native pro mobilní aplikaci a na základě metrik ISO / IEC 25000 budou ověřena východiska teoretické části diplomové práce. Na základě syntézy poznatků z rešeršní části a výsledků dosažených v praktické části práce, budou formulovány závěry diplomové práce.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

komponentový vývoj, vývojový proces, react native, přepoužitelnost, komponenta, frontend, mobilní aplikace

---

Doporučené zdroje informací

BROWN, Alan W.. Large-Scale, Component-Based Development. Prentice Hall PTR. First Edition. 2000. ISBN: 0-13-088720-X

CRNKOVIC, CHAUDRON M. and LARSSON S.. Component-Based Development Process and Component Lifecycle. 2006. Tahiti: 2006. ISBN 0-7695-2703-5

EISENMAN, Bonnie. Learning React Native – Building native mobile apps with JavaScript. Second Edition. O'Reilly Media, 2017. ISBN 978-1-491-98914-2.

HUMPHREY, Watts S.. Managing the Software Process. 1989. Addison-Wesley Professional. ISBN: 0-201-18095-2

WHITCHEAD, Katherine. Component-Based Development: Principles and Planning for Business Systems. 2002. Boston: Addison-Wesley. ISBN 0-201-67528-5.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Petr Benda, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 20. 7. 2020

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 14. 11. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci Vliv komponentového vývoje na vývojový proces softwaru jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2021

____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Petru Bendi, Ph.D. za vstřícný přístup a věcné připomínky. Dále bych chtěl také poděkovat všem blízkým, kteří mě podporovali v průběhu práce.

Vliv komponentového vývoje na vývojový proces softwaru

Abstrakt

V teoretické části práce je vysvětlen klasický a komponentový přístupu vývoje software včetně příkladů vývojových modelů. Byly zdůrazněny obecné výhody a nevýhody komponentového přístupu a představen standart ISO/IEC 250xx – SquaRE, ze kterého byly vybrány vhodné metriky pro porovnání komponentového a klasického přístupu vývoje software a měření jakosti software v rámci případové studie. Závěrem teoretické části jsou představeny základy JavaSciprtového frameworku React Native v kontextu komponentového vývoje.

V praktické části je popsán postup vývoje frontendové komponentové knihovny podle komponentového Y-modelu pro případovou studii mobilní aplikace v React Native. Vybrané části mobilní aplikace jsou implementovány také klasickým přístupem. Vybrané metriky standartu ISO / EIC 25023 byly spočítány a porovnány pro oba použité přístupy.

Na základě získaných poznatků z teoretické a praktické části práce jsou vyhodnoceny závěry pro srovnání výhod a nevýhod komponentového přístupu v porovnání s klasickým přístupem.

Ze zjištěných závěrů vyplývá, že případová studie potvrdila výhody komponentového přístupu oproti klasickému přístupu a pomáhá ke snížení vynaložených nákladů na projekt.

Klíčová slova: komponentový vývoj, vývojový proces, react native, přepoužitelnost, komponenta, frontend, mobilní aplikace

The influence of component-based development on the software development process

Abstract

The theoretical part of the final thesis describes the classical and component approach to software development, including examples of development models. The general advantages and disadvantages of the component approach were emphasized. The ISO / IEC 250xx – SquaRE standard was introduced with selected suitable metrics for comparing the component and classical approaches of software development and measuring software quality in case study. The basics of the JavaScript framework React Native were introduced in the context of the component development approach.

The practical part of the final thesis describes the process of developing a frontend component library according to the component Y-model for a case study of a mobile application in React Native. Selected parts of the mobile application are also implemented by the classic approach. Selected metrics of the ISO / EIC 25023 standard were calculated and compared for both approaches.

Conclusions are evaluated to compare the advantages and disadvantages of the component approach in comparison with the classical approach.

Evaluated conclusions found that the case study of thesis confirmed the advantages of the component approach in compare with classical approach. The component approach helps to reduce the costs of the project.

Keywords: component development, development process, react native, reusability, component, frontend, mobile application

Obsah

1 Úvod **12**

Cíl práce a metodika **13**

1.1 Cíl práce 13

1.2 Metodika 13

2 Teoretická východiska **14**

2.1 Proces vývoje software 14

2.1.1 Klasické modely procesu vývoje software 14

2.1.2 Komponentový přístup a modely procesu vývoje software 20

2.2 Jakost softwarového produktu 31

2.2.1 ISO/IEC 250xx - SQuaRE 32

2.2.2 Další metriky 35

2.3 React Native 36

2.3.1 Komponenty 38

3 Vlastní práce **40**

3.1 Vývoj a implementace komponent 40

3.1.1 Doménové inženýrští 40

3.1.2 Systémová analýza 41

3.1.3 Návrh systému 49

3.1.4 Frameworking 53

3.1.5 Implementace komponent 55

3.1.6 Sestavení 61

3.1.7 Archivace 62

3.1.8 Testování 63

3.2 Analýza jakosti obou přístupu 63

3.2.1 Vybrané metriky ISO / EIC 25023 64

3.2.2 Další metriky 68

3.3 Výsledky porovnání komponentového a klasického přístupu vývoje 69

4 Závěr **71**

5 Seznam použitých zdrojů **74**

Seznam obrázků

Obr. 1 Vodopádový model životního cyklu [2] 15

Obr. 2 Spirálový model životního cyklu [2] 16

Obr. 3 RUP model životního cyklu [3] 18

Obr. 4 Životní cyklus komponentového přístupu [16] 24

Obr. 5 Obecný model komponentového přístupu [11] 26

Obr. 6 Rapid Application Development model [5] 27

Obr. 7 V-Model 1 [15] 27

Obr. 8 V-Model Detail [15] 28

Obr. 9 Y-Model [17] 29

Obr. 10 Model-W [16] 31

Obr. 11 Struktura SQuaRE [18] 33

Obr. 12 Výpočet virtuálního DOM a vykreslení [24] 37

Obr. 13 Přihlašovací obrazovka (zdroj: vlastní zpracování) 42

Obr. 14 Styly tlačítek (zdroj: vlastní zpracování) 43

Obr. 15 Domovská obrazovka (zdroj: vlastní zpracování) 43

Obr. 16 Domácí obrazovka se skrytým menu (zdroj: vlastní zpracování) 45

Obr. 17 Obrazovka novinek a událostí (zdroj: vlastní zpracování) 45

Obr. 18 Obrazovka detail novinky (zdroj: vlastní zpracování) 46

Obr. 19 Obrazovka pracovní nabídky (zdroj: vlastní zpracování) 47

Obr. 20 Obrazovka jídelníčků (zdroj: vlastní zpracování) 48

Seznam tabulek

Tabulka 1 Zvýšení kvality softwaru v komponentovém přístupu [6] 22

Tabulka 2 Porovnání komponentového a tradičního přístupu [14] 23

Tabulka 3 Vybrané metriky ISO / EIC 25023 [20] 34

Tabulka 4 Porovnání klasického a komponentového přístupu (zdroj: vlastní zpracování) 70

1 Úvod

V dnešní době probíhá masivní přesun do online prostředí, kde je umožněno jednoduše a rychle pro zákazníka nabídnout produkty nebo službu. V online světě vznikl kompletně nový trh oddělený od starého s tím rozdílem, že online svět potřebuje nové služby a aplikace vytvořit a naprogramovat.

Vzniká tak obrovská poptávka po nových aplikacích a službách, které umožňují vznik a růst tohoto segmentu, který se rychle stane dominantním a hlavním proudem pro většinu populace.

S větším a větším rozšiřováním aplikací a programů zajišťující chod tohoto nového světa rostou také náklady a úsilí udržet tento svět v chodu. Vývojové společnosti se snaží snižovat náklady na vývoj, jelikož náklady na rozvoj a údržbu programů poskytovaných jejich klientům tvoří velké části jejich rozpočtu. Vytvářet programy a aplikace kompletně od nuly je velice nákladné jak časově, tak finančně a údržba obrovských systémů je velice drahá.

Zde se nabízí způsob, jak levněji a rychleji vyvinout potřebný program, jak ho jednoduše udržovat a nenechat ho narůst do obřích rozměrů, kde jediná úprava může zapříčinit následky, které je nutné dlouhé hodiny opravovat a zkoumat.

Komponentový přístup vývoje software pomůže udržet strukturu programu, oddělit funkční části, zjednodušit a zrychlit údržbu a opravy, zrychlit nový vývoj nebo dokonce jednoduše poskládat výsledný program z již hotových funkčních celků za nízké náklady v krátkém čase, které si může zákazník dovolit zaplatit.

Pro vývoj s komponentovým přístupem je potřeba přehodnotit zaběhlé a známé postupy a procesy pro vývoj software. Do procesů je potřeba začlenit vývoj nových komponent a zároveň uvažovat použití již existujících a dostupných komponent namísto nového vývoje. Nejedná se pouze o krok implementace, ale o kompletní uvažování již od prvních kroků návrhu systému. Musí se uvažovat komponentově a maximalizovat přepoužívání komponent napříč programem za účelem zjednodušení a zefektivnění vývoje finálního produktu.

Cíl práce a metodika

1.1 Cíl práce

Hlavním cílem diplomové práce je analýza komponentového přístupu při vývoji frontendu aplikace z hlediska procesu vývoje software a následné vyhodnocení výhod a nevýhod tohoto přístupu v porovnání se standardním přístupem k vývoji softwarových aplikací.

Dílčím cílem práce je vývoj konkrétní komponentové knihovny ve frameworku React Native a komplexní charakteristika a zhodnocení způsobu tvorby komponent v kontextu frameworku React Native.

1.2 Metodika

Metodika teoretické části diplomové práce vychází ze studia odborných informačních zdrojů vztahujících se ke zvolenému tématu. Na základně analýzy odborných zdrojů budou formovány vlivy komponentové tvorby na vývojový proces software.

V praktické části práce bude provedena implementace frontendové komponentové knihovny ve frameworku React Native pro mobilní aplikaci a na základě metrik ISO / IEC 25000 budou ověřena východiska teoretické části diplomové práce. Na základě syntézy poznatků z rešeršní části a výsledků dosažených v praktické části práce budou formulovány závěry diplomové práce.

2 Teoretická východiska

2.1 Proces vývoje software

Software

Software je tvořen množinou programových jednotek a jejich vzájemných vazeb. Jednotky softwarového produktu mohou být moduly, objekty, komponenty nebo služby, které navzájem komunikují a definují chování produktu. [1]

Metodika, model životního cyklu, proces vývoje software

Pojmy metodika, model životního cyklu nebo proces vývoje software lze v kontextu této diplomové práce považovat za téměř identické pojmy, které popisují komplexní postupy nebo návody pro vývoj softwaru. Metodika popisuje všechny etapy řešení, tedy u softwaru se jedná o všechny fáze životního cyklu softwaru. Metodika řeší problematiku z nadhledu a nepopisuje detailně, jak danou operaci provést. Metodiky vývoje softwaru procházejí určitým vývojem a odráží požadavky na vývoj softwaru dle doby, kdy byly definovány. [2]

2.1.1 Klasické modely procesu vývoje software

Klasické neboli tradiční modely jsou prověřené léty používáním ve vývojovém životním cyklu software. Tradiční metody jsou orientovány na funkcionalitu a požadavky jsou definovány vždy na počátku životního cyklu softwaru a v jeho průběhu jsou neměnné. Mění se pouze zdroje nebo čas. Charakteristická je objemná dokumentace, direktivní řízení a nemožnost se odchýlit od plánu. [2]

Nejznámější a velice využívaný model je vodopádový model. I když je vnímán, jako neefektivní model, často se využívá pro jeho jednoduchost, snadné použití a jednoduchou kontrolu. Jedná se o nejvíce statického zástupce klasických modelů. [2]

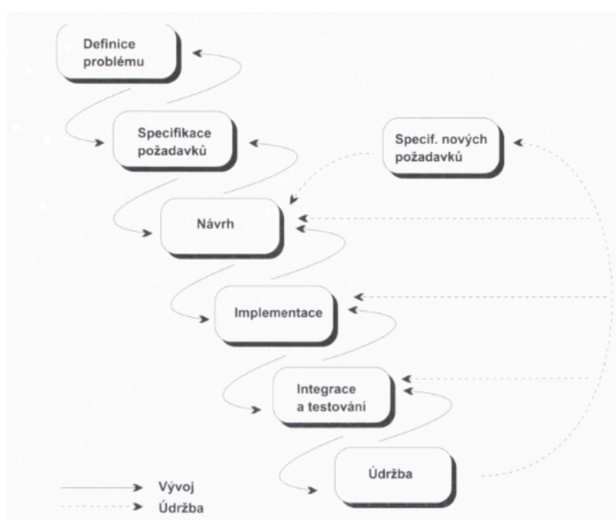
Spirálový model se snaží odbourat nedostatky vodopádového modelu. Zaměřuje se na analýzu rizik a používá iterativní způsob vývoje, který je velice efektivní na vývoj rozsáhlých projektů. [2]

Rational Unified Proces model se přizpůsobuje typu projektu a vývojovému týmu a existují různé vyrianty tohoto modelu dle specifik projektu. Jedná se o velice komplexní a variabnilní model, a proto má větší náklady při plánování projektu. [2]

Unified Process model je open source odlehčená verze Rational Unified Proces modelu. Kvůli menšímu rozsahu je vhodný spíše pro menší projekty. [2]

2.1.1.1 Vodopádový model

Nejstarší model životního cyklu, který definoval Winston W. Royce v roce 1970. V dnešní době se od tohoto modelu spíše upouští a je nahrazován modernějšími a efektivnějšími modely, ale nelze mu upřít převrat, který vnesl členění softwerového procesu s logickou návazností v celém odvětví vývoje software. Model je sekveční s jednotlivými fázemi, které jsou prováděny v přesně definované posloupnosti. Viz obr. 1. Nelze vstoupit do další fáze, pokud není kompletně dokončena předchozí fáze. Velký zřetel se bere na počáteční fáze, jelikož odhalení chyby při navrhování systému není tak nákladné, jako odhalení chyby v pozdějších fázích. Neprobíhají žádné iterace nebo průběžná komunikace s klientem. [2]



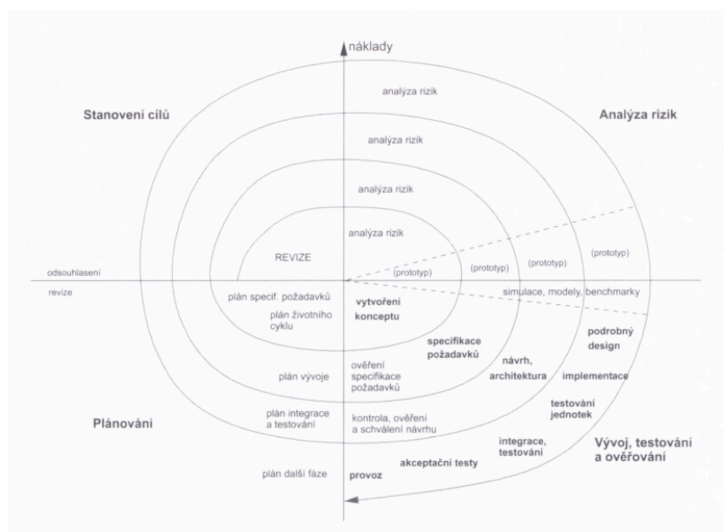
Obr. 1 Vodopádový model životního cyklu [2]

Nulová možnost reagovat na změnové požadavky klienta. Vývojový proces běží dle definovaných specifikací na počátku. Díky tomu lze jednoduše určit harmonogram a projekt přehledně řídit a kotrolovat. Nevýhodou je testovovací fáze až na konci samotného životního cyklu, jelikož závažné chyby se mohou projevit při testování.

2.1.1.2 Spirálový model

V roce 1986 byl definován Spirálový model Berrym Boehmem, který vylepšil původní vodopádový model o prvky iterativního přístupu a opakované analýzy rizik. Výsledek analýzy rizik v každé iteraci ovlivňuje následující vývoj v dané iteraci. Změny jsou oproti vodopádovému přístupu možné, ale vždy jen na konci každé iterace. Každá iterace má následující kroky: [2]

- Stanovení cílů – analýza cílů, rizik a rozsahu iterace
- Analýza rizik – vyhodnocení rizik
- Vývoj – vývoj a ověření výstupů vývojové fáze
- Plánování – pro další iteraci



Obr. 2 Spirálový model životního cyklu [2]

Spirálou se začíná ve středu směrem ven a spirála poukazuje na jednotlivé fáze, ve kterých se iteruje. Spirála naznačuje finanční a časové náklady. [2]

- **1. iterace** se zabývá riziky ohrožujícími vývoj, definováním základního konceptu vývoje a zvolení metod
- **2. iterace** se zaměřuje na ověření specifikace požadavků a znovu je provedeno stanovení cílů, analýza rizik a plánování
- **3. iterace** se věnuje vytvoření a ověření detailního designu a znovu je provedeno stanovení cílů, analýza rizik a plánování
- **4. iterace** provede implementaci, testování, integraci a znovu je provedeno stanovení cílů, analýza rizik a plánování

Všechny události, které můžou ohrozit splnění předem definovaných cílů, jsou považovány za rizika. Pokud je riziko identifikováno, je potřeba zjistit jeho pravděpodobnost a rozsah ohrožení projektu. Z tohoto důvodu v každé iteraci se provádí fáze analýza rizik.

Existují tři druhy prototypu ve spirálovém modelu životního cyklu. [2]

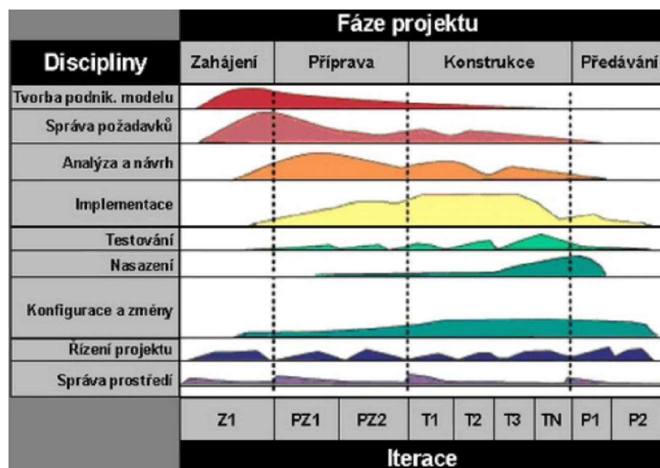
- **Ilustrativní prototyp** se soustředí na vzhled a uživatelské rozhraní
- **Funkční prototyp** se zaměřuje na jádro systému a cyklicky přibývají implementace rozšiřujících funkčností
- **Ověřovací prototyp** kontroluje vymezenou část projektu z pohledu funkčního a technologického

Spirálový model je vhodnější pro větší projekty, kde jsou dostupní experti na vyhodnocování rizik a řízení složitějšího iterativního přístupu. Z tohoto důvodu odradí dost vývojových týmů. Model počítá s řadou rizik a změnových požadavků v průběhu vývoje.

2.1.1.3 Rational Unified Proces model (RUP)

Společnost Rational Software definovala koncept modelu založeného na iterativním přístupu vývoje a architektury, který vychází z Objectory Procesu. Objectory proces poprvé zužitkoval případ užití a objektově orientovaný návrh na základu zkušeností Ivara Jacobsona z roku 1987. RUP je založeno na objektově orientovaném iterativním přístupu využívající UML jazyk. Základním prvkem je případ užití, což je posloupnost kroků systému za účelem dodání hodnoty uživateli.

Největší důraz je kladen na analýzu, desing, řízení zdrojů, dokumentaci a plánování. [2]



Obr. 3 RUP model životního cyklu [3]

Fáze RUP modelu, které jsou uvedeny na vodorovné ose: [2]

- **Zahájení** – definuje účel, případy užití, rozsah, náklady a rizika projektu
- **Příprava** – analýza rizik, návrh architektury, plánování, přípravy tvorby evolučního prototypu
- **Konstrukce** – dokončení analýzy, designu, prototypů, zahájení testování a vyhodnocení
- **Předávání** – předání hotového projektu klientovi, zaškolení, opravy a ladění

Na svislé ose se vyskytují procesy, u kterých je vyznačena míra důležitosti v každé fázi modelu. V každé fázi je každý proces různě intenzivní až na řízení projektu a správu prostředí, která je v každé fázi vždy stejně intenzitní.

Model vymezuje základní praktiky pro efektivní vývoj: [3]

- **Iterativní vývoj software** – výhodou je odhalení rizik a možnost změn již v úvodních fázích
- **Správa požadavků** – požadavky se mohou dynamicky měnit
- **Architektura založená na komponentách** – implementace již vytvořené komponenty může uspořit významný čas
- **Vizuální modelování** – model neboli zjednodušená realita v jazyce UML umožní lepší pochopení systému a simulace systému

- **Ověřování kvality software** – průběžné testování a zjišťování je méně nákladné než ve finální fázi projektu
- **Řízení změn software** – na změny je třeba reagovat a zvážit přizpůsobení vývojového procesu dle aktuální situace

Dále model vymezuje základní elementy [2]

- **Pracovníci** – chování a odpovědnost týmu i jednotlivců
- **Činnosti** – práce vykonaná týmem nebo jednotlivcem za konkrétním účelem
- **Meziprodukty** – prezentovatelné části projektu
- **Pracovní procesy** – definovaná posloupnost kroků vedoucí k cíli

Model je robustní, přizpůsobivý a obecný, a proto se hodí pro dlouho řadu projektů a týmů. [2]

2.1.1.4 Unified Proces (UP)

UP vychází stejně jako RUP z Objectory Procesu od Ivara Jacobsona z roku 1987. Jedná se o zjednodušený, ale zato open source model, u kterého není k dispozici tolik nástrojů, ale je dostačují pro řízení projektu v iteracích s objektově orientovaným přístupem. Hlavní myšlenky UP jsou: [2]

- **Rizika a případy užití** jsou hlavními faktory pro řízení vývoje
- **Architektura systému** je stěžejní pro celý projekt
- **Iterativní a inkrementální** způsob vývoje

Každá iterace obsahuje malý vodopádový přístup skládající se z: [2]

- Plánování
- Analýza a návrh
- Implementace
- Testování
- Předání projektu

Již v průběhu vývoje klient dostává nové funkce, které jsou funkční a s každou další iterací produkt nabývá nových funkcí. Fáze UP jsou stejné jako u modelu RUP.

2.1.2 Komponentový přístup a modely procesu vývoje software

Komponentový přístup vývoje má nesporné výhody oproti tradičnímu přístupu v přepoužitelnosti a komponentové orientaci, která zajišťuje spolehlivější a méně nákladný vývoj software. [6] Vývoj velkých, komplikovaných systémů, které musí být udržované a upravované, čelí požadavkům na rychlý a ekonomicky šetrný vývoj. Odpovědí na tyto požadavky je komponentově orientovaný přístup, neboli CBSD. Komponentový přístup je založen na vybírání spolehlivých, robustních a opakovaně použitelných softwarových komponentách, pokud jsou dostupné k pořízení od externího dodavatele a lze je integrovat do architektury vyvíjeného produktu. V opačném případě si tým tyto softwarové komponenty vyvine sám od nuly tak, aby splňovaly vlastnosti komponentového přístupu a mohl je jednoduše přepoužít a integrovat do systému. Tím pádem se stává velice důležité správně identifikovat vhodnou komponentu, kterou lze jednoduše integrovat a je vhodná na opakované použití tak, aby se vyplatily náklady na ni vynaložené. [7] CBSD přístup se úspěšně ověřil jako výhodný přístup v mnoha ohledech, a proto se stále vyvíjí a je široce využíván při vývoji e-business, kancelářských, desktopových i webových aplikacích. [8]

CBSD je hojně využíván, jelikož nabízí jednodušší řízení vývoje složitých systémů, zlepšování kvality softwaru, zkrácení doby a s tím spojených nákladů na vývoj, zvýšení produktivity vývojového týmu, zkrácení doby uvedení produktu na trh, nižší náklady na údržbu, zvýšenou spolehlivost systému a navíc opětovné použití komponenty v budoucím vývoji nebo i příležitost nabídnout hotové komponenty jako samostatný produkt k prodeji napříč celým trhem. [8] CBSD nabízí úplně nový trend ve vývoji velkých systémů za předpokladu, že mezi mnoha velkými a robustními systému existuje nezanedbatelná podobnost, díky které lze přejít od implementace systému od nulky k integraci částí systémů. [9]

Komponentu lze definovat jako samostatnou jednotku s přesně definovaným rozhraním a s přímou závislostní pouze na kontextu. Komponentu lze nezávisle nasadit a je vyvinuta za účelem použití pro třetí stranu. [10] Nejdůležitější vlastností komponenty je její rozhraní. Pomocí rozhraní komponenta komunikuje s dalšími komponentami nebo s prostředím, kde je implementována. Rozhraní komponenty je přístupový bod, který nabízí sadu popsaných metod, které prostředí může použít. Rozhraní má svou specifikaci, která je respektována komponentou a prostředím, které komponentu využívá. Rozhraní má dva

základní typy rozhraní. Vyžadované a nabízené rozhraní. Vyžadované rozhraní popisuje základní funkčnost, aby se zajistil bezchybný běh komponenty v prostředí nebo při komunikaci s ostatními komponentami. Naopak nabízené rozhraní popisuje funkčnost, kterou samotná komponenta nabízí. Každá komponenta by měla mít jasně a srozumitelně definované rozhraní, které není skryté, opakující se, proměnlivé a upravitelné, ale zároveň odpovídá modelu komponenty, je nezávislé na prostředí, má specifické funkce, odpovídá dokumentaci a potvrzuje myšlenku přepoužitelnosti. [11]

2.1.2.1 Obecné výhody komponentového přístupu

Zkrácený vývojový čas

Komponentový přístup značně napomáhá zkrátit vývojový čas a tím i náklady. CBSD přístup používá již vyvinuté komponenty buď interně nebo z externího zdroje. Následkem je dramatické zkrácení doby vývoje, pokud potřebné komponenty jsou již dostupné a není je potřeba vyvinout od nuly. Výsledkem není implementace detailní funkčností, ale pouze ladění efektivní komunikace komponent mezi sebou a prostředím, integrace komponent do architektury systému a možné úpravy komponent tak, aby odpovídaly požadavkům. [7]

Snížení vynaloženého úsilí

Komponentový přístup systémově směřuje k maximálnímu přepoužití komponent a tím snižuje nadbytečnou práci vynaloženou vývojovým týmem. S tím jsou spojené náklady a zkrácení dokončení produktu. [12]

Přepoužitelnost

Komponenty jsou od začátku vývoje zamýšleny tak, aby bylo je možné přepoužít v různých systémech. Z toho důvodu jsou velice dobře popsané jejich rozhraní, ale funkčnost je zapouzdřena. Díky přepoužitelnosti v jednou vytvořené komponenty se tento přístup stává velice populární ve vývojářské komunitě. Komponenty pochází z existujích knihoven, což znamená, že byly již mnohokrát použity a otestovány. To vede k rychlejšímu dokončení produktu a snížení nákladů. [13]

Přizpůsobivost

Komponenta může být před finálním použitím upravena do požadovaného stavu dle specifikace různými metodami. Například white box a black box metody. [13]

Škálovatelnost

Použitím komponentového přístupu se automaticky zvyšuje škálovatelnost systému. Použité komponenty jsou uloženy na jednom místě v knihovně komponent. Případná úprava nebo úprava funkčnosti se upravuje pouze v komponentě a není potřeba zásahu do více částí systému, pokud se změna netýká rozhraní komponenty. [13]

Kvalita softwaru

Základní princip zvyšování kvality softwaru, je systematické testování a oprava chyb. Pokud software používá komponentový přístup a komponenty jsou používány ve více systémech, tak je zřejmé, že komponenty se ladí a vylepšují s každým použitím. Je zřejmé, že komponenta použitá víckrát bude mít vyšší kvalitu než komponenta, která je použitá pouze jednou. [12]

Standardizace ISO 9126 definuje šest charakteristik, jak dosáhnout kvalitního softwaru. V následující tabulce je popsáno, jak komponentový přístup ovlivňuje ISO charakteristiky. [6]

Tabulka 1 Zvýšení kvality softwaru v komponentovém přístupu [6]

Funkčnost	Funkčnost se dramaticky zvyšuje použitím již existujících komponent.
Udržovatelnost	Modulová architektura komponentového přístupu umožní jednoduchou výměnu komponent.
Použitelnost	Použití komponenty ve více systémech podporuje standardizaci grafického rozhraní a pro uživatele zjednoduší orientaci v různých systémech, které používají stejné komponenty.
Efektivita	Výkonnost systému může být ovlivněná pouze několika málo komponentami v celém systému. Tyto komponenty mohou být interně optimalizovány bez ovlivnění specifikace nebo komponenty lze přesunout mezi platformami za účelem zvýšení výkonu, aniž by to ovlivnilo funkčnost systému.
Spolehlivost	Spolehlivost je zvyšována s každým použitím komponenty v dalším

	systému, jelikož komponenty procházejí procesem kontroly kvality opakovaně. Systém postavený na kvalitních komponentách obecně zvyšuje svoji spolehlivost.
Přenositelnost	Specifikace komponenty není závislá na platformě. Komponentu lze rychle vyvinout pro novou platformu, aniž by to mělo dopad na ostatní komponenty.

Produktivita

Produktivita se zvyšuje díky menšímu počtu řádků kódu. To má za následek i snížení úsilí při testování a také šetří práci při analýze a návrhu systému. Obecně tedy přispívá ke snížení nákladů. Zpočátku vývoje se produktivita nezvyšuje z důvodu potřeby vyvinout komponenty, které lze v dalších fázích vývoje přepoužít. Z dlouhodobého hlediska se produktivita začne významně zvyšovat. [12]

2.1.2.2 Životní cyklus komponentového přístupu

Životní cyklus vývoje je zaměřen na aktivity spojené s vývojem jako jsou analýza požadavků, design, vývoj a testování. Životní cyklus se skládá z prováděných kroků v přesně definovaném pořadí, což pomáhá dosáhnout konzistenci v projektu, na kterém pracuje spousta lidí s různými specializacemi. Životní cyklus nebo také procesní model zahrnuje všechny činnosti po celou dobu životnosti projektu. Tradiční modely nejsou použitelné pro komponentový přístup, jelikož žádný z nich výslovně nepodporuje princip opakovaného použití vývoje, a proto komponentový přístup navrhl své vlastní modely. [14]

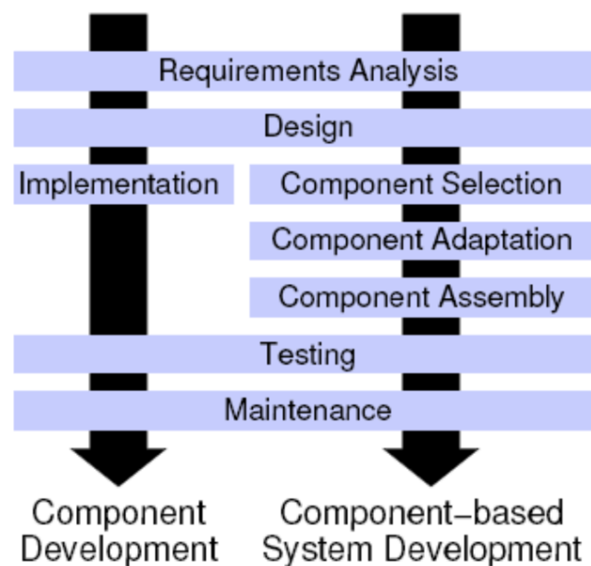
Tabulka 2 Porovnání komponentového a tradičního přístupu [14]

Komponentový přístup vývoje	**Tradiční přístup vývoje**
Vývoj systému z existujících komponent	Vývoj systému od nuly
Komponenty a systém integrované z těchto komponent jsou již vyvinuty	Systém je vyvinut
Výběr a hodnocení komponent je speciální	Žádní podobné kroky v životním cyklu

krok životního cyklu	
Velké úsilí je věnováno výběru komponent, testování a ověřování funkčnosti	Velké úsilí je věnováno vývoji
Přepoužití je hlavním cílem	Přepoužití není uvažováno

Hlavní myšlenkou komponentového přístupu je postavit systém z již existujících komponent. To rozděluje vývoj do dvou úrovní. [15]

- **Vývoj systému**– identifikace přepoužitelných komponent na základně požadavků a integrace komponent, které a utváří celek
- **Vývoje komponent** – vývoj komponent od nuly vývojovým týmem nebo dodání komponent od třetích stran



Obr. 4 Životní cyklus komponentového přístupu [16]

V obrázku 4 levá část představuje vývoj systémů zahrnující fáze požadavků, analýzy, designu, implementace komponent, testování a údržbu systému. V pravé části je vývoj komponent, kdy je proces rozšířen o výběr, úpravu a sestavené komponenty. Více pozornosti by mělo být věnováno vývoji komponent. [16]

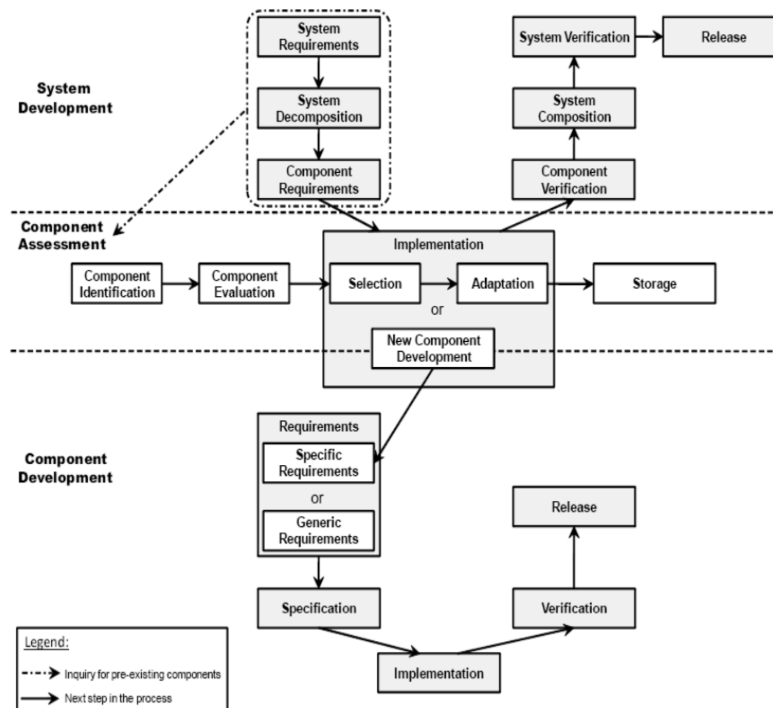
Životní cyklus komponentového vývoje má svá specifika, které je potřeba dodržet, abychom ve vývojovém procesu dosáhli všech výhod, které komponentový vývoj nabízí. [4]

- **Identifikace komponent** – vybrání možných komponent z vývojového a businessového hlediska, seznam slouží pro další kroky
- **Vybrat komponent odpovídající požadavkům** – požadavků nelze vždy dosáhnout za pomoci pouze existujících komponent. Architektura a analýza musí zohlednit komponenty, které jsou dostupné.
- **Vytvoření proprietárních komponent** – komponenty, které nejsou dostupné nebo neexistují, je nutné vyvinout. Je to nákladnější postup, ale základní komponenty by měly být vyvinuty interně, protože budou klíčovou vlastností produktu.
- **Upravit vybrané komponenty dle požadavků** již existujích komponent – některé komponenty jsou použity bez úprav, některé je potřeba upravit parametrizací a některé komponenty je potřeba obalit kódem, aby mohly být efektivně využity.
- **Vytvořit a použít komponenty pomocí frameworku pro komponenty** – framework, který je v základu systému a zajišťuje komunikaci mezi komponentami.
- **Nahrazení zastaralých komponent** – tento krok je součástí údržby, opravy chyb nebo vývojem nových požadavků a funkčností.

2.1.2.3 Obecný model komponentového přístupu

Základním principem komponentového přístupu je rozdělit procesy do vývoje systému a vývoje komponent. Oba dva oddělené procesy mohou postupovat klasickým přístupem s kroky: definování požadavků, specifikace, implementace a testování. V komponentovém přístupu jsou výhodou existující komponenty pro krok definování požadavků, jelikož lze přizpůsobit požadavky již od začátku dle komponent. Systémové požadavky se dělí na podrobnější požadavky na základně znalosti domény a dostupných modulů a komponent. Vznikají tedy komponentové požadavky, které vycházejí ze systémových požadavků. Komponenty, které přesně neodpovídají požadavkům, budou upraveny. Požadavky, které komponenty nebudou splňovat, budou vyvinuty ve fázi vývoje komponent. Po implementaci komponent a ověření komponentových požadavků budou

všechny komponenty integrovány do finálního systému, který je ověřen dle systémových požadavků. [11]



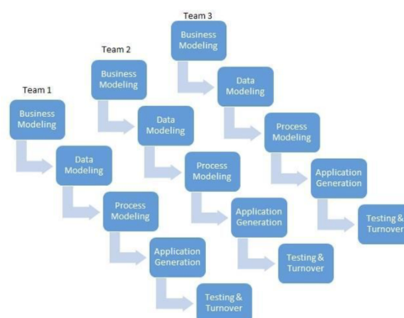
Obr. 5 Obecný model komponentového přístupu [11]

Komponentový přístup je zčásti podobný tradičnímu přístupu. Nicméně vývoj komponent, které mají být vhodné k přepoužití i v jiných systémech, je mnohem složitější, aby komponenty mohly fungovat v různých kontextech systému. [11]

2.1.2.4 Příklady modelů komponentového přístupu

Rapid Application Development Model (RAD)

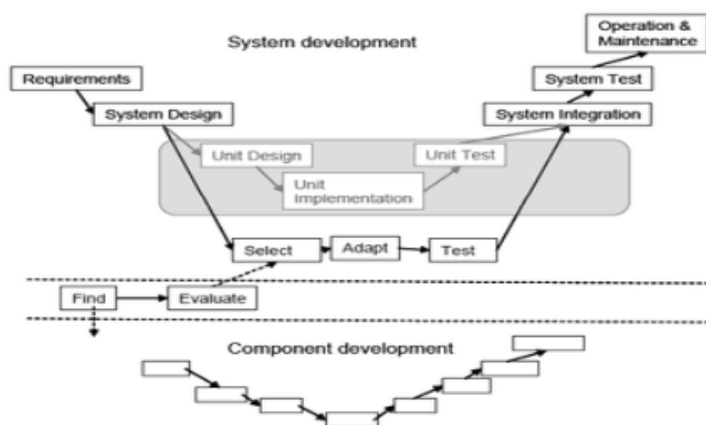
RAD model je založen na přístupu rychlého vývoje. Model má pár nevýhod, jako například oddělené týmy, které pracují na oddělených komponentách. Není vhodné použití tohoto modelu, pokud je vyšší technická náročnost a mohlo by docházet k technickým nerozuměním a nedostatkům mezi komponentami. RAD se využívá za předpokladu, že výsledný produkt může být složen z nezávislých komponent, které vyvíjejí oddělené týmy. Komponenty následně formují výsledný produkt. [5]



Obr. 6 Rapid Application Development model [5]

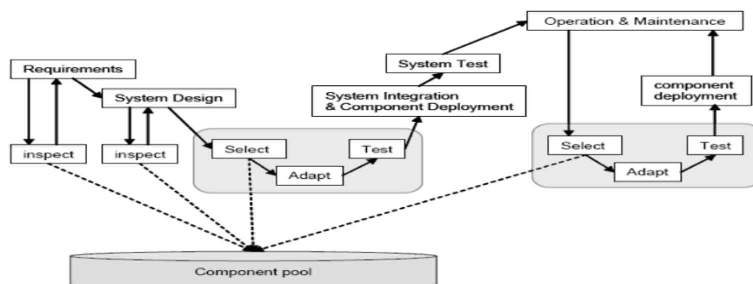
V-Model

Komponentový V-Model vychází z tradičního V-modelu, který přebírá principy od vodopádového modelu. V-Model se používá na velkých projektech s dlouhou životností. Tradiční přístup probíhá s unit návrhem, unit vývojem a unit testováním, které jsou velice časově náročné. Komponentový přístup namísto toho přidává fáze výběru, přizpůsobení a testování komponent s následnou integrací komponent do systému. V případě, že komponenty nejsou dostupné nebo dostatečně nesplňují požadavky a kompatibilitu se systémem, přidají se kroky hledání a hodnocení komponent. Pokud je komponenta nevyhovující, dojde k fázi vývoji komponenty, která je nezávislá na vývoji systému.



Obr. 7 V-Model 1 [15]

Tento ideální model často nenachází vhodné komponenty z důvodu, že neexistují hledané komponenty nebo nejsou s vyvíjeným systémem kompatibilní. Zároveň nezohledňuje fázi údržby systému v případě, že některé z komponent narušují požadavky celého systému. Více reálný V-Model je zobrazen v detailnějším vyobrazení obr. 8. [15]



Obr. 8 V-Model Detail [15]

Analýza požadavků požaduje znalost komponent, které jsou dostupné. Je vhodné s klientem vyjednat požadavky, které jsou splnitelné dostupnými komponentami, aby nebylo potřeba dodávat komponenty od třetích stran, popřípadě implementovat nové komponenty od nuly. [15]

Návrh systému podléhá stejné podmínce, jako analýza požadavků, a to jsou dostupné komponenty. Některé komponenty považované za vhodné z analýzy nemusí splňovat všechny technické požadavky a budou z návrhu systému vyloučeny a vyměněny za jiné komponenty, pokud jsou dostupné. Možnost kombinace komponent postavených na různých technologiích není tak reálná v praktickém využití z hlediska architektury a efektivity systému. [15]

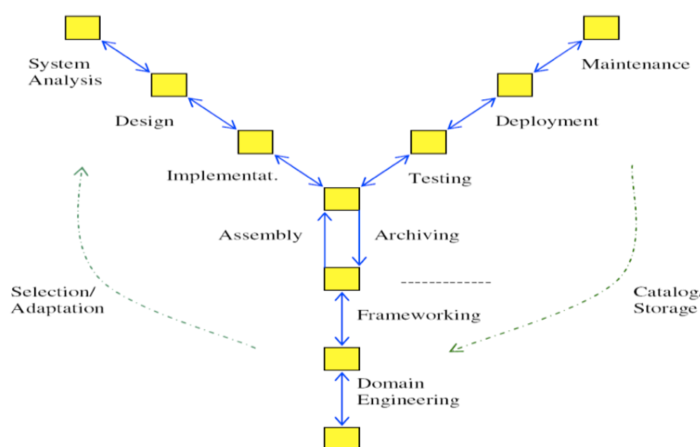
Integrační fáze systému se skládá z postavení infrastruktury pro komunikaci komponent, jako je komponenta pro databáze a frameworku systému pro správu a komunikaci komponent mezi sebou. Integrace konkrétní komponenty do systému se nazývá nasazení komponenty. [15]

Testování a ověření funkčnosti systému může být složitou částí vývoje, jelikož některé komponenty mohou být poskytnuty třetí stranou, takzvané black-box komponenty, takže vývojový tým nedokáže jednoduše opravit a identifikovat chybu v komponentě. Může nastat případ, že se chyba projeví na úplně jiném místě, i když chybu způsobuje komponenta. Proto je velmi důležité jasně a přesně definované rozhraní, vstupní a výstupní hodnoty u každé komponenty. [15]

Během údržby systému mohou být komponenty odstraněny, nahrazeny nebo jen pouze upraveny tak, aby byly splněny měnící se požadavky na funkčnost systému. Proces údržby má několik stejných kroků, jako integrace komponent. [15]

Y-Model

Y-Model je od základu navržen jako model pro komponentový přístup vývoje. Tento model se skládá z fází doménového inženýrství, frameworkingu, sestavování, archivace, systémové analýzy, návrhu systému, implementace, testování, nasazení systému a údržba. Tento model kombinuje úplně nové kroky doménového inženýrství, frameworking, sestavování a archivace s již používanými tradičními kroky vývoje software. Y-Model klade větší důraz na předpoklad přepoužitelnosti a opětovného použití komponent. [17]



Obr. 9 Y-Model [17]

Doménové inženýrství se zaměřuje na identifikaci komponent v dané doméně, neboli oboru. Příkladem může být bankovnictví. Opakující funkčností v této doméně může být například vytvoření účtu, kontrola zůstatku, výpis transakcí. Tento krok identifikuje opakující se funkčnosti pro více systémů ze stejné domény. Pokud je komponentový přístup naším cílem, doménové inženýrství by mělo být vždy provedeno v úvodu analýzy. [17]

Frameworking lze chápat jako proces vytvoření kostry, šablony nebo obecné struktury pro vývoj softwaru v dané doméně. To má zachytit sémantické vztahy mezi komponentami v dané doméně. Vždy je výhodnější připravit Framework, který bude kompatibilní s dostupnými komponentami, i když by se mohl framework ochudit o inovativní nápady, které by ale ovšem zapříčinily nutnost úpravy nebo dokonce vývoj nových komponent. Tudíž náklady na projekt by se razantně zvýšily. [17]

Sestavení systému se zaměřuje na výběr opakovaně použitelných komponent a frameworku z konkrétní domény. Produkt je složen z těchto prověřených a opakovaně použitelných prvků. [17]

Archivace má za cíle dobře zdokumentovat, kategorizovat a uložit nově vzniklé nebo upravené komponenty, aby byly jednoduše dohledatelné a srozumitelné pro opětovné použití v dalších systémech. Přepoužitelnost nemá pouze za cíl použít co nejvíce existujících komponent, ale zároveň vytvořit komponenty, které budou přepoužitelné v budoucnu. [17]

Systémové analýza je součást klasického postupu vývoje. Analytik se snaží identifikovat možné komponenty a rozpadnout systém na menší oddělené části. Identifikuje funkčnost systému a zachytí to v grafické nebo textové podobě. [17]

Návrh systému vytváří kostru pro následující fázi implementace, kdy jsou vybrány vhodné komponenty odpovídající požadavkům. [17]

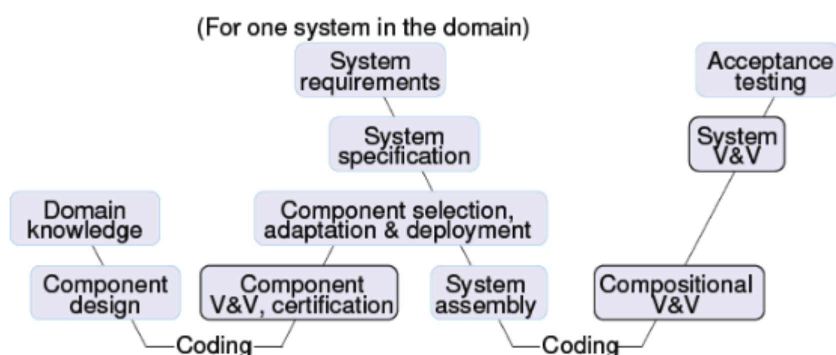
Návrh systému bude v kroku implementace přeložen do programovacího jazyka. Implementace vyžaduje definování datových struktur a algoritmů pro poskytnutí požadované funkčnosti systému. Nejjednodušší přístup je danou komponentu oddělit od celku a rozhodnout se, zda je jednodušší použít existující komponentu nebo ji vyvinout od začátku. Komponenta by měla být jednoduše použitelná a konfigurovatelná pro různé typy použití jak v black-box přístupu, tak i v možné modifikaci komponenty, tedy white-box přístupu. Vývoj opakovaně použitelných komponent stojí více úsilí a času, který lze pozorovat až po delší době při zásobě většího počtu komponent, které lze rychle přepoužívat. [17]

Testování probíhá po sestavení a implementaci komponent. Testování lze rozdělit do dvou částí. Testování funkčnosti samostatných komponent a testování celého systému, jako celku. Testování samostatné komponenty probíhá tak, že se ověří chování samotné komponenty s definovaným vstupem a zda její výstup odpovídá definovaným požadavkům. Lze použít různé metody jako white-box, black-box, kontrola kódu, formální důkazy apod. Po dokončení všech jednotlivých testů je proveden integrační test, zda komponenty spolu komunikují a fungují, jak je uvedeno ve specifikaci. Mohou být provedeny další metody testování, jako například výkonnostní test, akceptační test, test

kvality, bezpečnostní test a test finální instalace. Výsledkem je funkční systém připravený k nasazení. [17]

W-Model

W-Model kombinuje dva V-Modely. Jeden pro vývoj komponent a druhý je zaměřený na vývoje systému. Životní cyklus komponenty se skládá ze dvou fází, návrh komponent a nasazení komponent, a je přizpůsoben kontextu domény. Ve fázi návrhu komponenty jsou komponenty identifikovány, navrženy a implementovány dle požadavků a jsou uloženy do seznamu dostupných komponent. Dostupné komponenty jsou vhodné pro celou doménu, pro kterou byly navrženy a nejsou výlučně vhodné pouze pro jeden systém. Ve fázi nasazení se komponenty implementují do určitého systému, který je ve vývoji. Životní cyklus systému se také liší, kde místo všech fází vývoje komponenty je pouze krok nasazení komponenty. Životní cyklus systému se spojuje s životním cyklem komponenty v iteracích, jelikož jednotlivé komponenty jsou nasazovány do systému jedna po druhé. [16]



Obr. 10 Model-W [16]

2.2 Jakost softwarového produktu

Jakost softwarového produktu vymezuje ukazatele, které určují míru uspokojení potřeb uživatelů při užívání daného produktu za předem stanovených podmínek. Uživatelé mohou mít odlišné potřeby, a právě jejich potřeby ovlivňuji vnímanou jakost softwarového produktu z pohledu každého uživatele. Jakost je tedy potřeba brát jako relativní pojem. [18]

Jelikož jakost může znamenat pro každého trochu něco jiného, byl stanoven model jakosti. Model jakosti se skládá ze základních šesti charakteristik jakosti softwaru, aby bylo možné jakost definovat [18]

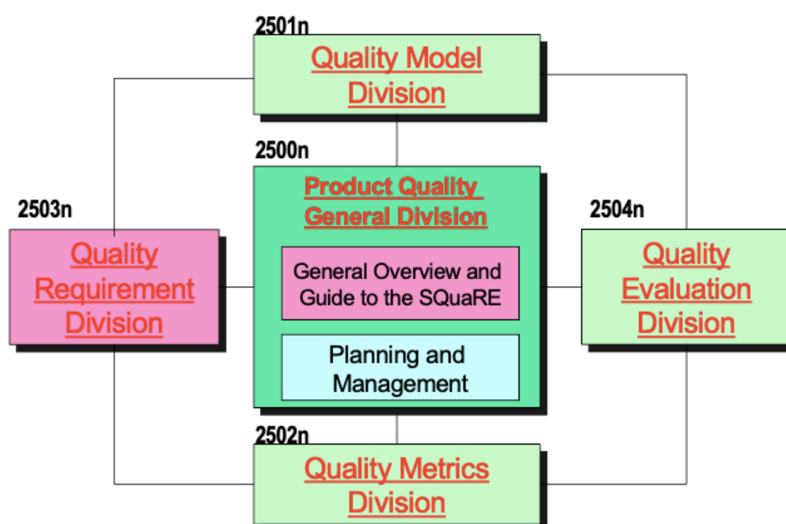
- **Funkčnost** – schopnost zajistit ze stanovených podmínek funkce systému, kdy se hodnotí, zda funkce jsou splněny nebo ne
- **Bezporuchovost** – schopnost zajistit požadovanou úroveň výkonu při použití softwarového produktu definovaným způsobem
- **Použitelnost** – definuje vynaloženou míru úsilí uživatelem, aby byl schopen softwarový produkt použít stanoveným způsobem
- **Účinnost** – schopnost poskytnout výkon ve srovnání s použitými zdroji za stanovených podmínek, jako jsou nároky na čas nebo rozsah paměti
- **Udržovatelnost** – schopnost softwarového produktu být opraven nebo měněn na základě zjištěných nedostatků nebo nových požadavků
- **Přenositelnost** – schopnost softwarového produktu být používán v různých prostředích, jako je operační systém, hardwarová platforma nebo spolupráce s jinými softwarovými produkty

2.2.1 ISO/IEC 250xx - SQuaRE

Standart ISO/IEC 250xx také nazývaný SQuaRE je soustava norem zabývající se jakostí softwarového produktu. Utřídila již existující standarty ISO/IEC 9126, ISO/IEC 14598 , ISO/IEC 12119 a doplnila jejich nedostatky, jako jsou předpisy formulací požadavků a obecné zásady teorie měření a tvorby metrik. Název SQuaRE je zkratkou slov Software Quality Requirements and Evaluation. [18]

- **2500n - Obecná část**
    - **25000 - Obecný přehled a průvodce po SQuaRE** – terminologie, architektonický model a přehled dokumentů pro uživatele normy
    - **25001 – Plánování a management** – požadavky a pokyny pro podporu řízení softwarové specifikace požadavků na produkt a hodnocení

- **25010 – Model jakosti** – model interní a externí kvality softwarového produktu a jeho použití. Obsahuje charakteristiky a podcharakteristiky.
- **2502n – Metriky pro jakost [19]**
    - **25020 – Referenční model a průvodce metrikami** – referenční model a definice pro měření jakosti produktu, dat a jakosti užití
    - **25021 – Prvky měření kvality** – základy a míry pro měření během životního cyklu vývoje software
    - **25022 – Metriky pro jakost při použití**
    - **25023 – Měření kvality systému a softwarového produktu**
      25024 – Metriky jakosti dat
- **25030 – Požadavky na jakost** – doporučení pro proces definující požadavky na kvalitu
- **2504n – Hodnocení jakosti**
    - **25040 – Přehled o procesech hodnocení**
    - **25041 – Postup vývojáře**
    - **25042 – Postup akvizitéra**
    - **25043 – Postup hodnotitele**



Obr. 11 Struktura SQuaRE [18]

2.2.1.1 25023 – Měření kvality systému a softwarového produktu

Norma ISO / IEC 25023 určuje metriky kvality pro kvalitativní měření softwarového produktu z pohledu jednotlivých vlastností popsaných v ISO/IEC 25010. Je určena k použití společně s ostatními normami ze skupiny ISO / IEC 2502n. Norma ISO / IEC 25023 vysvětluje, jak aplikovat jednotlivá měření na softwarový produkt a sadu metrik pro každou charakteristiku. Metriky jsou rozděleny dle základních charakteristik jakosti software. Metriky funkčnosti, výkonnosti / účinnosti, kompatibility, použitelnosti, spolehlivosti, bezpečnosti, udržovatelnosti a přenositelnosti. [20]

Z pohledu tématu této práce má význam se zaměřit na vybrané metriky z oblasti udržovatelnosti a přenositelnosti.

Tabulka 3 Vybrané metriky ISO / EIC 25023 [20]

ID	Jméno	Popisek	Funkce
MMo-1-G	Spojení komponent	Jak jsou komponenty nezávislé na ostatních komponentách a na změnách v nich.	X = A / B A = Počet komponent, které jsou implementovány bez vlivu na jiné komponenty B = Počet komponent celkově
MRe-1-G	Přepoužitelnost prvků	Kolik prvků je přepoužitelných.	X = A / B A = Počet prvků které jsou navrženy k přepoužití. B = Počet prvků celkově
MMd-1-G	Účinnost úpravy	Jak efektivně jsou implementovány úpravy v porovnání s očekávaným časem.	$X = \sum_{i=1\ do\ n} (A_i/B_i)\ /\ n$ A = Celkový čas vynaložený na úpravu B = Očekávaný čas vynaložený na úpravu N = Počet měřených úprav

PRe-1-G	Podobnost použití	Jaký poměr funkcionalit nahrazeného produktu může být implementovaný bez jakékoli úpravy.	X = A / B A = Počet funkcionalit vhodných k implementaci bez nutných úpravy B = Počet přenesených funkcionalit

2.2.2 Další metriky

LOC – Lines of code

Lines of code neboli počet řádků je nejznámější metrika. LOC se nejčastěji používá k určení velikosti projektu a s tím souvicející i jeho složitostí. Počtem řádků se myslí počet řádků zdrojového kodu SLOC, nikoli počet řádků strojového kódu. Počty řádků mohou, ale nemusí započítávat prázdné nebo zakomentované řádky. Problém této metriky může být v tom, že kód může být napsaný na více i méně řádků a záleží pouze na programátorovi a jeho schopnostech, jakým způsobem implemetuje kód. [21]

CYCLO - McCabova cyklomatická složitost

McCabovu cyklomatickou složitost popsal v roce 1976 TJ.McCabe. V době, kdy byla metoda navržena, byla snaha udržet rozsah kódu do 50 řádků kvůli testovatelnosti a udržitelnosti. McCabe uvedl příklad, kdy 25 navazujících podmínek konstrukce if-then může vytvářet až 33,5 milionů nezávislých programových cest. To zapříčiní nemožnost testovat všechny potencionální scénáře. McCabe proto navrhl metodu měření cyklomatické složitosti, která byla původně zamýšlena pro měření testovatelnosti kódu. [22]

V teorii grafů existuje pojem cyklomatické číslo. Toto číslo reprezentuje počet nezávislých kružnic v grafu. V oblasti softwaru cyklomatická složitost znázorňuje počet lineárně nezávislých cest, kudy program může téct. V případě že testovaný kód bude mít jeden vstupní a jeden výstupní bod, cyklomatická složitost se bude počítat následovně:

$$V(G)=e-n+2*p$$

kde

V (G) - cyklomatické číslo

N - počet uzlů, kde uzel grafu představuje neoddělitelnou skupinu příkazů, příkladem může být tělo cyklu

e – počet hran, kde orientace hran představuje, jak a v jakém pořadí se budou jednotlivé skupiny příkazů provádět

p – počet napojených komponent neboli také počet koncových uzlů

Cyklomatickou složitost lze také chápat jako počet binárních podmínek + 1. V případě, že existují jiné než binární podmínky, třícestné a vícecestné, tak třícestná podmínka se spočítá jako 2 binární podmínky. V případě n-cestných podmínek se počítá n-1 binárních podmínek. Iterace cyklu se počítá jako jedna binární podmínka. [22]

Z důvodu složitosti výpočtu existují automatizované nástroje pro výpočet cyklomatické složitosti nad zdrojovým kódem.

2.3 React Native

React Native vychází z frameworku ReactJS, který byl vytovřen společností Facebook pro tvorbu aplikací s webovým rozhraním. React Native stejně jako ReactJs je JavaScriptový framework pro nativní mobilní aplikace platformy iOS a Android. React Native byl představen v roce 2015 a stal se rychle široce používaným frameworkem u mnoha známých společností. React Native používá JavaScriptovou syntaxi JSX, která využívá API nativních komponent na cílových platformách. [23]

Výhody

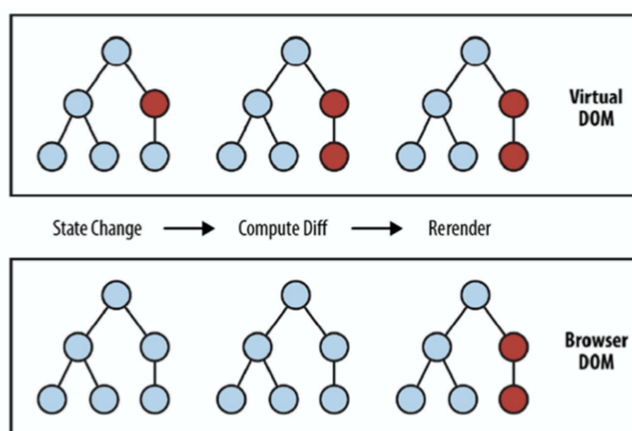
- Aplikace vypadá jako nativní aplikace
- Stejný kód pro obě platformy – snížené náklady
- Velice podobné s ReactJS
- Hot Realoading
- Úprava v JavaScriptu není kontrolována v obchodech aplikací Applu a Google
- Lze psát nativní kód [23]

Nevýhody

- Opoždění podpory nových nativních API
- Zhoršený debbug z důvodu React Native vrstvy [23]

Virtuální DOM

DOM (Document Object Model) je aplikační rozhraní, díky kterému lze přistupovat k objektům XML dokumentu. DOM má stromovou strukturu a lze jej upravovat na přesně určeném místě. React Native využívá koncept virtuálního DOM. Uživatelské rozhraní je nahrané v paměti a synchronizované se skutečným DOMem. React Native vyhodnotí změny DOMu nahraného v paměti a poté překreslí elementy ve skutečném DOMu, kterých se změna týká. [24]



Obr. 12 Výpočet virtuálního DOM a vykreslení [24]

React Native Bridge

React Native má vrstvu mezi kódem a cílovou platformou, kterou modifikuje bridge. Bridge zajišťuje volání nativních API cílové platformy ze zdrojového kódu v React Native. Pro platformu iOS volá API v Objective-C nebo Switft a pro platformu Android volá API v Java nebo Kotlin. React Native volá asynchronně API cílové platformy, což zajišťuje nativní pocit uživatele z aplikace. [24]

JSX

JSX neboli JavaScript XML umožní do JavaScript zdrojového kódu vložit XML značky tak, aby se jednalo o validní kód. JSX se využívá v ReactJS a React Native s tím rozdílem, že ReactJS používá HTML značky a React Native ne. JSX soubory mají koncovku „.jsx“. Pro volání JavaScript zdrojového kódu v XML značkách se používají složené závorky. [23]

2.3.1 Komponenty

Komponenta v React Native je samostatný soběstačný zdrojový kód, který má stav, atributy a metody. Komponenty v React Native zapouzdřují vizuální elementy. Každá komponenta vždy musí implementovat metodu render(). Metoda render vrací jediný element, který zastupuje danou komponentu a může obsahovat další podkomponenty nebo elementy. Každá komponenta používá data typu State a Props:

State

State je speciální objekt každé komponenty, který v sobě drží vnitřní stav komponenty. State má proměnlivé data a mění se metodou setState(). Tedy State se může průběžně měnit v průběhu životního cyklu komponenty. Inicializace Statu není povinná. [23]

Props

Props je speciální objekt každé komponenty, který v sobě drží vlastnosti dané komponenty. Props jsou neměnná data, která jsou jednosměrně získávána od nadřazené komponenty. Komunikace je pouze jednosměrná – rodič => potomek. Props nelze uvnitř komponenty měnit a jsou to data pouze ke čtení. Do Props lze vložit primitivní data, celé elementy nebo funkce. Props se volají jako XML atributy a uvnitř komponenty se volají pomocí tečkové notace. [23]

Stylování

Stylování v React Native je velice podobné stylování kaskádových stylů pro webové stránky s tím rozdílem, že se nepoužívá pomlčková konvence, ale velbloudí konvence pro názvy vlastností stylů. Pro rozložení prvků se hlavně využívá flexbox a téměř vůbec se nepoužívá pixelové nebo procentuální rozložení. Každý objekt má Props style, do které se přímo inline může vkládat styl nebo odkázat referenci objektu typu StyleSheet. JavaScriptové objekty typu StyleSheet se vytváří metodou create(), díky které můžeme vytvořit více stylů v jednom objektu. Potom je tedy možné odkazovat referenci stylu pomocí tečkové notace namísto inline stylu, což výrazně zjednodušuje a zpřehledňuje práci a zkracuje kód. [23]

Funkcionální komponenty

Komponenty lze vytvářet dvěma způsoby. Klasické třídní komponenty nebo takzvané funkcionální komponenty, což je novější a hojně používaný přístup. Vlastnosti komponenty Props se do komponenty předávají jako parametry funkce a komponenta vrací kořenový element z komponenty metodou render(). Výhodou funkcionálních komponent je kratší a přehlednější kód. S verzí React 16.8. funkcionální komponenty již nejsou bezstavové a lze měnit jejich vnitřní stav pomocí takzvaných Hooks. [25]

React Hooks

React Hooks ve verzi React 16.8. vznikly za účelem sdílení logiky mezi komponentami a vylepšení práce s životním cyklem komponenty. Hooks řeší problém, kdy původně bylo možné pracovat s vnitřním stavem komponenty s třídními komponentami, nikoli s funkcionálními komponentami. Hooks umožňují funkcionálním komponentám používat State komponenty a reagovat na životní cyklus komponenty. React Hooks lze použít pouze ve funkcionálních komponentách a nelze je využívat uvnitř cyklů, vnořených funkcích nebo podmínkách. [26]

3 Vlastní práce

Tato část je zaměřena na identifikaci frontendových komponent, implementaci komponent a sestavení funkčního interface aplikace z implementovaných komponent. Z pohledu komponentového přístupu vývoje je zvolen Y-model komponentového procesu vývoje, podle kterého se postupuje v jednotlivých fázích tak, aby byly zajištěny všechny potenciální výhody komponentového přístupu nejen z implementačního a vývojového pohledu, ale i z pohledu procesního, které byly popsány v teoretické části.

Po implementaci a finálním otestování funkčnosti aplikace složené z frontendových komponent lze provést analýzu jakosti komponentového přístupu v porovnání s vybranou částí aplikace, která byla původně implementována klasickým přístupem bez komponent.

3.1 Vývoj a implementace komponent

Y-model uvažuje nejen vývoj od nuly, ale i interně vyvinuté existující komponenty nebo komponenty třetích stran. V této případové studii se jedná kompletně o nový vývoj od nuly. Frontendové komponenty, které vyplynou z analýzy, jsou tak specifické pro tuto aplikaci, že nepřipadalo v úvahu použít komponenty třetích stran. Tento fakt se bere jako východisko na začátku vývoje a je tomu přizpůsoben vývojový proces.

3.1.1 Doménové inženýrští

Aplikace, pro kterou je vytvořena frontendová komponentová knihovna, je univerzitní mobilní aplikace pro studentky ČZU. Hlavním cílem mobilní aplikace je jednoduše a efektivně poskytnout důležité funkce pro studentský život v kampusu univerzity během studia. Propojuje univerzitní systémy do jednoho místa, které má uživatel stále na dosah „v kapse“. Nyní má student přístup do univerzitních systémů z různých míst, což není uživatelsky přívětivé. Student se stále musí přepínat mezi jednotlivými systémy. Tento problém řeší tato studentská aplikace.

Interface aplikace je navržena tak, aby umožnila jednoduchý rozvoj a integraci nových funkcionalit v budoucím rozvoji. Nové funkce bude jednoduché přidat, aniž by bylo nutné od základu předělávat interface aplikace. To je hlavním motivem vytvoření komponentové knihovny, která zjednoduší a zefektivní budoucí vývoj.

Grafický návrh mobilní aplikace je navržen vhodně pro komponentový přístup. Během návrhu se počítalo s maximálním navržením přepoužitelných grafických prvků, aby bylo možné ve fázi implementace založit vývoj na přepoužitelných komponentách. Fáze návrhu interface je základní a důležitou částí komponentového přístupu vývoje software. Mobilní aplikace s takto navrženým interface je vhodná ke splnění cílů této práce, tedy implementovat aplikaci a splnit kritéria komponentového přístupu za účelem výhod komponentového přístupu.

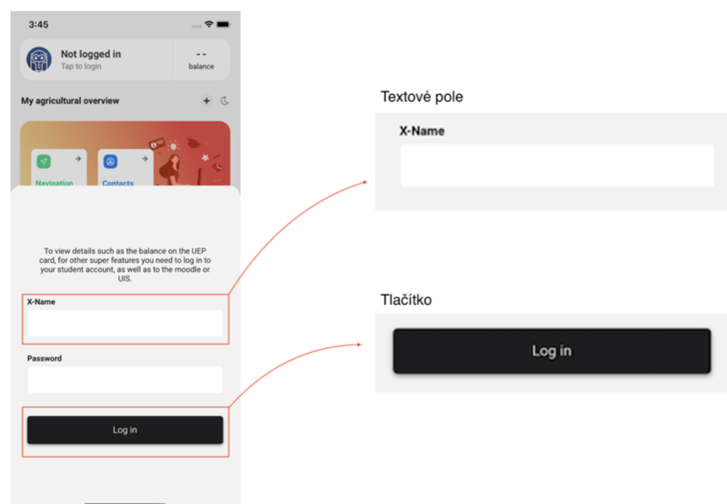
Hlavními funkcemi aplikace jsou zobrazení aktuálních jídelníčků ve stravovacích zařízeních v kampusu, mapa pro orientaci v kampusu, informační zdroj o kampusu a životě na univerzitě pro jednoduché zorientování pro nové studenty, zobrazení aktuálních novinek a událostí univerzity, zobrazení pracovních nabídek a možnost přihlášení ke studentskému účtu.

Z pohledu doménového inženýrství lze uvažovat některé oblasti jako opakující se v doméně aplikací. Například výpis pracovních nabídek, výpis novinek a událostí na univerzitě nebo výpis jídelníčků by mohl být vhodný pro použití již hotových implementací z frontendového pohledu, pokud takové komponenty jsou dostupné pro přepoužití v úložišti komponent nebo dostupné od třetí strany.

3.1.2 Systémová analýza

V systémové analýze z pohledu komponentového přístupu se rozpadne systém aplikace do oddělených funkčních celků za účelem identifikace a doporučení, které frontendové prvky jsou vhodné k implementaci, jako samostatné komponenty a které naopak je zbytečné implementovat komponentově, protože se nejeví v budoucnu jako přepoužitelné. Implementace nepotřebných komponent by bylo neefektivní, zbytečně by se tím vyčerpaly zdroje a aplikace by byla složena ze zbytečného množství komponent.

Přihlašovací obrazovka



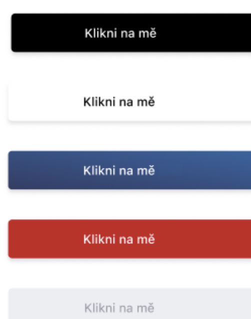
Obr. 13 Přihlašovací obrazovka (zdroj: vlastní zpracování)

Přihlašovací obrazovka slouží pro přihlášení v aplikaci ke studentskému účtu s přihlašovacími údaji, které studenti využívají napříč všemi univerzitními systémy. Tudíž není potřebná obrazovka pro uživatelskou registraci. Aplikaci je možné využívat i bez přihlášení ke studentskému účtu. Funkčnosti, pro které je nutné studentské přihlášení, si přihlášení automaticky vynutí, aby bylo možné funkčnost využít. V opačném případě funkčnost zůstane nedostupná. Z tohoto důvodu je přihlašovací obrazovka navržena jako modální dialog, který vždy při potřebě přihlášení vyjede ze spodu obrazovky a umožní uživateli se přihlásit.

Na přihlašovací obrazovce se vyskytují dva základní elementy, které jsou jednoznačně vhodné k implementaci jako komponenty.

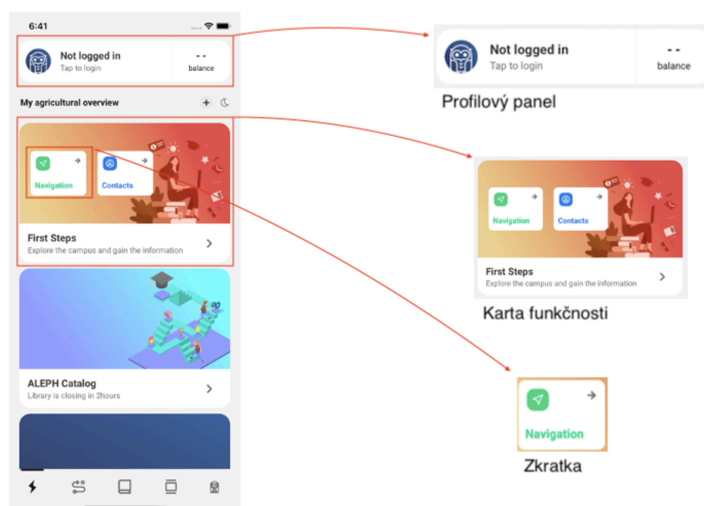
Prvním elementem je textové pole, které je dostupné již jako nativní komponenta v knihovně React Native, ale za účelem designové modifikace je vhodné tento prvek definovat jako komponentu, protože ji lze jednoduše upravovat a definovat validace vstupu z jednoho místa, namísto kopírovaní stylování a validací do oddělených částí aplikace. V případě nutnosti úpravy by bylo potřeba myslet na všechna místa aplikace, kde je tento element použit. Z důvodu širokého využití v různých obrazovkách s odlišným rozložením prvků je doporučené, aby komponenta textového vstupu podporovala různé velikosti zobrazení.

Druhý element je tlačítko, u kterého se stejně jako u textového vstupu dá očekávat široké využití napříč celou aplikací. Obdobně jako u textového vstupu se doporučuje implementovat tlačítko jako komponentu. Kvůli specifickému stylu definovaném na jednom místě a definování různých stylů a velikostí viz obr 14.



Obr. 14 Styly tlačítek (zdroj: vlastní zpracování)

Domovská obrazovka



Obr. 15 Domovská obrazovka (zdroj: vlastní zpracování)

Domácí obrazovka slouží jako základní obrazovka po otevření aplikace pro základní orientaci mezi všemi funkčnostmi v aplikaci. Aplikace má hlavní orientační prvek spodní TabBar. V TabBaru jsou umístěny nejdůležitější funkčnosti, a proto jsou vždy

dostupné uživateli. Domácí obrazovka obsahuje karty, které odkazují na všechny funkčnosti aplikace včetně obsažených ve spodním TabBaru.

V horní části obrazovky je profilový panel uživatele, který je závislý na tom, zda se uživatel přihlásil ke studentskému účtu nebo ne. Tento profilový panel se může opakovat napříč aplikací ve více obrazovkách, a proto je vhodné ho implementovat jako komponentu. Stisknutím tohoto panelu se otevírá přihlašovací okno v modálním dialogu.

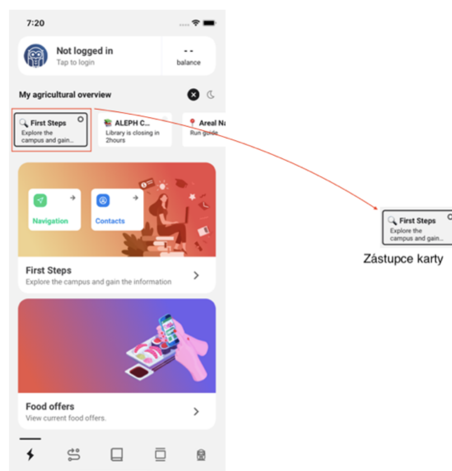
Hlavní částí domácí obrazovky jsou karty funkčností, které odkazují na jednotlivé funkčnosti aplikace. Karta má dva různé způsoby odkazování.

1. Funkčnost je zároveň ve spodním TabBaru a dotykem karty se přepne vybraná položka v TabBaru
2. Funkčnost není uvedena ve spodním TabBaru a dojde k otevření nové obrazovky, ale položka TabBaru se nemění

Z důvodu opakovaného použití nejen na domácí obrazovce je vhodné kartu implementovat jako komponentu.

Každá karta může mít zrychlené volby, takzvané zkratky, které urychlí použití aplikace uživateli a odkazují na specifickou funkčnost uvnitř nově otevřené obrazovky. Dvě zkratky jsou vykresleny na první kartě. Tyto elementy zkratek jsou také vhodné implementovat jako komponentu, jelikož se jejich použití může opakovat napříč aplikací nejen na domovské obrazovce.

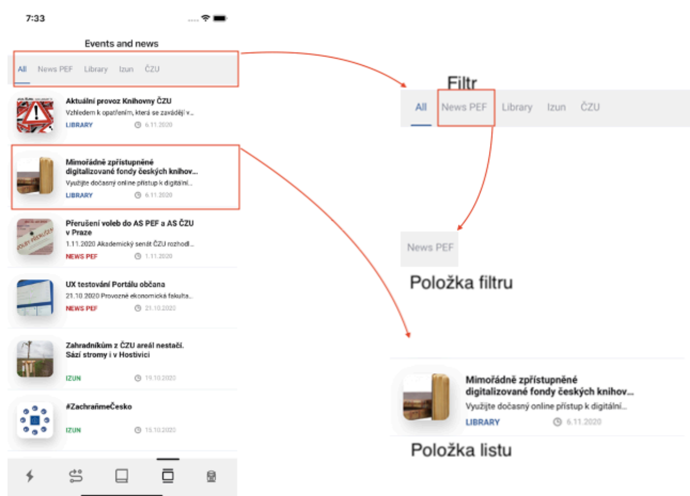
Z technického důvodu je zkratky nutné implementovat jako dvě komponenty. První komponenta zajišťuje zobrazení zkratek v kartě jako celek. Druhá komponenta představuje samostatné položky zkratek, které se mohou objevit v jedné kartě vícekrát.



Obr. 16 Domácí obrazovka se skrytým menu (zdroj: vlastní zpracování)

Karty se dají zobrazovat a skrývat v menu, které je schované pod ikonkou „+“. Jedná se o vertikální menu, ve kterém vybráním zástupce karty určenou kartu zobrazíme nebo skryjeme. Tímto způsobem si uživatel jednoduše upraví domácí obrazovku podle svých potřeb a nebude mu tam překážet karta funkčnosti, kterou nepotřebuje. Zástupci karet v menu jsou znovu opakující se prvky, které je vhodné implementovat jako komponenty.

Obrazovka novinek a událostí



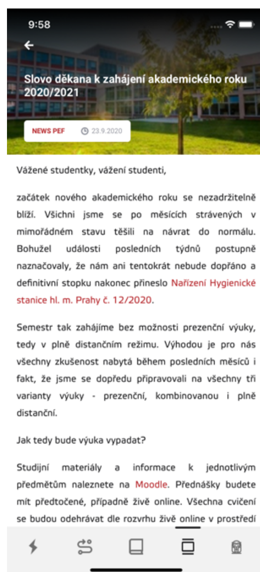
Obr. 17 Obrazovka novinek a událostí (zdroj: vlastní zpracování)

Obrazovka novinek a událostí je jedna z hlavních funkčností aplikace, a proto má položku ve spodním TabBar. Po vybrání karty na domácí obrazovce se přepne TabBar na položku novinky a události. Obrazovka novinek a událostí nevyžaduje přihlášení ke studentskému účtu a lze její funkčnost využít plně bez přihlášení.

V horní části obrazovky je vykresleno menu, které slouží jako filtr položek dle kategorie. Automaticky je vybrána obecná kategorie „Vše“ bez filtrování. Po vybrání kategorie se zaflirtují položky v listu. Celé menu s kategoriemi je vhodné implementovat jako komponentu, jelikož se hojně využije i na jiných obrazovkách.

Z technického hlediska je výhodné implementovat položku menu také jako komponentu, jelikož se opakuje v menu.

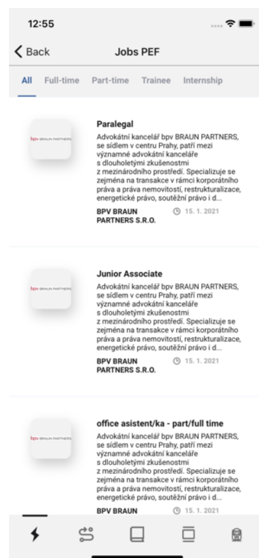
Každá položka v listu novinek reprezentuje jednu novinku nebo událost. Po vybrání položky se otevře nová obrazovka s detailem položky. Položka listu jako celek je vhodná k implementaci jako komponenta, jelikož se opakuje nesčetněkrát na této obrazovce. Položka obsahuje obrázek, nadpis, popis, kategorii a datum.



Obr. 18 Obrazovka detail novinky (zdroj: vlastní zpracování)

Na obrazovce detail novinky se nevyskytuje žádný frontendový prvek, který měl být nutně implementován jako samostatná komponenta. Teoreticky by bylo možné implementovat položku s kategorií a datem novinky jako komponentu, ale z aktuálně plánované funkčnosti nevyplývá, že by se tento prvek mohl někde opakovat.

Obrazovka pracovních nabídek



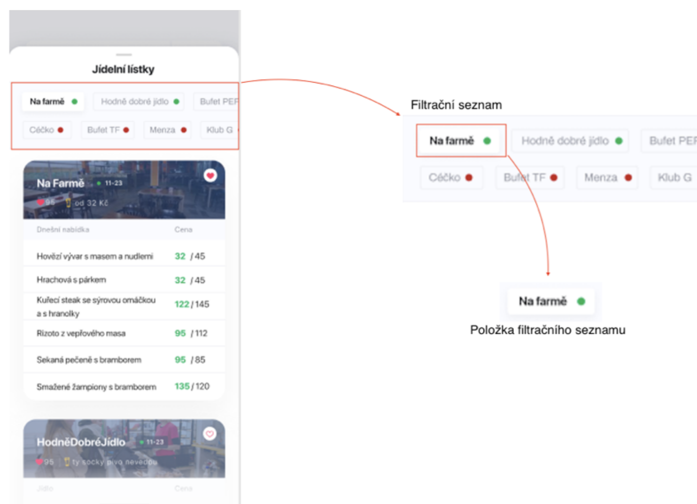
Obr. 19 Obrazovka pracovní nabídky (zdroj: vlastní zpracování)

Obrazovka pracovních nabídek není hlavní funkčností mobilní aplikace, a proto nemá svoji položku ve spodním TabBaru aplikace. Pracovní nabídky lze otevřít z karty na domovské obrazovce. Pro zobrazení pracovních nabídek není nutné uživatelské přihlášení.

Jsou zde přepoužity všechny komponenty z obrazovky novinek a událostí. Jedná se o komponenty filtračního menu, položky menu a položky listu nabídek. Není nutné implementovat další komponenty.

Po vybrání položky pracovní nabídky aplikace otevře v nativním prohlížeči adresu webové stránky, kde se zobrazí detail pracovní nabídky.

Obrazovka jídelníčků



Obr. 20 Obrazovka jídelníčků (zdroj: vlastní zpracování)

Obrazovka jídelníčků se neřadí mezí hlavní funkčnosti aplikace, a proto nemá položku ve spodním TabBaru aplikace. Jediným místem, odkud lze obrazovku jídelníčků otevřít, je karta jídelníčků na domovské stránce. Obrazovka se otevře jako modální dialog ze spodní části obrazovky podobně jako přihlašovací obrazovka. To evokuje uživatelský pocit, že obrazovka není hlavní funkčností, ale je to jen funkčnost pro rychlé zjištění informací. Informace jsou o aktuálně nabízených jídlech ve stravovacích místech v kampusu univerzity včetně cen jídel se studentskou slevou i bez studentské slevy. Pro obrazovku jídelníčků není nutné přihlášení ke studentskému účtu a je plně funkční pro každého uživatele.

V horní části obrazovky lze najít filtrační seznam stravovacích míst, ve kterém lze dynamicky vybírat pouze požadované položky. Celý toto filtrační seznam je vhodné implementovat jako komponentu z důvodu možného přepoužití v jiných částech aplikace.

Z technického důvodu je vhodné položky filtračního seznamu také implementovat jako samostatnou komponentu. Položka ve filtračním seznamu se opakuje několikrát, a proto je také vhodné ji implementovat jako komponentu.

Naopak položky, které reprezentují jednotlivá stravovací místa a poskytují seznam aktuálně nabízených jídel s cenami, není nutné implementovat jako samostatnou komponentu. Jídelníček je natolik specifický, že není pravděpodobné jeho opakované použití v jiné části aplikace.

3.1.3 Návrh systému

Z analýzy funkčností aplikace byly navrženy k implementaci vhodné komponenty, které již v některých částech aplikace budou moci být rovnou přepoužity, a tím se urychlí a zjednoduší fáze implementace. Zároveň byly identifikovány některé komponenty, které aktuálně nebudou přepoužity, ale z jejich povahy je vysoce pravděpodobné, že v budoucím rozvoji a s přidáváním nových funkčností budou moci být přepoužity. Jak bylo doporučeno v analýze, tyto frontendové části budou také implementovány komponentově a připraveny pro budoucí rozvoj.

Textové pole

Textové pole je základní prvek pro textové vstupy, kde uživatel zadává textová data do aplikace. Hlavní motivací implementace textového pole je specifická grafická podoba textového pole odpovídající navrženému designu, který je jednotný napříč celou aplikací.

Příkladem použití komponenty je přihlašovací obrazovka, kde uživatel zadává přihlašovací jméno a heslo. Textové pole má následující vlastnosti:

- Nadpis textového pole
- Inicializační vstupní hodnota, která může být předvyplněná
- Možnost skrytí znaků, například pro heslo
- Návratová hodnota vstupu pro formulář

Tlačítko

Tlačítko je základní prvek pro potvrzení uživatelské akce. Hlavním důvodem implementace tlačítka komponentově je grafická podoba tlačítka z grafického návrhu. Tlačítko se využije v různých návrzích obrazovek, a proto komponenta potřebuje definovat různě barvené návrhy a různé velikosti tlačítka.

Příkladem použití tlačítka je použití tlačítka pro potvrzení uživatelského přihlášení na přihlašovací obrazovce. Tlačítko má následující vlastnosti:

- Titulek, kde ke každému schématu tlačítka patří definovaná barva titulku tlačítka
- Barevné schéma, kde je definováno pět typů barevných schémat

    - Černé, bílé, modré, červené, šedé
- Velikost, kde jsou definovány tři typy velikostí
    - Velké, středí, malé
- Aktivita tlačítka, kdy ke stavu disabled patří šedivé schéma tlačítka
    - Enabled, disabled

Karta profilu uživatele

Karta profilu uživatele je prvek, kde se předává uživateli informace, zda je přihlášen k uživatelskému účtu, nebo je odhlášený. Tento stavový prvek se předpokládá využít na více místech aplikace. Příkladem využití je domovská stránka, která je výchozí obrazovkou pro celou aplikaci. Po kliknutí na kartu profilu uživatele se nabízí přihlašovací okno v modálním dialogu.

Karta profilu uživatele má následující vlastnosti:

- Ikonka karty s defaultní ikonkou pro nepřihlášeného uživatele nebo s fotkou uživatele po přihlášení
- Popisek, kde pro nepřihlášeného uživatele je vysvětlení akce pro přihlášení a pro přihlášeného uživatele je napsán status / studijní obor / ročník uživatele
- Zůstatek, kde je napsán aktuální peněžní zůstatek na studentském účtu UEP a měna zůstatku
- Popisek zůstatku, který je určen pro popis částky

Karta funkčnosti

Karta funkčnosti je základním prvkem domovské stránky, která spouští jednotlivé funkčnosti po kliku na kartu. Karta funkčnosti, která je plánovaná pro využití v jiných částech aplikace, představuje základní grafický prvek aplikace, a proto je nutné ji implementovat jako komponentu.

Karta funkčnosti má následující vlastnosti:

- Titulek karty
- Popisek karty, který není povinný
- Cestu k obrázku na pozadí

- Cílovou obrazovku po kliku uživatelem
- Stav k otevření nové obrazovky nebo pouze přepnutí TabBaru
- Pole objektů možných zkratek k vykreslení na kartě, které nejsou povinné

Sekce zkratek v kartě funkčnosti

Sekce zkratek v kartě funkčnosti je komponentou, která zapouzdřuje položky zkratek, které je možné vykreslit na kartě.

Sekce zkratek má následující vlastnosti:

- Pole objektů zkratek

Položka zkratky v sekci zkratek

Položka zkratky v sekci zkratek představuje položku zkratky, která odkazuje na specifickou obrazovku v rámci dané funkčnosti.

Položka zkratky v sekci zkratek má následující vlastnosti:

- Barva textu
- Ikona
- Popisek
- Odkaz na funkčnost nebo externí odkaz na webu

Zástupce karty funkčnosti

Zástupce karty funkčnosti je položkou v menu nastavení domovské stránky, která přidává nebo odebírá karty funkčností z domovské stránky tak, jak si chce uživatel přizpůsobit domovskou stránku. Zástupce karty je vhodné implementovat jako komponentu, z důvodu možného přepoužití v jiných místech aplikace.

Zástupce karty funkčnosti má následující vlastnosti:

- Titulek
- Popisek
- Ikonu
- Stav vybraný / nevybraný
- Návratovou hodnotu pro akci domovské stránky

Filtrační menu

Filtrační menu je základním prvkem pro výběr dle kategorie v listu v aplikaci. Představuje komponentu, která zapouzdřuje položky ve filtračním menu. Příkladem použití je obrazovka novinek nebo obrazovka pracovních nabídek. Použití na více obrazovkách jasně ukazuje výhodu implementace jako komponenty.

Filtrační menu má následující vlastnosti:

- Pole objektů kategorií
- Návratovou hodnotu pro akci filtrace

Položka filtračního menu

Položka filtračního menu představuje jednu kategorii v menu. Je vhodné ji implementovat komponentově z technického důvodu funkčnosti filtračního menu. Vybráním položky ve filtračním menu se v návratové hodnotě vrací identifikátor kategorie pro akci filtrace.

Položka filtračního menu má následující vlastnosti:

- Jedinečný identifikátor položky
- Titulek položky
- Hodnotu položky
- Návratovou hodnotu pro filtrační menu
- Identifikátor vybrané položky pro první zobrazení filtračního menu

Položka seznamu

Položka seznamu představuje základní grafický prvek pro vykreslení seznamu položek. Položka seznamu po kliknutí otevírá novou obrazovku detailu položky nebo přesměrovává na externí odkaz. Položku seznamu je vhodné implementovat komponentově, protože je použita na obrazovce novinek a událostí a zároveň na obrazovce pracovních nabídek.

Položka seznamu má následující vlastnosti:

- Titulek položky
- Popisek položky
- Obrázek položky

- Identifikátor kategorie, ke které náleží
- Datum položky, které dokáže přeložit na „dnes“ a „včera“ pokud se jedná o včerejší datum
- Odkaz na detail položky nebo externí odkaz, kam odkáže po kliku na položku
- Pole identifikátor všech kategorií

Filtrační menu 2

Filtrační menu 2 je druhým typem filtračního menu pro výběr dle kategorie v listu v aplikaci. Představuje komponentu, která zapouzdřuje položky ve filtračním menu. Rozdíl tohoto menu je, že dokáže vybrat více položek najednou, namísto přepínání mezi jednotlivými položkami. Příkladem použití je obrazovka jídelníčků, kde menu dokáže přidávat a odebírat jídelníčky na obrazovce jídelníčků.

Filtrační menu 2 má následující vlastnosti:

- Pole objektů kategorií
- Návratovou hodnotu pro akci filtrace

Položka filtračního menu 2

Položka filtračního menu 2 představuje jednu položku, kterou lze filtrovat a zároveň jde vybrat více položek najednou. Je vhodné ji implementovat komponentově z technického důvodu funkčnosti filtračního menu 2. Vybráním položky ve filtračním menu se v návratové hodnotě vrací identifikátor nebo pole identifikátorů kategorie pro akci filtrace.

3.1.4 Frameworking

Celá aplikace, včetně přepoužitelných frontendových komponent, je vyvinuta v Javascriptovém frameworku React Native, který je vhodný pro multiplatformní vývoj pro mobilní aplikace. Jelikož komponenty jsou vyvíjeny pouze pro React Native Framework, postačí dodržet předepsanou strukturu projektu dle dokumentace a není zapotřebí vyvíjet vlastní Framework pro správu komponent a jejich kompatibilitu.

Základní struktura projektu je následující:

```
src
    assets
    components
    feature
        featureName
            components
            service
    locales
    navigation
    styles
```

- Assets – složka pro statické soubory, jako jsou obrázky a ikonky
- Components – složka pro přepoužitelné komponenty napříč aplikací
- Feature – složka pro jednotlivé funkčnosti aplikace
    - FeatureName – zde je umístěn základní soubor obrazovky funkčnosti
        - Components – složka pro komponenty funkčnosti, které nejsou využity jinde v aplikaci a nejsou přepoužity
        - Service – složka pro zajištění volání externích zdrojů
- Locales – složka pro umístění jazykové lokalizace aplikace
- Navigation – složka pro základní navigaci skrz aplikaci
- Styles – složka pro globální styly aplikace, fonty a předdefinované barvy pro celou aplikaci

Jak z popisu vychází, tak komponenty, které byly v analýze identifikovány jako přepoužitelné komponenty, budou umístěny ve složce Components, kde budou na dosah pro implementaci kdekoli v celé aplikaci. Nevznikne tak zmatek a vývojář vždy ví, kde komponentu hledat a kam se na ni v implementaci odkázat.

Komponenty, které jsou z technického hlediska potřebné implementovat jako komponenty, ale nebudou přepoužity nikde jinde, je vhodné uložit ve složce Feature / Components, aby zbytečně nebyla hlavní složka pro přepoužívané komponty plná a nepřehledná. V případě, že budoucí analýza nových funkčností zjistí, že komponenta specifická pro danou funkčnost je vhodná k přepoužití v nových návrzích, tak lze jednoduše komponentu přesunout do složky Components a upravit implementaci s odkazem na komponentu z původní funkčnosti.

3.1.5 Implementace komponent

Podle návrhu komponent jsou komponenty implementovány ve složce Components, odkud jejich implementace lze přepoužívat v celé aplikaci. Rozhraní komponent je implementováno dle návrhu, kde mohou být některá rozhraní lehce rozšířena z technických důvodů, se kterými návrh nemusel počítat nebo je neodhalil.

Podle zadání jsou komponenty implementovány v jazyku React Native, který využívá JSX (JavaScript XML soubory) se XML specifickými značkami pro React Native. Příkladem může být View, Image, ImageBackground, Text, Button, Input apod. Většina implementovaných komponent převážně využívá značky View, Image a Text, ze kterých vytváří funkční celky, neboli komponenty.

Každá komponenta má definované své rozhraní pro data, která jsou nutná pro její vykreslení. Rozhraní definují Props komponenty, které v sobě ukládají data získaná skrze rozhraní od nadřazené komponenty. V komponentě se Props data adresují k vykreslení na specifické místo v XML znacích.

Některé komponenty (stavové) si musí udržet vnitřní stav, kde se využívá State. Data uložená ve State se mohou dynamicky měnit a komponenta na to reaguje.

Všechny komponenty jsou implementovány, jako funkcionální komponenty. Z důvodu použití funkcionálních komponent se využívají takzvané React Hooks, díky kterým si funkcionální komponenty mohou předávat informaci o změnu stavu State.

CustomInput

CustumInput je implementace pro komponentu textového pole, která má dle návrhu definované rozhraní s datovými typy:

- label?: String;
- value?: String;
- secureTextEntry?: boolean;
- setState?: any;
- state?: any;
- callback: any;

Lze si všimnout rozšíření rozhraní o parametry setState a state, které slouží k přenášení a nastavování vnitřního stavu komponenty z místa, kde je komponenta použita.

Label není povinný, jelikož některé inputy nemusí nutně vyžadovat popisek, takže v případě nezaslání dat se popisek textového pole nevykreslí. Value slouží k inicializační hodnotě, která také není povinná, jen pro případ, kdy bychom chtěli textové pole předvyplnit daty pro uživatele. SecureTextEntry zajišťuje formátování textového pole a skrytí psaných znaků. Callback parametr je použit pro návratovou hodnotu textového pole, aby v místě použití implementace bylo možné získat data z textového pole zpět. CallBack se volá s každou změnou obsahu textového pole, což zajišťuje, že komponenta předává vždy aktuální data nazpět.

CustomButton

CustomButton je implementace pro komponentu tlačítka, která má dle návrhu definované následující rozhraní:

- title: String;
- theme?: String;
- size?: String;
- disabled?: boolean;
- onPress: any;

Rozhraní komponenty se oproti návrhu rozšířilo o některé parametry. Theme má předdefinované hodnoty [black, white, blue, red, grey] a size má předdefinované hodnoty [large, middle, small], které ovlivňují grafické zobrazení tlačítka dle návrhu. Oba parametry nejsou povinné a defaultně má tlačítko černý grafický návrh. Parametr disabled není povinný a automaticky se počítá, že tlačítko není ve stavu disabled, pokud parametr nedostane data.

Parametr onPress vrací informaci o akci zmáčknutí tlačítka do místa, kde je komponenta tlačítka použita. V této komponentě nejsou použity žádné vnitřní stavy komponenty, jelikož to není zapotřebí.

ProfileCard

ProfileCard je implementace pro komponentu karta profilu uživatele, která má dle návrhu definované následující rozhraní:

- title: String;
- description: String;
- balance: String;
- balanceDescription: String;
- setState: any;
- state: any;
- visible: boolean;

Rozhraní komponenty se rozšířilo o pár parametrů oproti návrhu z technického důvodu. První čtyři parametry slouží pro přenos dat k vykreslení, jako je titulek, popisek, zůstatek a popisek zůstatku.

Parametry setState a state slouží k přenosu možnosti ovlivnění vnějšího stavu, kde je komponenta využita, aby pomocí parametru visible mohla nastavit odpovídající stav pro otevření nebo zavření přihlašovací obrazovky po kliknutí na profilovou kartu uživatele.

CardItem

CardItem je implementace komponenty karty funkčnosti, která má dle návrhu definované následující rozhraní:

- title: String;
- description?: String;
- imagePath: any;
- navigation: any;
- target: any;
- shortCuts: ShortCut[]
- tabBarShortCutTarget?: String;

Rozhraní komponenty je rozšířené o pár parametrů oproti návrhu z technického důvodu implementace. Title, description a imagePath slouží pro vykreslení dat v komponentě.

Parametr navigation slouží k přenesení kontextu navigace a možnosti otevření nové obrazovky po stisknutí karty. Parametr target určuje, na jakou obrazovku se má navigace po stisknutí karty odkázat.

Poslední parametr tabBarShortCutTarget slouží k zajištění akce, pokud karta funkčnosti nemá otevřít nové okno, ale pouze přepnout spodní TabBar aplikace, který slouží jako základní orientační prvek aplikace.

ShortCuts je parametr pro vykreslení volitelných zkratek na kartě funkčnosti.

ShortCutEnhancedView

ShortCutEnhancedView je implementace komponenty sekce zkratek v kartě funkčnosti, která dle návrhu má definované následující rozhraní:

- navigation: any;
- shortCuts: ShortCut[];

Rozhraní se rozšířilo oproti návrhu o jeden parametr Navigation, který předává kontext navigace k otevření nové obrazovky po vybrání zkratky.

Parametr shortCuts z pole zkratek vykresluje položky zkratek, ke kterým používá další komponentu TouchableShortCutButton.

TouchableShortCutButton

TouchableShortCutButton je implementace komponenty položka zkratky v sekci zkratek, která dle návrhu má následující rozhraní:

- textColor: string;
- icon: any;
- target: string;
- label: string;

Rozhraní nebylo upraveno oproti původnímu návrhu. Všechny parametry slouží k předání dat k vykreslení, jak bylo popsáno v návrhu komponenty.

CardSettingsItem

CardSettingsItem je implementace komponenty zástupce karty funkčnosti, která dle návrhu má následující rozhraní:

- title: String;
- description?: String;
- icon: String;
- selected: boolean;
- callback: any;

Rozhraní komponenty se proti návrhu nijak nezměnilo. Parametry title, description a icon předávají data k vykreslení komponenty. Parametr selected je pro inicializační stav vnitřního stavu komponenty vybráno / nevybráno.

Parametr callback je parametr pro návratovou hodnotu, která vrací vnitřní stav do místa, kde je komponenta použita, aby mohla spustit akci po stisknutí karty.

TabFilter

TabFilter je implementace komponenty filtračního menu, která má dle návrhu následující rozhraní:

filterItems: any;
callback: any;

Rozhraní se oproti návrhu nezměnilo, kde filterItem předává pole položek a callback je návratová hodnota vyvolaná kliknutím na položku filtračního menu a předána do místa použití komponenty, aby vyvolala příslušnou akci.

TabFilterItem

TabFilterItem je implementace komponenty položky ve filtračním menu, která dle návrhu má následující rozhraní:

- itemKey: string;
- itemTitle: string;

- itemValue: string;
- callback: any;
- selectedItem: boolean;

Rozhraní se oproti návrhu nijak nezměnilo. Po vybrání položky vrací TabFilterItem v parametru callback jedinečný identifikátor nadřazenému filtračnímu menu, který tuto hodnotu očekává a posílá dále.

NewsFeedItem

NewsFeedItem je implementace komponenty položka seznamu, která dle návrhu má následující rozhraní:

- title: string;
- description: string;
- imagePath: string;
- category: any;
- date: Date;
- contentUrl: string;
- categories: any;
- navigation: any;
- target: any;

Rozhraní komponenty bylo oproti návrhu rozšířeno o dva parametry navigation a target. Zbylé parametry slouží ke správnému vykreslení položky tak, jak bylo navrženo v návrhu komponenty. Parametr navigation předává kontext o navigaci, aby bylo možné po vybrání položky otevřít novou obrazovku s detailem položky. Parametr target slouží k identifikaci, jakou obrazovku navigace má otevřít po vybrání položky.

FoodPointProviderFilterScrollView

FoodPointProviderFilterScroolView je implementace komponenty filtračního menu 2, která má dle návrhu následující rozhraní:

filterItems: any;

callback: any;

Rozhraní se oproti návrhu nezměnilo, kde FoodProviderFilterItem předává pole položek a callback je návratová hodnota vyvolaná kliknutím na položku filtračního menu 2 a předána do místa použití komponenty, aby vyvolala příslušnou akci.

FoodProviderFilterItem

FoodProviderFilterItem je implementace komponenty položky ve filtračním menu 2, která dle návrhu má následující rozhraní:

- itemKey: string;
- itemTitle: string;
- itemValue: string;
- callback: any;
- selectedItem: boolean;

Rozhraní se oproti návrhu nijak nezměnilo. Po vybrání položky vrací FoodProviderFIlterItem v parametru callback jedinečný identifikátor nadřazenému filtračnímu menu 2, které tuto hodnotu očekává a posílá dále.

3.1.6 Sestavení

Po fázi implementaci přepoužitelných komponent, které byly identifikovány ve fázi návrhu systému, nic nebrání komponenty použít v navržených místech aplikace dle navrženého frameworkigu aplikace. Sestavením aplikace z implementovaných přepoužitelných komponent vznikne plně funkční multiplatformní mobilní aplikace dle systémové analýzy.

Přihlašovací obrazovka používá implementované komponenty:

- Textové pole (CustomInput)
- Tlačítko (CustomButton)

Domovská obrazovka používá implementované komponenty:

- Karta profilu uživatele (ProfileCard)
- Karta funkčnosti (CardItem)
- Sekce zkratek v kartě funkčnosti (ShortCutEnhancedView)

- Položka zkratky v sekci zkratek (TouchableShortCutButton)
- Zástupce karty funkčnosti (CardSettingsItem)

Obrazovka novinek a událostí používá implementované komponenty:

- Filtrační menu (TabFilter)
- Položka filtračního menu (TabFilterItem)
- Položka seznamu (NewsFeedItem)

Obrazovka pracovních nabídek používá implementované komponenty:

- Filtrační menu (TabFilter)
- Položka filtračního menu (TabFilterItem)
- Položka seznamu (NewsFeedItem)

Obrazovka jídelníčků používá implementované komponenty:

- Filtrační menu 2 (FoodPointProviderFilterScroolView)
- Položka filtračního menu 2 (FoodProviderFilterItem)

Ze sestavení komponent do funkční aplikace podle systémové analýzy vychází, že Domovská obrazovka využívá nejvíce přepoužitelných komponent, ale Obrazovka novinek a událostí a Obrazovka pracovních nabídek sdílí nejvíce komponent.

3.1.7 Archivace

Všechny komponenty, které byly vytvořeny v rámci vývoje mobilní aplikace byly uloženy ve struktuře projektu ve složce Components. Pro případné přepoužití v rámci budoucího vývoje této mobilní aplikace vývojáři naleznou komponenty na jednom místě a lehce se zorientují, jaké komponenty mají danou funkčnost. Pro případ nejasnosti každá komponenta má v sobě uvedenou dokumentaci svého rozhraní, aby bylo jednoduché komponentu přepoužít a nevznikly nejasnosti, jaká data komponenta očekává a vývojář nemusel zkoumat vnitřní kód komponenty.

Celý projekt je zálohovaný v týmovém úložišti Git, kde je možné komponenty najít, pokud by v rámci týmu byla vyvíjena jiná aplikace a tyto komponenty by byly vhodné k přepoužití v dané aplikaci.

Komponentová knihovna ovšem nebude publikována jako veřejná knihovna komponent pro použití třetích stran. Frontendová komponentová knihovna je velice

specifická po grafickém návrhu a nedá se tedy očekávat, že by některá třetí strana chtěla komponenty přepoužít takzvanou metodou black-box.

3.1.8 Testování

Aplikace je vyvíjena v jazyku React Native, který umožňuje multiplatformní vývoj. Aplikace je připravena na obě nejpoužívanější mobilní platformy iOS a Android. Vývoj byl z převážné časti proveden v simulátoru iOS na rozměru obrazovky iPhone 11.

Pro otestování všech komponent byl zvolen modulový test, kdy byla každá komponenta testována samostatně vývojářem na iOS platformě po dokončení vývoje.

Pro odlišné velikosti obrazovek na iOS platformě byl zvolen obecný test průchodem aplikací, kde se otestovalo korektně grafické vykreslení komponenty dle jejich návrhu a správná funkčnost zapojených komponent do místa jejich použití. Stejným způsobem se přistoupilo k otestování Android platformy na odlišných velikostech obrazovky, a navíc byly otestovány správné závislosti použitých knihoven v aplikaci.

Obecně průchod aplikací s integrovanými komponentami na obou platformách iOS a Android prošel bez chyby a funkčně i graficky se aplikace chovala dle návrhu a analýzy aplikace.

3.2 Analýza jakosti obou přístupu

Prvním způsobem, jak měřit jakost komponentového přístupu, je zaměřit se na vybrané metriky standartu ISO / EIC 25023 z teoretických východisek této práce. Záměrně byly vybrány metriky týkající se komponentového přístupu z oblasti udržovatelnosti a přenositelnosti, které dokážou zhodnotit kvalitu komponentového přístupu implementované frontendové knihovny a v některých případech porovnat rozdíl oproti klasickému nekomponentovému přístupu.

Druhým způsobem, jak zhodnotit a porovnat výhody a nevýhody komponentového přístupu oproti klasickému přístupu k vývoji software, je za pomoci zvolených klasických metrik z teoretické části. Metriky LOC – lines of codes a CYCLO - McCabova cyklomatická složitost budou měřeny na dvou odlišných zdrojových kódech.

První zdrojový kód bude implementace části mobilní aplikace za použití implementovaných frontendových komponent v rámci této diplomové práce.

Druhý zdrojový kód bude implementace identické části mobilní aplikace s tím rozdílem, že nebude použit komponentový přístup. Znamená to, že všechny komponenty, které byly implementovány odděleně jako samostatné a nezávislé části zdrojového kódu, budou napevno zaneseny do zdrojového kódu, kde byly původně komponenty použity. V komponentovém přístupu komponenty získávaly data pro vykreslení skrze komponentové rozhraní. Nyní vnitřní kód komponenty bude součástí jednotného zdrojového kódu dané obrazovky a získá data přímo v místě použití namísto rozhraní.

Metriky budou vyhodnoceny nad oběma zdrojovými kódy a z výsledků měření budou vyhodnoceny výsledky k porovnání komponentového a klasického přístupu.

3.2.1 Vybrané metriky ISO / EIC 25023

3.2.1.1 Spojení komponent

Spojení komponent hodnotí, jak jsou komponenty nezávislé na ostatních komponentách a na změnách v nich pomocí následujícího vzorce:

$$X = A / B$$

A je počet komponent implementovaných bez vlivu na jiné komponenty

B je počet komponent celkově

Z implementované knihovny frontendových komponent je většina komponent nezávislých na ostatní komponentách:

- Textové pole (CustomInput)
- Tlačítko (CustomButton)
- Karta profilu uživatele (ProfileCard)
- Karta funkčnosti (CardItem)
- Zástupce karty funkčnosti (CardSettingsItem)
- Položka seznamu (NewsFeedItem)

Některé jsou přímo na sobě závislé, jelikož fungují dohromady a bez sebe se neobejdou, ale zároveň jedna ze závislých komponent může fungovat sama o sobě. Tučně zvýrazněné jsou tedy ty komponenty, které jsou výlučně závislé na jiné komponentě.

- **Filtrační menu (TabFilter)**
- Položka filtračního menu (TabFilterItem)
- **Filtrační menu 2 (FoodPointProviderFilterScroolView)**
- Položka filtračního menu 2 (FoodProviderFilterItem)
- **Sekce zkratek v kartě funkčnosti (ShortCutEnhancedView)**
- Položka zkratky v sekci zkratek (TouchableShortCutButton)

Výpočet je tedy následující: X = 3 / 12

X = 0,25

Spojení komponent je tedy 25%, což značí nízkou úroveň spojení. Znamená to tedy 75% nezávislosti komponent na sobě. 75% vypovídá o vysoké úrovni samostatnosti a přenositelnosti komponent do jiného vývoje, kam není potřeba přenášet další komponenty. Zamezí se tím redundantnímu zdrojovému kódu a zvýšené složitosti projektu.

3.2.1.2 Přepoužitelnost prvků

Přepoužitelnost prvků hodnotí, kolik prvků je přepoužitelných a lze je použít bez úpravy na jiných místech aplikace či dokonce v jiných projektech. Používá s k tomu následující vzorec:

$$X = A / B$$

A je počet prvků, které jsou navrženy k přepoužití

B je celkový počet prvků

Jelikož celá knihovna byla od začátku uvažována, navržena a implementována na základech principů přepoužitelnosti, všechny komponenty jsou vhodné k přepoužití.

Výpočet je tedy následující: X = 12 / 12

X = 1

Přenositelnost komponenty je tedy 100%, což značí vysokou úroveň přepoužitelostni a splnění cíle vytvoření komponentové knihovny, kterou lze přepoužít v budoucím rozvoji aplikace a snížit tak budoucí náklady na vývoj aplikace.

Výpočet pro stejné části kódu, který byl implementován klasickým nekomponentovým přístupem je následující:

$$X = 0 / 12$$

Přenositelnost při nekomponentovém přístupu je 0%, což může zvyšovat náklady na vývoj a celkově na celý projekt.

3.2.1.3 Účinnost úpravy

Účinnost úpravy hodnotí, jak efektivně jsou implementovány úpravy v porovnání s očekávaným časem. Zde lze porovnat komponentový a klasický přístup vývoje v samostatných výpočtech.

Za účelem měření jsou zvoleny 2 komponenty, které jsou ve stávající implementaci aplikace použity na více místech nebo opakovaně.

První komponenta je Karta funkčnosti (CardItem), která je opakovaně použita na domovské stránce aplikace. V případě potřeby úpravy této komponenty, například změna rozložení prvků v kartě a přidání notifikační prvku karty, lze odhadovat dobu implementace na 2 hodiny práce. V případě, že karta funkčnosti je ve stávající implementaci použita 4x na domovské stránce, tak by teoreticky 4x vzrostla pracnost implementace na oddělených místech. Na druhou stranu se dá očekávat, že po úpravě první karty by programátor postupoval podle vytvořeného vzoru a pracnost by klesla na půl hodinu na každý prvek. Z toho lze tedy vyvodit, že při komponentovém přístupu by implementace programátorovi trvala 2 hodiny a změna by se projevila na všech místech použití komponenty. V případě klasického nekomponentového přístupu by programátorovi úprava karty funkčnosti vyšla na 2 + 0,5 + 0,5 + 0,5 hodiny, tedy celkem 3,5 hodiny. Tento čas je navíc relativní, jelikož s množstvím opakování tohoto prvku roste i časová náročnost úpravy, ale u komponentového přístupu časová náročnost nenarůstá.

Druhou komponentou je Položka seznamu (NewsFeedItem), která je použita ve dvou obrazovkách, a to v obrazovce novinek a událostí a obrazovce pracovních nabídek. Komponenta je přepoužita na dvou odlišných místech. V případě potřeby úpravy této

komponenty, například změna rozložení popisku na více než 2 řádky, úprava potrvá programátorovi 0,5 hodiny. V případě komponentového přístupu komponentu upraví na jednom místě a změna se projeví na vše místech použití komponenty. V případě klasického přístupu programátor musí upravit stejně zdrojový kód na dvou odlišných místech, což by mohlo zabrat další 0,5 hodiny. Tedy pracnost se prodlouží o 0,5 hodiny, což je dvojnásobný čas pracnosti. Čas pracnosti roste s počtem použití komponenty v celé aplikaci a navíc může dojít k nepřesnostem a zanesení chyb a neintegritě zdrojového kódu na dvou místech.

Vzorec pro výpočet účinnosti úpravy je následující:

$$X = \sum_{i=1\ do\ n} (A_i/B_i)\ /\ n$$

A je celkový čas vynaložený na úpravu

B je očekávaný čas vynaložený na úpravu

N je počet měřených úprav v rámci komponenty

Čím se víc X blíží 0, tak se jedná o více efektivní úpravu.

Výpočet pro komponentový přístup:

Karta funkčnosti: (2/2,2) / 2 = **0,4545**

Položka seznamu: (0,5 / 0,7) / 1 = **0,714**

Výpočet pro klasický nekomponentový přístup:

Karta funkčnosti: (3,5 / 2,2) / 2 = **0,795**

Položka seznamu: (1 / 0,7) / 1 = **1,428**

Výsledkem měření jednoznačně vychází efektivnější účinnost úprav pro komponentový přístup. Neefektivnost nekomponentového přístupu roste s počtem použití a nutnosti úprav na více místech.

3.2.1.4 Podobnost použití

Podobnost použití hodnotí, jaký poměr funkcionalit nahrazeného produktu může být implementováno bez jakékoli úpravy, což je základním principem komponentového přístupu vývoje. Díky definovanému rozhraní komponent lze komponenty použít kdekoli a

stačí jen správně pochopit a implementovat definované rozhraní komponenty. V porovnání s klasickým nekomponentovým přístupem, kde není definované žádné rozhraní pro použití části kódu na jiném místě, je vždy nutné upravit kód vzhledem k místu použití a pochopit vnitřní logiku komponenty.

K výpočtu se používá následující vzorec:

$$X = A / B$$

A je počet funkcionalit vhodných k implementaci bez nutných úprav

B je počet přenesených funkcionalit

Všechny implementované komponenty byly použity bez nutnosti jejich úprav na více místech aplikace, tedy výpočet je následující:

$$X = 12 / 12$$

To znamená 100% podobnost použití, což potvrzuje vysoký standard komponentového přístupu v porovnání s klasickým nekomponentovým přístupem, kde každý frontendový prvek pro použití na jiném místě by bylo nutné upravit vzhledem k místu použití:

$$X = 0 / 12$$

To znamená 0% podobnosti použití. 0% potvrzuje výhody komponentového přístupu.

3.2.2 Další metriky

3.2.2.1 LOC – lines of codes

Pro měření počtu řádků kódu byla vybrána domovská obrazovka, která byla implementována oběma způsoby, komponentovým přístupem za použití 5 komponent a klasickým nekomponentovým přístupem za použití kódu komponenty přímo v domovské obrazovce.

Domovská obrazovka implementovaná komponentovým přístupem má celkem **338 řádků** včetně stylování. Dále se dá sečíst s počtem řádků jednotlivých komponent.

- Karta profilu uživatele (ProfileCard): **88 řádků**
- Karta funkčnosti (CardItem): **84 řádků**
- Sekce zkratek v kartě funkčnosti (ShortCutEnhancedView): **33 řádků**
- Položka zkratky v sekci zkratek (TouchableShortCutButton): **67 řádků**

- Zástupce karty funkčnosti (CardSettingsItem): **80 řádků**

Celkově tedy implementace domovské stránky včetně použitých komponent je na **690 řádcích**.

Domovská obrazovka implementovaná klasickým nekomponentovým přístupem může sdílet stylování části kódu, které jsou v komponentovém přístupu použity jako komponenty, ale jinak bude počet řádků narůstat. Jedná se o jediný soubor, který má celkem **1001 řádků**, což je nárůst o 311 řádků oproti komponentovému přístupu.

3.2.2.2 CYCLO - McCabova cyklomatická složitost

McCabova cyklomatická složitost byla změřena pro oba způsoby přístupu vývoje, ale její výsledky nejsou použitelné pro implementaci frontendových elementů, kde nevzniká tak často možnost různých průchodů programem, a proto tato metrika nakonec nebyla zohledněna pro závěry této práce.

3.3 Výsledky porovnání komponentového a klasického přístupu vývoje

Klasický přístup vývoje má podobný postup, jako komponentový přístup s jedním zásadní rozdílem. Klasický přístup se nezaměřuje na přepoužití opakujících se prvků ve fázi specifikace, návrhu ani implementace. Je možné, že v průběhu implementace nebo v rámci údržby a refaktoringu zdrojového kódu může z iniciativy vývojového týmu dojít k vytvoření komponent a tím pádem ke zjednodušení a zpřehlednění zdrojového kódu. To se však odehraje až po původní implementaci a zvyšují se náklady a časová náročnost v průběhu údržby.

Komponentový přístup vývoje pamatuje na principy komponentového přístupu ve všech fázích procesu vývoje a již od samého začátku navrhne systém se zaměřením na komponenty a přepoužitelnost. Proces vývoje může být trochu časové náročnější a nákladnější, kvůli fázím identifikace komponent, vytvoření frameworku, úložiště komponent a dalším fázím, které v klasickém přístupu nejsou, ale v implementační fázi a v budoucím rozvoji se rychle začnou projevovat výhody a snižovat náklady.

Vlastní výpočet metrik je popsán v kapitole 3.2.1 a 3.2.2 a teoretický základ k metrikám podložen v kapitolách 2.2.1.1 a 2.2.2.

Potvrzení porovnání přístupů na měřených metrikách:

Tabulka 4 Porovnání klasického a komponentového přístupu (zdroj: vlastní zpracování)

metrika	**Klasický přístup**	**Komponentový přístup**
Spojení komponent	---	75 %
Přepoužitelnost prvků	0 %	100 %
Účinnost úpravy – Karta funkčnosti	0,795 (*)	0,4545 (*)
Účinnost úpravy – Položka seznamu	1,428 (*)	0,714 (*)
Podobnost použití	0 %	100 %
LOC – Domovský obrazovka	1001 řádků	690 řádků

(*) – čím je číslo nižší, tím je účinnost úpravy efektivnější

4 Závěr

V teoretické části byly popsány vývojové procesy klasické a vývojové procesy komponentového přístupu. Dále byly popsány obecné výhody komponentového přístupu, představeny základy javascriptového frameworku React Native a došlo k identifikaci a výběru vhodných metriky pro měření jakosti softwarového produktu z hlediska srovnání komponentového a klasického přístupu vývoje software.

V praktické části byla navržena a implementována frontendová komponentová knihovna podle popsaných postupů vývojového komponentového modelu Y-model, který byl popsán v teoretické části této práce. Dodržení postupů Y-model zajistilo splnění komponentového přístupu a dosažení všech výhod nabízených komponentovým přístupem v rámci vývoje software.

Pro ověření výhod komponentového přístupu byly podle normy ISO / IEC 25023 naměřeny metriky Spojení komponent, Přepoužitelnost prvků, Účinnost úpravy, Podobnost použití a porovnány na zdrojovém kódu identické aplikace implementované klasickým nekomponentovým přístupem.

Dále byly zvoleny klasické metriky pro měření jakosti zdrojového kódu LOC a CYCLO. Z důvodu frontendového zaměření zdrojového kódu byla metrika CYCLO vyřazena.

Všechny měřené a spočítané metriky ověřily výhody komponentového přístupu vývoje v porovnání s klasickým nekomponentový přístupem. Potenciální výhody byly stanoveny a popsány v teoretické části této práce.

Výhoda komponentového přístupu „Zkrácený vývojový čas“ byla ověřena metrikou Účinnost úpravy, který vyšel přibližně 2x rychleji pro komponentový přístup pro obě úpravy, za předpokladu nízkého přepoužití upravovaných komponent. Časová náročnost pro klasický nekomponentový přístup by rostla s každou opakující se implementací frontendového prvku, který v komponentovém přístupu představuje samostatná komponenta.

Výhoda komponentového přístupu „Snížení vynaloženého úsilí“ při aplikování komponentového přístupu lze také potvrdit metrikou Účinnost úpravy. Účinnost úpravy byla 2x rychlejší pro komponentový přístup oproti stejné úpravě v klasickém

nekomponentovém přístupu. Programátor nemusel vynaložit tolik práce, což se projevilo na času dokončení úpravy, aby dokončil úpravu v komponentovém přístupu.

Výhoda komponentového přístupu „Přepoužitelnost" byla ověřena metrikou Přepoužitelnost prvků, kde komponentový přístup značil 100% přepoužitelnosti a klasický nekomponentový přístup nulovou přepoužitelnost.

Výhoda komponentového přístupu „Přizpůsobivost" lze také potvrdit metrikou Účinnost úpravy, kde se potvrdilo, že není nijak časově náročné komponenty upravit, jak by bylo potřeba.

Výhoda komponentového přístupu „Škálovatelnost" lze také potvrdit metrikou Účinnost úpravy, jelikož změny se netýkaly změn rozhraní a dále to potvrzuje metrika Spojení komponent, která má 75% nezávislosti komponent na sobě.

Výhoda komponentového přístupu „Produktivita" byla ověřena metrikou LOC – počtem řádků, kde komponentový přístup měl přibližně o třetinu méně řádků než klasický nekomponentový přístup. To snížilo náklady. Z dlouhodobého hlediska by rozdíl mezi počtem řádků ještě více narůstal.

Závěrem je možné vyhodnotit, že případová studie potvrdila výhody komponentového přístupu oproti klasickému nekomponentovému přístupu.

V případě budoucího rozvoje, kdy se budou vyvinuté komponenty více přepoužívávat a zároveň se implementují nové komponenty, se rozdíly v produktivitě, škálovatelnosti, přizpůsobivosti, přepoužitelnosti, počtu řádků a celkově v nákladech na projekt budou ještě více projevovat a narůstat. Zároveň udržování aplikace by bez komponentového přístupu bylo mnohem náročnější a nákladnější.

Obecně lze potvrdit, že komponentový přístup je vhodný pro rozsáhlejší projekty, kde se počítá s dlouhodobým vývojem, častými úpravami a postupným přidáváním nových funkcionalit. Komponentový přístup se může zdát ze začátku vývoje nákladnější, náročnější a obecně časově náročnější. Výhody komponentového přístupu se začnou projevovat s růstem projektu a úvodní zvýšené vynaložené úsilí se postupem času začne vracet, a naopak ušetří podstatné zdroje a náklady projektu. Důležité je dodržet vývojový proces komponentového přístupu podle stanovených kroků, jelikož komponentový přístup není jen o fázi implementace projektu, ale o celkovém projektovém uchopení a řízení všech fází vývojového cyklu od komponentového návrhu až po testování komponent.

Nelze říct, že klasický nekomponentový přístup je nevýhodný a neefektivní. V případě projektů menších rozměrů, případových studií a prototypů, kde se nehledí na efektivitu vývoje a kvalitu projektu z dlouhodobého hlediska, se klasický přístup může hodit více. Z důvodu snížených počátečních nákladů a vynaložených zdrojů je klasický přístup výhodný v krátkém časovém horizontu. V případě, že se projekt začne rozrůstat z prototypu do komplexního a dlouhodobého projektu, je vhodné projekt převrátit do komponentového přístupu, pokud je to možné, nebo začít raději od začátku komponentově. Projekt vyvíjený pod klasickým přístupem v dlouhém časovém rámci může narůst do velkých rozměrů, kde je velice nákladná sebemenší úprava.

Je důležité zmínit, že každý přístup má své výhody i nevýhody a je významné si na začátku vývoje projektu uvědomit, jaký má projekt cíl a rozsah. Špatně zvolený přístup by mohl zapříčinit neúspěch celého projektu i při zvýšeném vynaloženém úsilí.

Na základě provedeného výzkumu a případové studie, která jasně definuje typ projektu větších rozměrů a dlouhodobého vývoje, se jasně jeví výhodnější komponentový přístup, který přináší potenciál výrazného snížení nákladů vynaložených na projekt.

Náklady se sníží nejen zkrácením vynaloženého času na implementační fázi, jak by se z prvního pohledu mohlo zdát, ale celkový komponentový přístup zkrátí vynaložený čas na práci při návrhu aplikace a vytváření grafických návrhů, kde komponentový přístup jasně vymezí hranice navrhovaných funkcionalit. Uvažování komponentově ve fázi návrhu se na vynaloženém čase a nákladech mnohonásobně projeví v implementační, testovací a údržbové fázi.

5 Seznam použitých zdrojů

[1] Buchalcevová , Alena. *Metodiky budování informačních systémů.* Praha : Oeconomica, 2009. ISBN 978-80-245-1540-3.

[2] Kadlec, Václav. *Agilní programování metodiky efektivního vývoje softwaru.* Brno : Computer Press, 2004. ISBN 80-251-0342-0.

[3] Testování softwaru. *RUP – Rational Unified Process.* [Online] [Citace: 15. 11 2020.] http://testovanisoftwaru.cz/manualni- testovani/modely-zivotniho-cyklu-softwaru/rup/.

[4] *A Comparision Between Traditional and Component Based Software Development Process Models.* DULAWAT, MANJU KAUSHIK & M. S. Issue 3, Udaipur : J. Comp. & Math. Sci., 2012, Sv. vol 3. 308-319.

[5] *Component Based Software Development – "An Efficient Approach".* Dr. Parul Gandhi, Kritika Vashisht , Kunal Dawra , Shanu Jaitly , Prasenjit Banerjee. Issue 1, místo neznámé : International Journal of Scientific Engineering and Science, 2018, Sv. vol 2. 2456-7361.

[6] Veryard, Richard. *Component-Based development.* [Online] 1999. [Citace: 21. 11 2020.] http://www.users.globalnet.co.uk/~rxv/CBDmain/cbdfaq.htm.

[7] *Component-based software development approach: new opportunities and challenges.* Pour, G. Santa Barbara : Technology of Object-Oriented Languages and Systems, 1998. 0-8186-8482-8.

[8] *Component-based software engineering for embedded systems.* Crnkovic, I. Saint Louis : International Conference on Software Engineering (ICSE), 2005. 1-59593-963-2.

[9] *Towards a classification model for component-based software engineering research.* Hall, Stephen, Sommerville, Ian a Kotonya, G. Belek-Antalya : Euromicro Conference, 2003, Sv. 29. 1089-6503.

[10] Clements, Linda M. Northrop Paul C. Software Engineering Institute. *A FRAMEWORK FOR SOFTWARE PRODUCT LINE PRACTICE.* [Online] 2012. [Citace: 11. 21 2020.] https://resources.sei.cmu.edu/asset_files/WhitePaper/2012_019_001_495381.pdf.

[11] *Towards Efficient Component-Based Software Development of Distributed Embedded Systems.* Séverine, Sentilles. Västerås : Mälardalen University, School of Innovation, Design and Engineering, 2009. 978-91-86135-43-0.

[12] Sametinger, Johannes. *Software Engineering with Reusable Components.* místo neznámé : Springer, 1997. 978-3-540-62695-4.

[13] *Aspect Component Based Software Engineering1.* Pedro J. Clemente, Juan Hernández. místo neznámé : University of Extremadura, 2003.

[14] *Candidate process models for component based software development.* K. Kaur, H. Singh. Amritsar : Journal of Software Engineering Guru Nanak Dev University, 2010. 18-19-4311.

[15] *Component-Based Development Process and Component Lifecycle.* Ivica Crnkovic, Michel Chaudron, Stig Larsson. Tahiti : International Conference on Software Engineering Advances , 2006. 0-7695-2703-5.

[16] *The W Model for Component-Based Software Development.* Kung-Kiu Lau, Faris M. Taweel and Cuong M. Tran. Manchester : EUROMICRO Conference on Software Engineering and Advanced Applications, 2011.

[17] *Y: A New Component-Based Software Life Cycle Model.* Capretz, Luiz. místo neznámé : Journal of Computer Science, 2005, Sv. 1.

[18] *Připravovaná řada norem ISO/IEC 25000 pro jakost produktu.* Prof. RNDr. Jiří Vaníček, CSc. místo neznámé : Česká zemědělská univerzita, Provozně ekonomická fakulta, Katedra informačního inženýrství.

[19] iso25000.com. ISO 25000 Portal. [Online] https://iso25000.com/index.php/en/iso-25000-standards/52-iso-iec-2502n.

[20] *System and software Quality Requirements and Evaluation (SQuaRE) - Mesurement of system and software product .* 25023, INTERNATIONAL STANDARD ISO / IEC. 2016. ISO / IEC 25023.

[21] *Software product metrics.* Li, Wei. 5, místo neznámé : IEEE, 2000, Sv. 18. 1558-1772.

[22] *A Complexity Measure.* McCabe, T. J. 4, místo neznámé : IEEE Transactions on Software Engineering, 1976, Sv. SE-2. 1939-3520.

[23] Zammetti, Frank. *Practical React Native: Build Two Full Projects and One Full Game using React Native.* místo neznámé : Apress, 2018. ISBN-13: 978-1484239384.

[24] Eisenman, Bonnie. *Learning React Native, 2nd Edition.* místo neznámé : O'Reilly Media, Inc., 2017. 9781491989142.

[25] Inc., Facebook. React. *React.Component.* [Online] [Citace: 27. 12 2020.] https://reactjs.org/docs/react-component.html.

[26] Inc., Facebook. React. *Introducing Hooks.* [Online] [Citace: 27. 12 2020.] https://reactjs.org/docs/hooks-intro.html.