Odhady software založené na scénářích

Scenarios based software estimations

Pavol Kaščák

Bakalářská práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Pavol KAŠČÁK**
Osobní číslo:	**A10668**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Téma práce:	**Odhady software založené na scénářích**

Zásady pro vypracování:

1. **Seznamte se s metodikami odhadů pracnosti a nákladů software.**
2. **Popište metodiky založené na scénářích.**
3. **Analyzujte požadavky na metodiku odhadu ve fázi use case modelu.**
4. **Návrhněte úpravy zvolené metodiky.**
5. **Porovnejte metodiky.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Boehm, B. W., et al.: Cost Models for future Software Life Cycle Process: COCOMO 2.0. The Netherlands, Amsterdam, Science Publishers 1995.**
2. **Boehm, B. W., et al.: COCOMO II Model Definition Manual. USA, Los Angeles, University of Southern California 1999**
3. **International Function Points User Group: Function Point Counting Practices Manual: Release 4.1. 1999.**
4. **Sedláčková, J.: Cenové odhady softwarových projektů. Masarykova univerzita v Brně, Fakulta informatiky, Brno, 2005. Diplomová práce.**
5. **Anda, B. Comparing Use Case based Estimates with Expert Estimates. Proc. of Empirical Assessment in Software Engineering (EASE),Keele, United Kingdom, April 8-10, 2002.**
6. **SMITH, John. The Estimation of Effort Based on Use Cases [online]. Somrs : IBM, 2003 [cit. 2011-01-24]. Dostupné z WWW: ftp://ftp.software.ibm.com/software/rational/web/whitepapers/2003/finalTP171.pdf**

Vedoucí bakalářské práce: **Ing. Radek Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **25. února 2011**

Termín odevzdání bakalářské práce: **7. června 2011**

Ve Zlíně dne 25. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato bakalářská práce se zabývá metodami odhadu nákladů softwarovích projektů v počátečních fázích životního cyklu softwaru, se zaměřením na metodu Use case points. Nejprve je obecně popsán životní cyklus softwaru s následným detailnějším popisem jeho částí. Část práce vysvětluje přístupy k vývoji software se zaměřením na objektově orientovaný přístup a rozdělení používaných metod s jejich charakteristikou. Následuje přesnější popis metody Use case points a její praktické použití formou reálných příkladů. Závěrem jsou zhodnoceny poznatky z dosažených výsledků.

Klíčová slova:

Use case points, UCP, metody odhadu nákladů softwarových projektů, případy užití, scénáře, COCOMO, FPA, Analýza funkčních bodů, životní cyklus software

ABSTRACT

This bachelor's thesis deals with software cost estimations methods in the early stages of software life cycle, focusing on the Use case points method. First, it is generally described software life cycle with the subsequent detailed description of its parts. Part of this work explains the approaches of software development with a focus on object-oriented approach and classification of used methods with their characteristics. It is followed by more accurate description of Use case points method and its practical application through real examples. At the end findings are evaluated form achieved results.

Keywords:

Use case points, UCP, software cost estimation methods, use cases, scenarios, COCOMO, FPA, Function point analysis, software life cycle

PODĚKOVÁNÍ

Chcel by som poďakovať môjmu vedúcemu, pánovi Ing. Radkovi Šilhavému, Ph.D. za jeho ochotu a rady, pri tvorbe tejto práce. Bc. Lukášovi Tínesovi za poskytnutie jeho práce použitej v prvom projekte pre odhady nákladov na softvér.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….…….

podpis diplomanta

OBSAH

ÚVOD

Životný cyklus softvérového projektu je zložitý proces, počas ktorého sa od potreby zákazníka cez rôzne druhy špecifikácií, návrhov, implementovania a testovania až k samotnému nasadeniu produktu do reálneho používania vo firmách stáva plnohodnotný produkt a súčasť denného používania užívateľmi. Určite veľkú časť tohto cyklu tvorí na konci aj jeho údržba, ktorá patrí do poslednej fáze cyklu.

V tejto práci sa naopak začneme zaoberať hneď úplným začiatkom tohto cyklu. Bude sa jednať o prvý krok, ktorý každá spoločnosť zaoberajúca sa vytváraním softvérových produktov musí spraviť. Je ním odhad nákladov spojených s jeho tvorbou. Každý zákazník by rád vedel, ako dlho bude musieť čakať a koľko prípadne musí zaplatiť za vyhotovenie produktu. Práve preto vzniká potreba odhadovať náklady a náročnosť na softvér.

Tieto odhady sú však netriviálny problém, práve preto, že predpovedať množstvo práce, ktorú tvorba softvéru spotrebuje, vyžaduje kompletné pochopenie úlohy resp. požiadavkou. Na začiatku tvorby softvéru nie sú známe presné špecifikácie, ľudia ktorí sú zainteresovaní nemajú kompletní obrázok o tom čo práca bude obnášať. Avšak tieto odhady sú nutné a žiadané hneď v ranných fázach projektu pre určenie, či je projekt vôbec možné uskutočniť, vzhľadom na náklady a následné výnosy. Tradičné modely a metódy (COCOMO) rátajú náklady na základe veľkosti projektu (napríklad podľa počtu riadkov kódu), ako vstupného parametru. V dnešnej dobe sa však každý veľký projekt neobíde bez objektovo orientovaného prístupu. Či už sa to týka len návrhu pomocou modelovacieho jazyka, *Unified modeling language* (UML) alebo samotnej implementácie v nejakom objektovom jazyku.

V tejto práci sa hlavne zameriam na tento fakt a rozoberiem metódu, ktorá priamo vychádza a využíva pre svoje odhady objektovo orientovaný model návrhu. Presnejšie takzvané prípady použitia, anglicky *use cases* (UC). Pri objektovo orientovanom spôsobe vývoja softvéru sú prípady použitia spôsob, akým je popísaná funkcionalita systému. Preto môže byť *use case model* použitý pre predpovedanie veľkosti budúceho systému hneď v ranných fázach softvéru. Tieto prípady použitia už priamo vychádzajú zo špecifikácií od zákazníka, sú súčasťou návrhu a modelujú sa v jazyku UML. Napríklad model COCOMO potrebuje pre svoj odhad vedieť počet riadkov kódu, ktorý v tejto fáze, ale veľmi ťažko dokážeme odhadnúť. Existuje a používa sa aj iný prístup nazývaný *Funcition points*

analysis (FPA), aj táto metóda bude prezentovaná ako aj jej výhody, nevýhody a porovnania jednotlivých metód v posledných kapitolách tejto práce. Ako neskôr uvidíme FPA a UCP si sú stým spôsobom podobné, lebo používajú princíp ohodnocovania funkcionality kladenú na systém, no každá metóda pre tento popis používa iný základ. Práca však začne postupným vysvetľovaním celého problému pri vývoji softvéru a životného cyklu, ktorým každý softvérový produkt prechádza než sa dostaneme k jednotlivým metódam používaným pri odhadov nákladov na vývoj softvéru.

I. TEORETICKÁ ČÁST

1 ŽIVOTNÝ CYKLUS SOFTVÉRU

Ako bolo naznačené v úvode, každý produkt pred nasadením prechádza určitými fázami, jedná sa práve o etapy životného cyklu. Tieto etapy sa vyskytujú v rôznych modeloch životného cyklu a odrážajú činnosti spojené s vývojom softvéru. Spomínané etapy sú nasledovné:

- Analýza a špecifikácia požiadavkou
- Návrh
- Implementácia
- Integrácia a testovanie
- prevádzka a údržba

Každá s týchto etáp tvorí určité percento na celkovom vynaloženom úsilí. Existuje viacej teoretických modelov napríklad vodopadový, iteratívny, špirálový no v prax a pri vedení reálnych softvérových projektov sa používa jeden model nazývaný *Rational unified proecess* (RUP).

RUP je síce založený na vývoji softvéru iteratívnym spôsobom, to znamená po každej iterácii je dostupný spustiteľný kód, ale už nie je len konceptom, ako vyššie spomenuté modely. Tento model kladie dôraz na vizualizáciu softvérového systému pomocou už spomínaného jazyka UML.

Teraz presnejšie popíšem hlavne prvú a druhú etapu cyklu, ktorá je v rámci mojej práce podstatná, nakoľko sa odhady nákladov na softvér uplatňujú v nich. Len pre úplnosť už len stručne popíšem aj ostatné etapy. Pri takomto rozdelení etáp, odhady na náklady softvéru zasahujú do dvoch z nich, stále však hovoríme o úvodnej etape, pretože často býva analýza a návrh softvéru spomínaná ako jedna etapa alebo vo veľmi úzkom spojení. Informácie prezentované v tejto kapitole vychádzajú z literatúry [3].

1.1 Etapy životného cyklu softvéru

1.1.1 Analýza a špecifikácia

V úvode už bol spomenutý nejaký prvý krok, ktorý musí firma spraviť pri vývoji softvéru. Tento krok predstavuje práve prvú etapu životného cyklu, nazývanú analýza a špecifikácia požiadavkou. Tu práve vzniká potreba odhadovať náklady na softvérový projekt. Vzniká tu priestor v ktorom sa uplatňujú rôzne prístupy ktorým bude ďalej venovaná značná časť práce. Analýza a špecifikácia je teda úvodnou fázou vo vývoji softvéru, v ktorej sa zaoberáme požiadavkami zákazníka resp. zadávateľov. Tieto informácie analyzujeme a špecifikujeme. Pozornosť v tejto etape by mala byť venovaná hlavne požiadavkám samotným, nie ich realizácii. Dôležitú a z hľadiska zamerania tejto práce hlavnú časť tejto etapy tvorí, takzvané štúdium vhodnosti resp. uskutočniteľnosti, ktorá zodpovie otázku, či sa do vývoja softvéru vôbec púšťať. Odpoveď nám poskytnú práve metódy odhadu softvéru, ktoré tvoria veľkú časť tejto úvodnej etapy životného cyklu softvéru. Ďalej býva súčasťou takzvaný plán akceptačného testovania, sú to testy na základe ktorých zákazník softvér prevezme. Práve zanedbanie niektorých častí v tejto etape môže viesť k úplnému kolapsu v ďalšom vývoji softvéru. Je nesmierne dôležité venovať veľkú pozornosť práve tejto etape.

1.1.2 Návrh

Pozornosť by som rád venoval aj etape návrhu softvéru, pretože odhady ľudských zdrojov a odhad doby trvania projektu, ktoré potrebujem vedieť ako manažér projektu, riešim práve v tejto etape. Návrh nadväzuje veľmi priamo na špecifikáciu a slúži k ujasneniu koncepcie systému. Zahŕňa presnejšiu špecifikáciu súčastí, výber algoritmov pre realizáciu požadovaných funkcií, na stanovenie logickej a fyzickej štruktúry dát a podobne.

1.1.3 Implementácia

Etapa implementácie zahrňuje programovú realizáciu softvérových súčastí vychádzajúcich z návrhu. Vypracovávanie dokumentácie k jednotlivým súčastiam a ich otestovanie. Jedná sa o samotné programovanie vývojovým tímom, ktorý vytvára na začiatku takzvané jednotkové testy, *unit tests*, na základe ktorých sa nevyskytujú

a nezavádzajú do vývoja chyby. Táto praktika tvorí časť agilnej metodológie, ktoré v dnešnej dobe prevláda pri vývoji softvéru.

1.1.4 Integrácia a testovanie

Po implementácie jednotlivých súčastí je potreba ich integrácia a testovanie, teda spájanie do jedného celku. Nastupuje testovanie systému ako celku, ktoré by malo odhaliť chyby, ktoré nebolo možné odhaliť pri samotnej implementácii jednotlivých podsystémov, nakoľko testy v tejto etape boli príliš konkrétne pre daný podsystém. Objavené chyby sú samozrejme opravované, čo nás vlastne dostáva do predošlej etapy. Testovanie súčastí nie je možné vynechať ani v prípade, že sme ich už predtým testovali, veľmi často sa totižto stáva, že sa do aplikácie zavádzajú nové chyby.

1.1.5 Prevádzka a údržba

Prevádzka a údržba posledná etapa v ktorej sa softvér môže nachádzať. Jedná sa o poskytovanie podpory formou údržby, operatívneho opravovania prípadných chýb, ktoré s nasadením a používaním softvéru do ostrej prevádzky súvisia. Ďalej sem patrí rozširovanie systému o novú funkcionalitu, či prispôsobovanie softvéru meniacim sa požiadavkám podľa potrieb zákazníka, ktoré však vyplývajú zo zmluvy uzavretej v úvodných etapách cyklu.

2 OBJEKTOVO ORIENTOVANÝ PRÍSTUP

Tento prístup sa začal uplatňovať začiatkom deväťdesiatych rokov, počas ktorých začalo vynikať viacero metodológií objektovo orientovanej (OO) analýzy a návrhu. Hlavnou devízou OO prístupu je predovšetkým stabilita navrhovaných častí a jednoduchosť ich zmien. Klasické prístupy pred OO modelovaním vychádzali zo zloženia veľkého počtu funkcií, ktoré mal systém implementované. Takýto prístup je však menej stabilný s pohľadu neustále meniacich sa požiadavkou od užívateľov. Preto sa začal používať a teraz kraľuje OO prístup. Ten nezasahuje len do implementačnej časti, ale aj do počiatočných etáp cyklu, analýzy a návrhu. Novšie metódy odhadu nákladov na softvér, ako napríklad UCP, využívajú prípady použitia (*use cases*), ktoré práve OO návrh ponúka pre popis funkcionality systému.

Tento boom mal však jeden nedostatok. Ako bolo spomenuté na začiatku, metodológií bolo veľa a každá z nich ponúkala vlastné procesy tvorby softvéru. Tu začali vznikať určité nekonzistencie medzi návrhármi, programátormi a ostatnými členmi tímu. Metodológia nemala potrebné páky na určenie nejakej komunikácie medzi členmi celého vývojového tímu. Preto bola potrebná technológia, ktorá by zjednotila jednotlivé procesy vývoja. Táto snaha vyústila k vytvoreniu už vyššie spomínaného jazyka UML. V celej kapitole sa vyskytujú informácie, ktoré som získal preštudovaním literatúry [3].

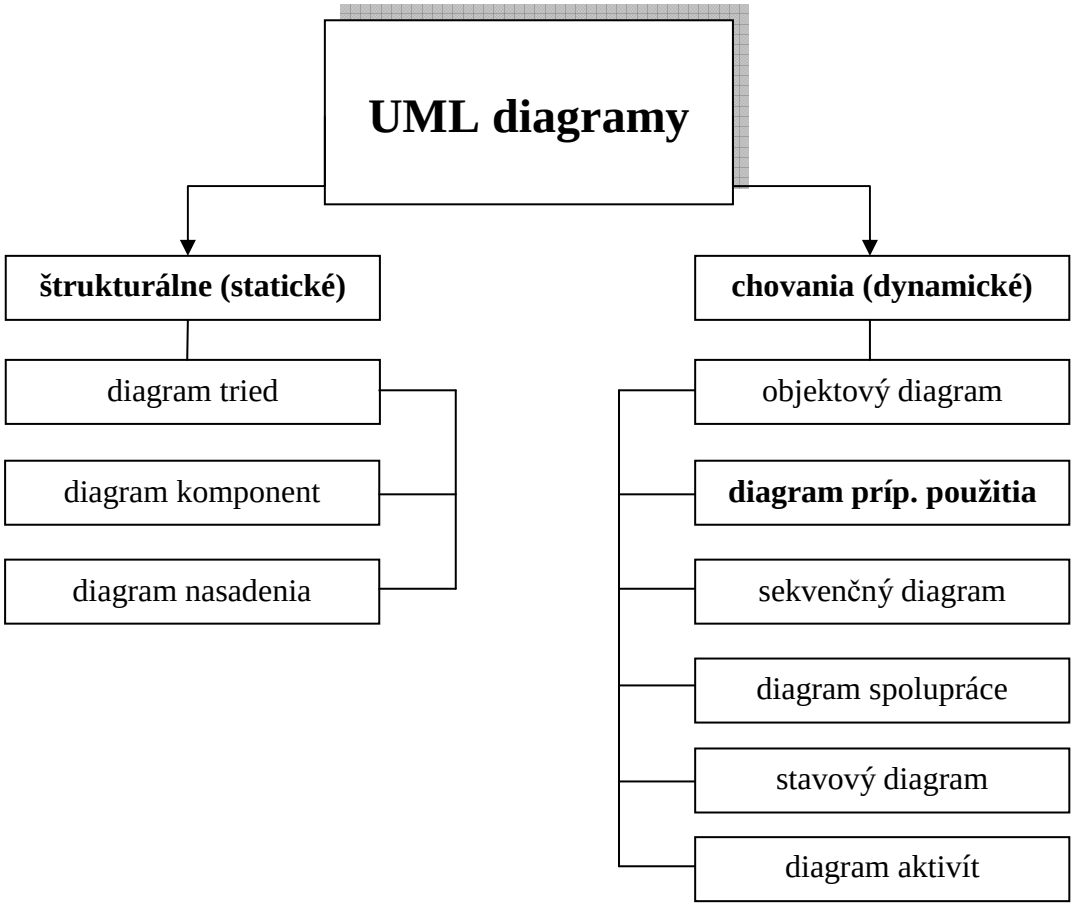
2.1 Jazyk UML a RUP

Tento modelovací jazyk uzrel svetlo sveta v roku 1995. Sám o sebe neponúka konkrétne metódy prístupu k analýze a návrhu softvéru, ale obsahuje prostriedky pre unifikovanú tvorbu modelov, podľa rôznych aspektov tvoreného systému. UML je súčasťou napríklad metodológie RUP, v ktorej slúži na už spomenutú vizualizáciu softvérových súčastí systému.

Aby sme trochu pochopili časť z UML, s ktorej čerpá metóda odhadu nákladov UCP, potrebujeme mať celkový obrázok o čo sa UML snaží. Stavebným kameňom UML je pohľad (*view*), je to názorné pomenovanie vlastnosti, na ktorej je UML postavené. Pohľad je projekcia tvoreného systému podľa jeho konkrétnych aspektov. UML pozná tieto základné pohľady:

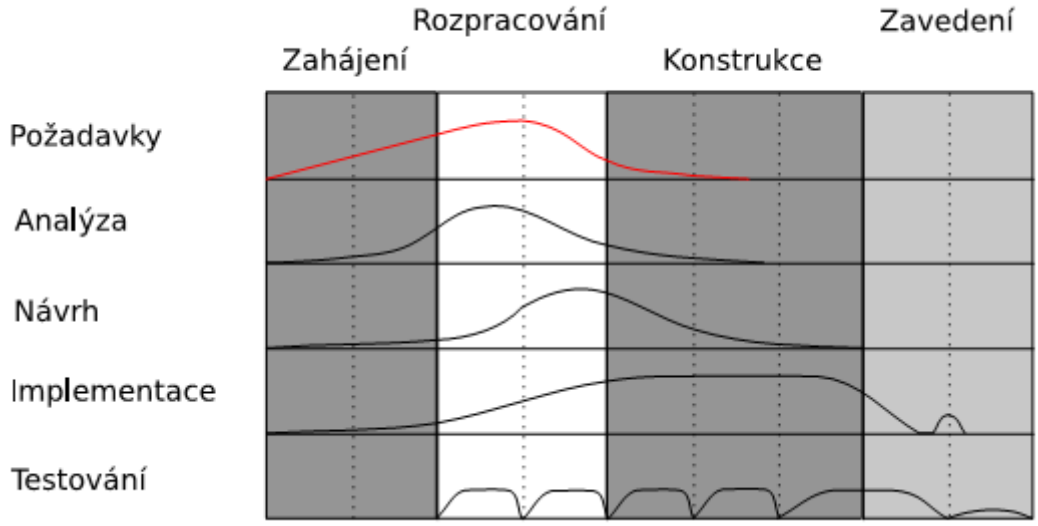
- Štrukturálny pohľad (*structural view*)
- Pohľad chovania (*behavorial view*)

K týmto pohľadom priraďuje a definuje diagramy, ktoré súžia pre jednoduchú vizualizáciu daného aspektu systému. Len pre úplnosť budem ilustrovať na nasledujúcom obrázku 2.1 najpoužívanejšie diagramy, ktoré sa v praxi často pri vývoji softvéru využívajú.



Obrázok 2.1: rozdelenie diagramov v UML

V RUP, ako jediným v praxi používaným modelom životného cyklu (nejedná sa len o model, ale celú metodológiu), sa prirodzene analýza a špecifikácia požiadavkou nachádza v prvej etape životného cyklu softvéru. Na základe obrázku 2.2 si môžeme urobiť predstavu, ako model funguje. Ako bolo spomenuté, pracuje v iteráciách. V prvej iterácii sa spolu s etapou špecifikácie aktivuje etapa analýzy a tak ďalej. To umožňuje rýchlo reagovať na nájdené nedostatky v pokročilejších etapách a modifikovať požiadavky zákazníka. Výstupom etapy špecifikácie je práve diagram prípadov použitia. Než si predstavíme taký bežný diagram prípadov použitia, v nasledujúcej kapitole presnejšie popíšem, čo to vlastne prípad použitia je. Ešte predtým si však názorne vysvetlíme ako RUP pracuje, nebudem však zachádzať do detailov.



Obrázok 2.2: RUP model [3]

Z obrázka je vidieť, štart vývoja v prvej etape (červená krivka), ktorú dosť podobne kopíruje druhá etapa. V mieste osy x, si môžeme predstaviť čas. Fakt, že graf presne nepopisuje mnou vyššie spomenuté rozdelenie etáp, nemá žiadnu váhu, nakoľko presné rozdelenie, ktorého by sa exaktne držali všetky softvérové firmy prakticky neexistuje. V rôznych materiáloch sa môžeme bežne stretnúť s rôznym rozdelením toho istého, teda etáp životného cyklu. Všetky však verne viac či menej, napodobujú rozdelenie, ktoré som uviedol v predchádzajúcich kapitolách. Obrázok len ilustruje charakter RUP. V práci sa nebudem podrobne zaoberať touto metodológiou, len spomenie hlavné prvky, ktorá vychádzajú s predošlého obrázka.

RUP zavádza štyri hlavné fázy vývoja (zahájenie, rozpracovanie, konštrukcia, zavedenie), z ktorých každý je rozdelený do niekoľkých iterácii, ktoré musia spĺňať stanovené kritériá pred vykonávaním ďalšej fáze. Vo fáze zahájenia, vývojári rozpracovávajú rozsah projektu. Druhá fáza, rozpracovanie, spočíva v analyzovaní prostriedkov, ktoré by uspokojili potreby projektu s väčším detailom a definujú jeho architektoniku. Fáza konštrukcie sa skladá s vytvorenia aplikačného dizajnu a následného písania zdrojového kódu aplikácie. V poslednej fázy tím nasadzuje vytvorený softvér zadávateľovi. Veľkou devízou tohto iteračného prístupu je, že po každej iterácii vzniká prototyp, čo je verzia aplikácie, ktorá je plne funkčná a teda umožňuje zadávateľovi reagovať na nedostatky a následne vývojový tím môže reagovať na tieto pripomienky.

2.2 Prostriedky pre analýzu a špecifikáciu

V názve práce sa vyskytuje slovo scenár. Zatiaľ som nikde neuviedol v akom zmysle slovo chápať alebo čo naznačuje. V textoch napr.:[4] a materiáloch pojednávajúcich o vývoji softvéru sa miesto slova scenár stretneme s označením *user story*, teda príbeh užívateľa, jedná sa však o synonymum. Ako bolo naznačené, už spomínané prípady použitia reflektujú funkcie vyvíjaného softvéru. Scenáre majú istú súvislosť práve s prípadmi použitia. Tento fakt vysvetlím na charakterizácii týchto pojmov, porovnaní rozdielov týchto spôsobov a ukážkach reálneho použitia využívaných pri analýze a hlavne špecifikácii systému.

2.2.1 Scenár

Scenár [4] špecifikuje potrebu, požiadavku užívateľa na systém. Zvyčajne je vo forme napísaných viet, bežným jazykom užívateľa resp. zadávateľa, aby každý užívateľ dokázal pochopiť scenár a vedieť čo znamená. Teda sú napísané napríklad v nejakom textovom alebo inom editore (word, excel), v závislosti na projekte. Reálny príklad scenára pre zobrazenie rozvozu pečiva napríklad v pekární, môže byť nasledovný:

„ *Zobraz rozvoz*

> Vodič rozvážajúci pečivo potrebuje zo systému zistiť, aké pečivo a kam ho má dopraviť. Prihlási sa do systému svojím prihlasovacím menom. Ten zobrazí rozpis rozvozu.“

2.2.2 Prípad použitia

Prípady použitia [4] sú podobné scenárom, pretože tak isto popisujú špecifické interakcie medzi užívateľom a systémom. Často však pri písaní prípadov použitia môžeme naraziť na nepresnosti pri komunikácii zákazníka so systémom. Prípady použitia znázorňujú akcie prevádzané užívateľom a spôsob, ako na ne systém reaguje.

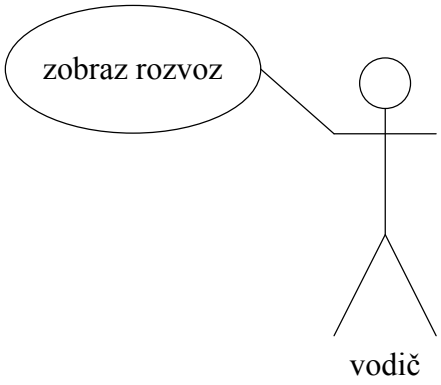
Identifikátor	UC01		
Názov	Prípad použitia: Zobraziť rozvoz		
Popis	Vodič chce zistiť aké pečivo a kam sa má dopraviť		
Priorita	2 – stredná	**Frekvencia**	denne

Vstupné	Vodič je prihlásený do systému	
Výstupné	Zobrazí sa zoznam miest s počtom pečiva	
Užívatelia	vodič	
Hlavný scenár	**Krok**	**Činnosť**
	1	Prípad použitia začína voľbou „zobraziť zoznam pre aktuálny deň“
	2	AK v zozname sú žiadané záznamy:
	3a	Zobrazí sa zoznam miest s adresou a počtom pečiva
	3b	Systém ponúkne možnosť tlače
Alternatívny scenár	**Krok**	**Činnosť**
	1.	Vodič sa môže kedykoľvek vrátiť do hlavnej ponuky
	2.	Vodič môže kedykoľvek opustiť systém
Poznámky	Ďalšie možné poznámky	

Tabuľka 2.2.1.1: Prípad použitia UC01, Zobraz rozvoz

Presne ako ilustruje tabuľka 2.2.1.1 môže vyzerať reálny prípad použitia. Odhad metódou UCP, ktorá vychádza s prípadov použitia, využíva podobne definované prípady použitia. Nie je stanovený žiaden štandard ako by mal prípad použitia vyzerať, ale vžila sa takáto tabuľková forma. Veľmi dôležité je popísanie takzvaných scenárov resp. krokov prípadu použitia ako vidíme v tabuľke 2.2.1.1. Podľa [5] existuje päť úrovní detailov v akom sa popis UC môže vyskytovať. Na tento fakt narazíme aj v kapitole 3 zaoberajúcej sa presne postupom odhadu metódou UCP.

V praxi sa stretávame aj so spomínaným diagramovým spôsobom ilustrovania prípadov použitia. Je to viac všeobecnejší pohľad na jednotlivé časti alebo aj na celý vyvíjaný systém v závislosti na danom projekte. Práca sa však nebude podrobne zaoberať akým spôsobom sa jednotlivé spôsoby špecifikácie riešia a ani bližšie popisovať ich vlastnosti. Len pre úplný prehľad k danému prípadu použitia uvediem jeho podobu pomocou často používaných diagramov prípadov použitia. Ten však bude znázorňovať len jedného aktéra a jednu akciu teda prípad použitia.



Obrázok 2.2.2.1: Use case diagram

V RUP sa diagramy prípadov použitia využívajú pre zachytenie požiadavkou zadávateľa. Jednou z aktivít RUP vo fáze špecifikácie je nájdenie prípadov použitia. Toto hľadanie nie je však jednouché a práve tento faktor ovplyvňuje výsledky pri jednej z metód odhadovania nákladov na softvér. Jeden prípad použitia je teda chápaný, ako jedna funkcia, ktorú systém vykonáva menom daného účastníka, používateľa systému. Každý prípad použitia má svoj názov, jednoznačný identifikátor a špecifikáciu ako môžeme vidieť v tabuľke 2.2.1.1.

2.2.3 Rozdiel medzi scenárom a prípadom použitia

Scenáre zachytávajú potreby. Pri písaní scenára zachytávame holú potrebu užívateľa. Je to niečo, čo užívateľ potrebuje pri jeho dennodennej práci. Prípady použitia zachytávajú chovanie, ktoré systém, softvér špecifikuje, za účelom splnenia týchto potrieb. Softvérový vývojár by mal byť schopný prečítať prípad použitia a získať predstavu, čo softvér potrebuje vykonávať. K tomu práve potrebuje viacej detailov popisujúcich všetky potrebné funkcie. Na druhú stranu scenáre sú jednoduché pre pochopenie zadávateľom. Pri ich písaní sa zameriavame na napísanie niečoho, čo dokáže pochopiť každý. Scenáre sú strohé a vyjadrujú celkovú myšlienku len v niekoľkých vetách. V rôznej literatúre napríklad [5], sa dá stretnúť s popisom prípadov použitia a scenárov do jedeného celku. Záleží od každej softvérovej firmy, ktorá týmto spôsobom postupuje pri vývoji svojich produktov, aký štýl a spôsob písania a unifikácie požiadavkou si zvolí. V práci sú kvôli prehľadu spomenuté rôzne spôsoby, ako sa je možné dočítať v nasledujúcich kapitolách. V prípadoch spomínaných v publikácii [5], je scenár časť prípadu použitia, tvorí jeho telo.

3 METÓDY ODHADU NÁKLADOV NA SOFTVÉR

Z praxe, dostupných materiálov a štúdií a samotného záujmu softvérových firiem, hľadať optimálne najpresnejší spôsob, ako odhadovať náklady spojené s vývojom softvéru vyplýva jeden nemilý fakt. Tým je neustála nepresnosť pri týchto odhadoch. Vytvorenie presného odhadu nákladov na projekt v rannej fázy životného cyklu softvéru je neustále veľkou výzvou. Problémom je, že väčšina manažérov a softvérových inžinierov ba dokonca samotných tímov, odhadujú náklady na projekt pomocou intuície alebo na základe ich vlastných odhadov. Pri najlepšom, odhadujú náklady na základe osobných skúseností, čo môže byt určite prínosom, ale tieto prípady a skúsenosti nie sú nijako metodicky zdokumentované, aby sa dali systematicky uplatňovať na nasledujúcich projektoch. Práve preto vznikajú modely, ktoré manažérom a SW inžinierom ponúkajú spôsob, ako tento problém riešiť.

V práci spomeniem tri metódy, s ktorých každá má špecifické vlastnosti a našla si uplatnenie v praxi. Informácie o prvých dvoch metódach boli čerpané z literatúry [2].

3.1 Odhad metódou COCOMO

COCMO (COnstructiv COst MOdel) je algoritmus, ktorý popísal v roku 1981 Barry Boehm vo svojej knihe [1]. Veľmi často ju nájdeme v materiáloch pod názvom COCOMO 81 (neskôr vznikla nová verzia COCOMO II). Metóda zakladá svoje odhady s počtu napísaných riadkov zdrojového kódu. Je teda potrebné mať informácie o veľkosti zdrojových kódov, z nich sa dá následne odhadnúť čas a cena potrebná na vývoj ak uvážime, že prácnosť, čas a cena sú úmerné veľkosti zdrojového kódu. Pre kalkulovanie o ktoré v odhadoch ide, si musíme tieto vzťahy pomenovať. Vychádzame z úvahy, že úsilie *E* a doba *T*, sú úmerné veľkosti kódu. Platia nasledovné vzťahy:

$$E = a(KLOC)^b$$

$$T = c\,E^d$$

kde,

E – Effort, je úsilie (počet vynaložených človekomesiacov)

T – Time, je doba (čas v mesiacoch)

KLOC – Kilo Lines Of Code, je hlavným identifikátorom veľkosti softvéru (počet riadkov kódu v tisícoch). Symboly *a, b, c, d* sú parametre v závislosti na zvolenom type úrovne

detailu a použitého vývojového typu. Presné hodnoty, ktoré sú stanovené pre daný projekt podľa jeho zaradenie znázorňuje tabuľka 3.1.1 pre základnú úroveň detailu a 3.1.2 pre strednú a pokročilú úroveň na konci tejto podkapitoly. Teraz si tieto úrovne a vývojové typy projektu charakterizujeme.

3.1.1 Úroveň detailu výpočtu

Táto vlastnosť projektu znázorňuje podmienky, za akých softvér vzniká a určuje mieru s akou do odhadu zasahujú faktory okrem počtu riadkov zdrojového kódu.

3.1.1.1 Základná úroveň (basic COCOMO)

Pri takejto úrovni sa počítajú odhady len na základe veľkosti programov, teda s počtu riadkov zdrojového kódu. Toto číslo je vyjadrené na základe KLOC, preto sa jedná len o hrubý odhad nákladov a výsledné hodnoty sú viac menej orientačné.

3.1.1.2 Stredný úroveň (intermediate COCOMO)

Stredná úroveň pridáva k odhadom okrem *KLOC* aj množinu faktor nazvané anglicky *cost drivers*, ktoré zahŕňajú subjektívne ohodnotenie produktových, hardvérových, personálnych a projektových atribútov. Spolu v týchto štyroch skupinách cost driverov pôsobí 15 atribútov ovplyvňujúcich výpočet odhadu nákladov softvéru. To znamená, že odhady stanovené pri tejto úrovni, sa považujú za presnejšie ako v predchádzajúcej úrovni.

3.1.1.3 Pokročilá úroveň (detailed COCOMO)

Na rozdiel od prvých dvoch úrovní, v ktorých sa výpočet aplikoval na celý projekt, v tejto úrovni sa výpočty aplikujú na každú etapy životného cyklu zvlášť. To znamená, že tretia úroveň pridáva k výpočtom v druhej úrovni aj vplyv jednotlivých etáp životného cyklu softvéru.

3.1.2 Vývojové triedy projektov

Ďalším faktorom ovplyvňujúcim odhady nákladov je zložitosť jednotlivých projektov. Tie B. Boehm klasifikoval do troch tried.

3.1.2.1 Organický projekt

Do tejto kategórie by mali byť zaradené malé projekty, rádovo tisíce riadkov kódu (do 50 tisíc), ktoré vyvíjajú menšie tími. Postu vývoja je známy lebo vychádza zo skúseností

s riešením podobných projektov z minulosti. Pre organický typ projektu sú špecifikované len mierne obmedzenia rozhrania. Väčšinou v projekte nie je zahrnutý špeciálny hardvér a požiadavky na inováciu programu nie sú prioritou. Príkladmi takýchto projektov sú rôzne vedecko – technické aplikácie, jednoduchý plánovací systém, interpret programovacieho jazyka a podobné.

3.1.2.2 Viazaný projekt

Presný opak organického typu projektu. Jedná sa o veľké, rozsiahle projekty, rádovo v státisícoch až miliónoch riadkov zdrojového kódu. Softvér musí pracovať bezchybne za veľmi prísnych podmienok čo sa týka jeho výkonnosti a odozvy. Vývojový tím pracuje so zložitými softvérovými aj hardvérovými systémami. Sú kladené striktné požiadavky na jeho spoľahlivosť, prenositeľnosť, modifikovateľnosť a termíny nasadenia. Zvyčajne sa tu tím stretáva s úplne novými problémami, s ktorými sa pri vývoji ešte nestretol. Tím je zložený s niekoľkých menších tímov, zvyčajne odborníkov na analýzu a návrh, programátorov, testerov, ktorí pracujú súbežne inak by sa vývojový proces predlžoval a viedol k viacerým zmenám. Preto bývajú projekty tohto typu napríklad vojenské systémy, systémy riadenia kozmických lodí, lietadiel, rozsiahle operačné systémy a podobné druhy softvérov.

3.1.2.3 Bezprostredný projekt

Ostatné typy projektov, ktoré nezapadajú ani do jednej s týchto tried sú zaradené práve sem. Jedná sa o stredne rozsiahle projekty rádov desaťtisíce riadkov zdrojového kódu (do 300 tisíc). Tím je zložený zo skúsených aj neskúsených vývojárov a riešia nie príliš známe úlohy, často býva zadanie zložité. Obmedzenia na rozhranie sú zreteľné a závislosť od špeciálneho hardvéru je na strednej úrovni. Patria sem menšie operačné systémy, bežné transakčné systémy, skladové aplikácie strednej zložitosti a tak podobne.

Vývojový typ	a	b	c	d
Organický	2.4	1.05	2.5	0.38
Bezprostredný	3.0	1.12	2.5	0.35
Viazaný	3.6	1.2	2.5	0.32

Tabuľka 3.1.1: Hodnoty parametrov pre základný model

Vývojový typ	a	b	c	d
Organický	$3.2F_c$	1.05	2.5	0.38
Bezprostredný	$3.0F_c$	1.12	2.5	0.35
Viazaný	$2.8F_c$	1.20	2.5	0.32

Tabuľka 3.1.2: Hodnoty parametrov pre stredný a viazaný model

3.1.3 Korelačný faktor

Z tabuľky 3.1.2 pre parameter *a*, vidíme pri konkrétnej číselnej hodnote aj ďalšiu premennú F_c, nazýva sa korelačný faktor a vyjadruje zohľadnenie už spomenutých pätnástich atribútov cost driverov. Tieto atribúty nadobúdajú jednu zo šiestich hodnôt v stupnici veľmi nízky, nízky, normálny, veľký, veľmi veľký, extrémne veľký, ako určuje tabuľka 3.1.3. Celkový korelačný faktor je súčinom všetkých pätnástich atribútov.

$$F_c = \prod_{k=1}^{15} F_k$$

Atribúty	Hodnotenie					
	Veľmi nízke	**Nízke**	**Normálne**	**Veľké**	**Veľmi veľké**	**Extrémne veľké**
Softvérového produktu						
RELY	0.75	0.88	1.00	1.15	1.40	
DATA		0.94	1.00	1.08	1.16	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Hardvérové atribúty						
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT		0.87	1.00	1.15		
TURN		0.87	1.00	1.07	1.15	
Atribúty vývojového tímu						
ACAP	1.46	1.19	1.00	0.86	0.71	
AEXP	1.29	1.13	1.00	0.91	0.82	
PCAP	1.42	1.17	1.00	0.86	0.70	
VEXP	1.21	1.10	1.00	0.90		
LEXP	1.14	1.07	1.00	0.95		
Atribúty metód vedenia tímu						
MODP	1.24	1.1	1.00	0.91	0.82	
TOOL	1.24	1.1	1.00	0.91	0.83	
SCED	1.23	1.08	1.00	1.04	1.1	

Tabuľka 3.1.3: Hodnoty atribútov ovplyvňujúce korelačný faktor

3.2 Odhad metódou funkčných bodov

Anglický názov metódy je *Function point analysis* (FPA) bola vyvinutá v roku 1983 A. Albrechtom. Rieši nedostatok COCOMO, keďže pre svoje odhady nepotrebuje vedieť počty riadkov kódu, ale zameriava sa na funkčné požiadavky systému a štruktúru dát. Týmto spôsobom sa odhad nákladov na vývoj softvéru rieši viac objektívnejšie. Vývojom metódy sa zaoberá organizácia IFPUG (*International Function Point User Group*).

Aby sa dosiahlo určitej objektívnosti, je potrebné do odhadov zatiahnuť nie len vývojový tým ale aj požiadavky užívateľa. FPA poskytuje práve mechanizmus, ktorý poskytuje tejto dvojici využiť definovanie funkčných požiadavkou. FPA umožňuje

stanoviť základné charakteristiky vyvíjaného softvéru, ktoré označujeme ako neupravené funkčné body, anglicky *Unadjusted Function Points* (UAF). Výpočet týchto neupravených funkčných bodov sa skladá z klasifikácie počtu požadovaných funkcií na systém do piatich tried, ktoré predstavujú funkčnosť systému (funkčné typy) a sú rozdelené do dvoch skupín. Prvú skupinu tvoria dátové funkcie, druhú skupinu transakčné funkcie. Takto klasifikované funkcie podľa FPA [7] majú tri druhy zložitosti (jednoduchá, priemerná, veľká) a na ich základe sa každej požadovanej funkcii priradí váha ako znázorňuje tabuľka 3.2.1.1. Súčtom týchto hodnôt potom dostávame neupravené funkčné body. K neupraveným funkčným bodom sa prinásobý ešte jedna hodnota, tzv. *Value Adjustment Factor* (VAF). Tento faktor na základe štrnástich atribútov, ktoré definujú systém o niečo špecifickejšie, upraví výpočet neupravených funkčných bodov na upravené funkčné body. S nich vyplýva celkový odhad nákladov na softvér (odhad LOC a odhad doby trvania vývoja).

3.2.1 Neupravené funkčné body

Ako bolo naznačené vyššie, k výpočtu neupravených funkčných bodov treba definovať a rozdeliť požadovanú funkcionalitu systému do piatich tried funkcií. Tieto triedy sú rozdelené do dvoch kategórií. Kategória dátových funkcií a kategória transakčných funkcií. V nasledujúcej podkapitole si tieto triedy bližšie charakterizujeme.

3.2.1.1 Dátové funkcie

Sú špecifikované dvomi triedami funkcií resp. súbormi.

Prvá definuje počet vnútorných logických súborov, anglicky *Internal Logical Files* (ILF). ILF reprezentujú dáta, ktoré sú skladované a udržiavané v rámci hraníc systému, ktorý vyvíjame. V podstate sú sem zaradené dátové súbory, na ktorých bude vyvíjaný systém postavený, s ktorých bude čerpať. Napríklad tabuľky, v prípade použitia nejakého relačného databázového systému, alebo aj obyčajné dátové súbory bez definície nejakých vzájomných relácií.

Druhá definuje počet súborov vonkajšieho rozhrania, anglicky *External Interface Files* (EIF). V tejto triede sa nachádzajú súbory, ktoré tak isto tvoria logické časti ako v prvom prípade, ale dáta samotné sú len naším systémom nejakým spôsobom využívané, nie spracúvané. Na tieto dáta systém len nejako odkazuje, nie sú ním priamo udržiavané ako v prvom prípade.

3.2.1.2 Transakčné funkcie

Sú charakterizované tromi triedami.

Externý vstup, anglicky *External Input* (EI), sú funkcie, ktoré umožňujú užívateľovi aplikácie udržiavať súbory s ktorými aplikácia pracuje (už spomínané ILF) formou pridávania, mazania a ich editácie. Je to teda určitý proces, transakcia, ktorá užívateľovi dovoľuje istú zmenu v rámci rozhrania systému.

Externý výstup, anglicky *External Output* (EO) je chápaný ako proces, ktorý preposiela dáta, ktoré opúšťajú hranice aplikácie. Výstupy sú často rôzne hlásenia, ktoré sa ďalej spracúvajú alebo poskytujú podklad pre ďalšie rozhodovanie. Príklad externého výstupu sú napríklad výstupy dát na obrazovku alebo nejaké externé zariadenia (často tlačiareň a pod.).

Externý dotaz, anglicky *External Inquery* (EQ) v tomto prípade hovoríme získavaní (dotazovaní) sa po konkrétnych informáciách v súboroch sprostredkovaných našou aplikáciou, nejedná sa teda o nejakú manipuláciu s dátami resp. súbormi.

Funkčné typy	**Zložitosť**			
	Jednoduchá	Priemerná	Veľká	Celkom
ILF	__ x 3 +	__ x 4 +	__ x 6 =	__
EIF	__ x 4 +	__ x 5 +	__ x 7 =	__
EI	__ x 3 +	__ x 4 +	__ x 6 =	__
EO	__ x 7 +	__ x 10 +	__ x 15 =	__
EQ	__ x 5 +	__ x 7 +	__ x 10 =	__
Neupravené funkčné body celkom (UAF)				__

Tabuľka 3.2.1.1: Výpočet neupravených funkčných bodov

Potom doplnením počtov jednotlivých rozdelených funkcií získame podľa tohto schéma neupravené funkčné body. Konštanta ktorou tieto počty násobíme sa nazýva váha, ta ako vidíme je určená špecificky pre každú triedu funkcií a ich zložitosti zvlášť. S rôznych publikácií a článkov [7] týkajúcej sa používania tejto metódy v praxi sa javí vhodné výpočet ukončiť a nepokračovať úpravou funkčných bodov. Ja však od prezentujem teoreticky kompletný proces výpočtu. Tento dôvod bude popísaný v záverečných

kapitolách, ktoré sa budú týkať zhodnotenia a porovnania jednotlivých metód z hľadiska použiteľnosti v praxi a ich nedostatkov.

3.2.2 Upravené funkčné body

Postupom času sa prišlo na fakt, že náklady projektu ovplyvňuje nie len jeho funkcionalita a štruktúra dát, ako prezentuje výpočet neupravených funkčných bodov, ale aj iné faktory. Podľa [7] výpočet upravených funkčných bodov ovplyvňuje už spomínaná hodnota VAF. Je to faktor, ktorý pozostáva zo štrnástich všeobecných charakteristík systému, anglicky *General System Characteristics* (GSCs). Každému zo štrnástich charakteristík je priradená váha v intervale od 0 (málo) do 5 (veľa). Po sčítaní váh všetkých štrnástich charakteristík, dostávame hodnotu vyjadrujúcu celkový stupeň vplyvu faktorov na aplikáciu, anglicky *Total Degree of Influence*, (TDI). Zoznam štrnástich faktorov predstavuje:

1. Dátová komunikácia
2. Distribuované spracovanie dát
3. Výkonnosť
4. Plné využitie konfigurácie
5. Rýchlosť spracovania transakcií
6. Priamy vstupy dát
7. Efektivita koncového užívateľa
8. Priama oprava
9. Zložitosť spracovania
10. Znovupoužitelnosť
11. Jednoduchá inštalácia
12. Jednoduché použitie
13. Možnosti nasadenia
14. Možnosť zmeny

Pre výpočet VAF teda platí vzťah:

$$VAF = (TDI * 0.01) + 0.65$$

a výpočet upravených funkčných bodov je definovaný ako:

$$AFP = UAF * VAF$$

3.2.3 Využitie funkčných bodov pre odhady nákladov

Konečne sme sa dopracovali k akémusi ohodnoteniu nášho vyvíjaného systému, máme spočíta jeho funkčné body, ktoré predstavujú normalizovanú metriku softvérového projektu, ktorá meria aplikačnú oblasť a neskúma tú technickú. Ak by sme chceli odhadovať LOC, ktorý nám môže urobiť predstavu o veľkosti aplikácie alebo pre ďalšie odhady doby vývoja napríklad aj inými metódami ako spomínané COCOMO, tak treba zohľadniť programovací jazyk v ktorom aplikácia bude vytvorená. Pre konkrétny jazyk existuje pomer koľko riadkov v ňom zaberie napísanie jedného funkčného bodu. Tieto podrobnosti však prekračujú hranice mojej práce. Viacej ohľadom metódy FPA je možno získať z materiálu [7], kde sú prezentované aj príklady výpočtov pri rôznych projektoch a rôzne druhy odhadov, ktoré nám výpočet funkčných bodov umožňuje. Uvediem len výpočet odhadu doby trvania vývoja *T*, na to nám poslúži vzorec:

$$T = FP^{0.4}$$

3.3 Odhad Use case points

Dostali sme sa do podkapitoly, pre ktorú sme budovali prostredie celé dve predošlé kapitoly. Kapitola 1 a kapitola 2 v ktorých som si pripravoval všetky prostriedky a zázemie, aby som túto metódu dobre uviedol. Informácie prezentované v tejto kapitole vychádzajú z materiálov [8]. Tak ako v týchto kapitolách bolo naznačované metóda odhadu nákladov *Use case points*, vychádza veľmi úzko s objektovo orientovaného pohľadu na celý proces vývoja. Hlavnú časť však nepochybne tvoria už spomínané scenáre a prípady použitia anglicky *use ceses*. Tie sa vyskytujú v rôznych podobách. Jednou s podôb sú tzv. formálne prípady použitia ako ukazuje tabuľka 2.2.1.1, ktoré poskytujú detailné informácie, ktoré však pre odhad nákladov nie sú až tak dôležité. Ďalej ako samotné scenáre, neformálne prípady použitia, ktorých vytvorenie nie je časovo náročné alebo použitím UML, v ktorom sa vyvíjaný systém namodeluje do diagramu prípadov použitia prezentované obrázkom 2.2.2.1. Máme teda na výber niekoľko možností, no pre výpočty odhadu nákladov je zjavne z časového hľadiska vhodná prezentácia pomocou neformálnej podoby prípadov použitia. Na tento spôsob využitia OO princípov analýzy a návrhu pre odhady softvérových projektov, prišiel v deväťdesiatich rokoch Gustav Karner, s ktorého sa vyvinula metóda UCP vlastnená teraz firmou Rational Software.

3.3.1 Proces odhadovania nákladov

V tejto časti si podrobne rozoberieme všetky dôležité prvky uplatňované v tejto metóde, ktoré systém popisujú a s ktorých sa následne odhad nákladov na projekt bude vytvárať. Ak chceme pre odhad nákladov použiť UCP, je potrebné si ujasniť pár základných prvkov, vyplávajúcich z požiadavkou zadávateľa. Javí sa ako dobrý spôsob nadefinovať, stavebné kamene UCP. Proces by mohol teda začať nižšie popísaným spôsobom.

3.3.1.1 Definovanie primárnych užívateľov

V podstate sa jedná o výpis všetkých užívateľov, ktorí budú náš systém priamo využívať, ktorých požiadavky sa systém snaží implementovať. Neskôr sa títo užívatelia transformujú do aktérov vystupujúcich v prípadoch použitia, samozrejme toto neplatí úplne. Prípady použitia modelujú aj iné druhy aktérov, nemusia to byť a často ani nie sú len fyzický užívatelia.

3.3.1.2 Definovanie gólov užívateľov

Použitie tejto metódy teda silno závisí od požiadavkou užívateľa, tieto požiadavky nazveme góly, sú to vlastne všetky možné interakcie užívateľa z naším systémom, funkcionalita, ktorú vyžaduje. K popisom týchto gólov využívame prípady použitia.

3.3.1.3 Definovanie prípadov použitia

Pri tomto bode sa môže zdať, že to je predsa jednoduché napísať pár prípadov použitia, ku každému požiadavku a pokračujeme ďalej k výpočtom odhadov. No tu však vzniká priestor pre problémy. Na to, aby bol systém týmito prípadmi použitia popísaný správne ideálne bez chýb a presne, musí byť každý jeden prípad napísaný na rovnakej úrovni detailu v rámci daného projektu. Ako bolo spomenuté v podkapitole 2.2.2 podľa knihy [5] existuje 5 úrovní, popisu prípadov použitia. Pre odhad nákladov je dôležité, aby pre daný vyvíjaný systém boli všetky prípady použitia napísané na rovnakej úrovni detailu. Práve tento faktor a táto časť, definovanie a vytváranie prípadov použitia na základe požiadavkou sa výrazne podpisuje na kvalite odhadov, preto často tento fakt býva klasifikovaný ako výrazná nevýhoda metódy UCP. Viac o nevýhodách a rôznych porovnaniach a prípadných vylepšeniach však až na záver tejto práce. Práca sa nebude špeciálne venovať problematike písania resp. vytvárania prípadov použitia a ich správnosti, pre informácie o tejto téme doporučujem [5].

3.3.1.4 Odhadov nákladov na projekt

Nasleduje samozrejme samotný matematický model, a spôsob, akým sa dá odhadnúť náklady na takto popísaný systém. Doposiaľ som nevysvetlil čo to vlastne sú body prípadov použitia, ako môžeme voľne preložiť z anglického originálu *use case points*. Jedná sa akúsi univerzálnu jednotku pomocou ktorej meriame, koľko úsilia je potrebné vynaložiť pri vývoji daného projektu na základe toho, čo všetko vyvíjaný softvér umožní užívateľovi robiť.

Každý projekt, ktorému odhadujeme jeho nákladovosť, je v UCP charakterizovaný týmito faktormi:

- Technický faktor implementácie (iné ako funkčné požiadavky)
- Faktor vplyvu okolia na vyvíjaný softvér
- Množstvo a komplexnosť prípadov použitia popisujúcich systém
- Množstvo a komplexnosť aktérov systému

Aby sme dosiahli výsledku, v našom prípade teda nejaký počet bodov prípadov použitia, komplexnosť prípadov použitia a komplexnosť aktérov popíšeme váhami a následne tieto váhy upravíme tak, aby odpovedali technickým a environmentálnym faktorom ovplyvňujúcim (popisujúcim) náš projekt. Na prvý pohľad UCP pôsobí rovnako ako FPA, s časti je to pravda. Ako som vyššie napísal jedná sa vlastne o váhové ohodnotenie a následne výpočet neupravených bodov, ktoré sa finálne na základe spomenutých faktorov upravia do konečnej podoby s ktorej sa náklady na softvér určujú. Teraz presnejšie popíšem, podľa akých faktorov sa jednotlivé projekty môžu odlišovať. A s čím všetkým sa pri ohodnocovaní komplexnosti aktérov a prípadov použitia môžeme stretnúť.

3.3.2 Technické faktory implementácie

Podľa Karnera existuje trinásť technických faktorov, ktorý daný projekt špecifikujú. Celkovo sa jedná o faktory, ktoré nemajú nič spoločné s požadovanou funkcionalitou. Pri odhadovaní nákladov na softvérové projekty existuje množstvo premenných, ktoré výrazne každý jeden projekt formujú resp. odlišujú. Tento fakt treba pri odhadoch zohľadniť nakoľko sa snažíme odhady mať čo najspoľahlivejšie. Faktory však nemajú rovnaký vplyv na projekt, preto každý jeden z nich má vlastnú číselnú konštantu vyjadrujúcu vplyv faktoru na projekt. Táto konštanta sa znásobí s nami priradeným hodnotením daného

faktoru, podľa konkrétneho projektu. V UCP teda hovoríme o týchto technických faktoroch:

Faktor	Názov (multiplikátor)	Popis
T1	Distribuovaný systém (2)	možnosť aplikácie byť distribuovaná/centralizovaný, čím väčšia schopnosť rozdelenia, tým väčšie ohodnotenie
T2	Časová odozva systému (1)	ak je rýchla odozva aplikácie dôležitým faktorom, priradíme väčšie ohodnotenie
T3	Efektivita pre užívateľa (1)	jednoduché pre ovládanie a používanie koncovým užívateľom, znamená väčšie ohodnotenie
T4	Komplexné spracovanie aplikácie (1)	ak implementuje komplexné, náročné algoritmy, zvyšujeme ohodnotenie
T5	Znovupoužiteľnosť zdrojového kódu (1)	čím viac je aplikácia schopná použitý kód znovu použiť, tým je ohodnotenie väčšie
T6	Náročnosť inštalácie (0,5)	ak je dôležité zameriavať sa na jednoduchosť inštalácie, zvyšuje sa ohodnotenie
T7	Použiteľnosť (0,5)	ak je dôležité zameriavať sa na ovládanie aplikácie, zvyšuje sa ohodnotenie
T8	Multiplatformnosť (2)	podporuje nasadenie na rôznych platformách, čím viac platform podporuje, tým je ohodnotenie vyššie
T9	Možnosť zmeny (1)	ak požaduje zákazník v budúcnosti zmeny v aplikácii zvyšuje sa ohodnotenie
T10	Veľká súbežnosť (1)	ak aplikácia rieši problémy so súbežným spracovaním dát, používanie zámkov, ohodnotenie je väčšie
T11	Zabezpečenie špeciálnej bezpečnosti (1)	vlastné požiadavky na zvýšenú bezpečnosť, čím väčší dôraz je kladený, tým je ohodnotenie väčšie
T12	Závislosť od kódu tretích strán (1)	množstvo závislostí od používania zdrojových súborov tretích strán znamená vyššie ohodnotenie
T13	Školenie užívateľov (1)	ak je požadované školenie užívateľov, zvyšuje sa aj ohodnotenie

Tabuľka 3.3.2.1: Technické faktory tvoriace projekt pri UCP metóde

3.3.2.1 Výpočet TCF

Pre každý faktor s tabuľky 3.3.2.1 sa potom priradí jeho ohodnotenie v intervale 0 až 5. Toto ohodnotenie predstavuje náročnosť daného faktoru pre daný projekt. Nami takto ohodnotený faktor sa znásobí príslušným multiplikátorom, až prejdeme všetkých trinásť, jednotlivé medzi súčiny sčítame (sumu si nazveme napríklad TFaktor) a podľa nasledujúceho vzťahu získavame technický faktor komplexnosti, *Technical Complexity Factor* (TCF).

$$TCF = 0.6 + (0.01 * TFaktor)$$

kde,

M_i^T – multiplikátor pre technický faktor

O_i^T – ohodnotenie pre technický faktor

$$TFaktor = \sum_{i=1}^{13} M_i^T * O_i^T$$

Týmto spôsobom sme vykonali technickú analýzu projektu, ako určuje metóda UCP. Podobným spôsobom sa prevedie analýza vplyvu faktorov prostredia, ktorý projekt špecifikujú.

3.3.3 Faktory vplyvu okolia na vyvíjaný softvér

Jedná sa o faktory, ktoré popisujú schopnosti vývojového tímu, čo tvorí dôležitú časť úspechu. Táto vyspelosť tímu sa pri odhadov UCP radí medzi faktory prostredia v ktorom vyvíjaný softvér bude vznikať. Vystupuje tu osem environmentálnych faktorov ako predstavuje tabuľka 3.3.3.1.

Faktor	Názov (multiplikátor)	Popis
E1	Znalosť projektu (1,5)	skúsenosť tímu s podobným druhom projektu
E2	Skúsenosti s aplikáciou (0,5)	relevantné v prípade zmien na už vytvorenej aplikácií, čím väčšia skúsenosť tým väčšie ohodnotenie
E3	Skúsenosť s OO programovaním (1)	čím skúsenejší tím v OOP tým väčšie ohodnotenie
E4	Schopnosti analyzovať (0,5)	schopnosť analytika, ktorí sa stará o požiadavky od užívateľov, čím schopnejší tým väčšie ohodnotenie
E5	Motivácia (1)	čím viac je tím motivovaný tým sa zväčšuje aj ohodnotenie
E6	Stabilita požiadavkou (2)	meniace sa požiadavky predstavujú predlžovanie vývoja, čím viacej zmien tým väčšie ohodnotenie
E7	Práca na polovičný úväzok (-1)	pracovníci, ktorí nie sú do projektu zapojený ako ich hlavný cieľ majú negatívny efekt, predstavujú väčšie ohodnotenie
E8	Náročnosť programovacieho Jazyka (-1)	Čím náročnejší jazyk pre tím, tým väčšie ohodnotenie (náročnosť vzhľadom k tímu, neexistuje všeobecné pravidlo, čo je, nie je náročné)

Tabuľka 3.3.3.1: Environmentálne faktory tvoriace projekt pri UCP metóde

3.3.3.1 Výpočet EF

Podobne ako pri výpočte TCF v predošlej časti, postupujeme rovnako aj pri výpočte tzv. environmentálneho faktoru, anglicky *Environmental Factor* (EF). Na začiatku ohodnotíme jednotlivé faktory hodnotou z intervalu 0 až 5. Túto hodnotu znásobíme s daným multiplikátorom. Nakoniec spočítam medzi výpočty a dostávam hodnotu, ktorú si nazvem EFaktor. Nasledovný vzťah prezentuje finálny výpočet vonkajších vplyvov na projekt.

$$EF = 1{,}4 + (-0{,}03 * EFaktor)$$

kde,

M_i^E – multiplikátor pre environmentálny faktor

O_i^E – ohodnotenie pre environmentálny faktor

$$EFaktor = \sum_{i=1}^{8} M_i^E * O_i^E$$

Čím väčší je EFaktor, tým je nižší celkový výsledok EF. Je to prirodzené, čím skúsenejší a lepší tím pracuje na projekte, tým menej námahy musia vynaložiť pri vývoji projektu.

3.3.4 Množstvo a komplexnosť prípadov použitia popisujúcich systém

Odhad metódou UCP je ako už bolo veľa, krát spomenuté, založený na prípadoch použitia. Doteraz som uviedol len faktory, ktoré odhad správne modifikujú, ale nie sú nosným kameňom odhadov. Každý s prípadov použitia prispieva k odhadom nákladov na softvérový projekt. Je prirodzené, že prípady použitia sa jeden od druhého rôznia. Stretávame sa tu tak isto s komplexnosťou jednotlivých prípadov použitia. Aby sme vyjadrili a charakterizovali prípady použitia podľa metódy UCP si vystačíme s rozdelením prípadov použitia do troch skupín, podľa ich komplexnosti.

- jednoduchý (*simple*)
- priemerný (*average*)
- komplexný (*complex*)

Toto delenie vzniká na základe počtu transakcií, ktoré prípad použitia obsahuje. Tu sa dostávame ku skrytému významu slova scenár. Scenár v tomto kontexte predstavuje práve sled interakcií, transakcií. Prípad použitia je teda kolekciou scenárov. Hlavný smer resp. postupnosť krokov v prípadoch použitia, vytvára vlastne jednoduchý scenár, v ktorom sa na konci dosiahne požadovaného cieľa. Existuje aj alternatívna cesta, môže ich byť niekoľko, ktoré vyjadrujú prípad, kedy sa požadovaného gólu nedosiahne priamo bez komplikácií, ako to je v prvom prípade. Všeobecne však neplatí, že počet krokov v prípade použitia je rovný počtu transakcií. Prípad použitia sa môže napríklad skladať s 5 krokov, ale bude sa jednať len o 2 transakcie a podobne. Transakcia podľa [8] je vnímaná ako výmena, interakcia, v ktorej ako prvý užívateľ niečo vykoná a následne mu náš systém odpovedá. Tabuľka 3.3.4.1 uvádza prehľad, akým spôsobom sa jednotlivé prípady použitia klasifikujú a ohodnocujú.

Trieda	Počet transakcií / scenárov	Ohodnotenie
jednoduchý	do 3	5
priemerný	4 až 7	10
komplexný	viac ako 7	15

Tabuľka 3.3.4.1: Klasifikácia prípadov použitia

Takto klasifikované a ohodnotené prípady použitia teraz znásobíme s ich príslušným multiplikátorom podľa tabuľky 3.3.4.1 a medzivýsledky sčítame. Dostaneme váhu prípadov použitia. Zavediem označenie s anglického prekladu *Unadjusted Use Case Weight* (UUCW). Toto číslo budem ďalej potrebné, k finálnym výpočtom UCP tak ako už spomínané technické a environmentálne faktory. Podobným spôsobom vypočítam aj váhy aktérov, presne ako popisuje nasledujúca kapitola, ešte však uvediem vzťahy, ktoré tento výpočet unifikovane znázornia.

$$UUCW = \sum_{i=1}^{3} N_i^{UC} * O_i^{UC}$$

kde,

N_i^{UC} – počet prípadov použitia pre danú triedu

O_i^{UC} – ohodnotenie prípadu použitia v danej triede

3.3.5 Množstvo a komplexnosť aktérov systému

Podobne ako prípady použitia, tak aj aktéri vystupujúci v analýze systému, sú nenahraditeľnou častou pri odhadoch nákladov na softvérový projekt. Tak ako je zrejmé s predchádzajúcej kapitoly, ani všetci aktéri nebudú „na rovnakej úrovni“. Len pre objasnenie, aktéri vystupujúci v tejto analýze sú samotný užívatelia nášho systému, ale veľmi často sa stáva, že každý si pod slovom užívateľ predstaví fyzicky nejakého človeka. Užívateľom môže a často býva nejaký ďalší systém, všeobecne neživá bytosť. Preto práve vznikla a je potrebná analýza komplexnosti aktérov systému, aby sme pri odhadoch vedeli presnejšie určiť náklady spojené s týmto dôležitým faktom akým sú užívatelia systému. Metóda podľa [6] užívateľov rozdeľuje do troch skupín ako prípady použitia.

- jednoduchý (*simple*)
- priemerný (*average*)
- komplexný (*complex*)

Jednoduchý užívatelia sú všetci tí, ktorí komunikujú zo systémom pomocou dobre preddefinovaného programového rozhrania, API z anglického *Application Programming Interface.* Tu UCP pokrýva možnosť, že užívateľom nemusí byť výhradne nejaká osoba, v tomto prípad sa zväčša bude jednať o nejaký iný systém, ktorý s naším softvérom komunikuje. Jedná sa teda o to, že títo aktéri komunikujú so softvérom pomocou špecifického, dobre definovaného mechanizmu. Toto rozhranie môže byť prezentované formou volania vzdialených procedúr RPC *Remote Procedure Call*, DLL *Dynamic Link Library* a podobne.

Priemerný užívatelia sú všetci tí, ktorý vystupujú ako fyzické osoby a komunikujú pomocou dobre špecifikovaného rozhrania alebo systémy komunikujúce cez sofistikovanejšie API. V prvom prípade sú tu osoby, ktoré používajú na komunikáciu napríklad príkazovú riadku *Command Line Interface* (CLI). V druhom napríklad systémy komunikujúce cez TCP/IP alebo využívajúci FTP, HTML, SOAP a podobné sofistikovanejšie protokoly.

Komplexní užívatelia komunikujú už napríklad cez grafické rozhranie, *Graphical User Interface* GUI. Všeobecne tu patria užívatelia používajúci spôsoby komunikácie širšieho významu a poskytujú viacej možností, funkcií a spôsobov než sú dostupné bežnými jednoduchými protokolmi. Napríklad AJAX ponúka určite viacej funkcií a možností, ktoré sú pre neho fundamentálne ako nejaký strohý a prostý protokol. Užívateľa komunikujúci cez vyspelé a na funkcionalitu bohaté HTML rozhranie, sú tak isto radení ako komplexní užívatelia [8]. Ako názov napovedá jedná sa o ako veľmi komplexné rozhranie užívateľ bude používať. Viacej ohľadom správnosti a spôsobu klasifikácie sa budem venovať až k záverečným porovnaniam a hodnoteniam prípadne vylepšeniam metód.

Tabuľka 3.3.5.1 prehľadne prezentuje ohodnotenie jednotlivých typov aktérov vystupujúcich v systéme. Výpočet váhy užívateľov z anglického *Unadjusted Actor Weight* (UAW) sa riadi nasledovným vzťahom.

$$UAW = \sum_{i=1}^{3} N_i^A * O_i^A$$

kde,

$N_i^A -$ počet aktérov v danej triede

$O_i^A -$ ohodnotenie aktérov danej triedy

Trieda	Ohodnotenie
jednoduchý	1
priemerný	2
komplexný	3

Tabuľka 3.3.5.1: Rozdelenie typov aktérov s ich ohodnotením

3.3.6 Výpočet odhadu nákladov na projekt

S takto definovaných častí nám zostáva všetky správne spojiť a tým prejsť k finálnym výpočtom. S predošlých výpočtov teda poznáme neupravené hodnoty prípadov použitia, aktérov a potom dve hodnoty faktorov, ktoré tieto výpočty upravia. Nasledujúci vzťah ešte zachytáva výpočet neupravených bodov prípadov požitia, *Unadjusted Use Case Points* (UUCP). Tieto neupravené body pomocou TCF a EF upravíme na finálny výsledok.

$$UUCP = UUCW + UAW$$

$$UCP = UUCP * TCF * EF$$

Týmto spôsobom máme spočítané celkový počet bodov prípadov použitia. Teraz nastáva čas kedy tieto body treba previesť na časový odhad. Lebo hodnota sama o sebe neurčuje nejaký časový údaj, len veľkosť v jednotke bodov prípadov použitia. Existuje niekoľko prístupov. Podľa Karnera, autora tejto metódy sa na vytvorenie jedného bodu prípadu použitia spotrebuje 20 hodín. Časom pri rôznych štúdiách sa však ukázalo, že táto hodnota nemusí všeobecne platiť pre všetky vývojové tímy a projekty.

Podľa práce pána Kirstena Ribu [9] založenej na pôvodnej práci Karnera vyplýva, že tento čas sa môže líšiť od 15 do 30 hodín na jeden bod prípadov použitia.

Iný prístup prezentovaný v [11] navrhuje spočítať počet environmentálnych faktoru v prvých šiestich faktoroch, ktoré sú ohodnotené vyššie ako 3 a počet environmentálnych faktorov v 7 a 8, ktoré sú menšie ako 3. Ak je tento počet menší alebo rovný 2, počítame s 20 hodinami na jeden bod. Ak je počet ako 3 alebo 4, počítame s 28 hodinami na jeden bod. No a ak je počet väčší ako 4, treba zvážiť či sa vôbec do projektu púšťať, nakoľko s tohto čísla vyplýva, že existuje príliš veľa environmentálnych faktorov, ktoré sú proti nášmu projektu, jednoducho treba počať, kým sa vývojový tím nejakým spôsobom nezlepší na takú úroveň, že by sa tento počet faktorov zmenšil. Teda vývoj tohto projektu by sa mal pozastaviť.

Došlo sa však aj na iný spôsob, ako priradiť nejaký časový rozmer pre vypočítané body. Najlepším spôsobom sa javí spočítať si vlastnú hodnotu pre náš tím s historických dáta a štatistík. Stačí vedieť počet bodov s nejakých predošlých projektov a čas, ktorý reálne spotrebovali a tieto hodnoty podeliť a máme hľadaný čas, ktorý presne sedí na náš tím a schopnosti. Lepšie závery sa nachádzajú v praktickej časti, v ktorej si nejaké príklady odhadov s UCP ukážeme.

II. PRAKTICKÁ ČÁST

4 VÝPOČTY METÓDOU USE CASE POINTS

V tejto kapitole si na príkladoch rôznorodých projektov ukážeme praktický dopad odhadov pomocou metódy UCP. Budú prezentované a podrobne rozobrané 2 projekty, jeden komplexnejší druhú výrazne jednoduchší, na ktorých budem prezentovať vlastnosti tejto metódy.

4.1 Projekt 1

Na začiatku uvediem len pre prehľadnosť a aby si doteraz nie príliš znalý čitateľ uvedomil celý obrázok o tom ako a s čoho tieto odhady plynú príklad reálneho projektu. Na tomto projekte prezentujem, príklad zadania a následné vytvorenie modelu prípadov použitia s diagramom použitia a všetkými aktérmi a prípadmi použitia až po samotný výpočet odhadu metódou UCP. Tento projekt je príklad reálneho zadania, ktoré som prebral zo zadania školského projektu na VUT FIT v Brne. Všetky fakty nachádzajúce sa v tejto časti podkapitoly 4.1 vychádzajú z [10].

4.1.1 Zadanie projektu IS pre hotel

Hotel Slovan potrebuje informačný systém, ktorý by dopomohol k zvýšeniu kvality poskytovaných služieb a k zefektívneniu práce zamestnancov. Hotel má predstavu o informačnom systéme, ktorý by poskytoval zamestnancov informácie o aktuálnych kapacitách hotela s možnosťou rezervácie izieb. Informačný systém by ďalej mal povereným osobám (riaditeľ, recepčný) poskytovať údaje o izbách, aktuálnych pobytoch, hosťoch, rezerváciách, poskytovaných službách a príslušných platbách. Vedúcim osobám (riaditeľ) by mal ďalej poskytovať informácie o zamestnancoch, ich pracovných aktivitách a vykonaných službách a tým dosiahnuť ich optimálne pracovné zaťaženie. Dôležitou súčasťou IS by malo byť rozhranie pre recepčného. Malo by mu umožňovať vytvárať rezervácie, ďalej musí disponovať právom vytvárať a ukončovať pobyty v hoteli. Na základe požiadaviek ubytovaných, rovnomerne rozdeľuje služby personálu. Personál bude mať taktiež prístup k IS prostredníctvom osobných terminálov (napr. pager), ktorý má tvoriť spätnú väzbu medzi zamestnancom a recepčným. Ubytovaným hosťom sa ponúka možnosť objednávania služieb priamo u recepčného. Rozhranie riaditeľa by malo umožňovať celkovú kontrolu a riadenie prevádzky hotela zahrňujúc možnosť zmeny personálu, poskytovaných služieb i ubytovacích kapacít a

cien. Informačný systém bude riešený ako aplikácia, ktorá bude komunikovať s centrálnou databázou. Systém by mal disponovať prehľadným užívateľským rozhraním.

4.1.2 Analýza aktérov

S vyššie definovaného zadania vyplývajú nasledovné charakteristiky aktérov. Tieto sú dôležitou časťou pri samotných výpočtoch odhadov na náklady softvéru.

Abstraktný aktéri

- **Zamestnanec:** predstavuje všeobecného aktéra systému, združuje všetkých ostatných aktérov do hierarchie, ako je možné vidieť v diagrame prípadov použitia namodelovaného pomocou UML.

Aktéri

- **Recepčný:** jeho úlohou je správa pobytov, ubytovaných, rezervácií, platieb
- **Personál:** prijíma len požiadavky na služby, ktoré po jej vykonaní potvrdí

Špeciálni aktéri

- **Správca:** predstavuje aktéra sa najvyššími právomocami, pripadajú mu funkcie ako recepčnému navyše disponuje právom na správu zamestnancov, izieb a dostupných služieb.
- **Systém platieb:** jedná sa o program, ktorý na základe vytvorenia pobytu a služieb s ním spojených automaticky vytvára záznam o platbe.

4.1.3 Diagram prípadov použitia

K takto špecifikovanému projektu, treba vytvoriť na začiatok diagram prípadov použitia, ten je prezentovaný na obrázku 4.1.2.1.

Obrázok 4.1.2.1: Diagram prípadov použitia pre projekt hotel Slovan vychádza z [10]

Už na prvý pohľad je vidieť, že sa jedná o rozsiahlejší projekt pre menšiu softvérovú spoločnosť zamestnávajúcich pár vývojárov. Podobné prípady sú klasické zadania pre malý tím vývojárov. Keďže doposiaľ som nemal príležitosť ani skúsenosť pracovať na rozsiahlejších projektoch obsahujúce stovky tried ako sú rozhodne rôzne projekty veľkých firiem ako IBM, Microsoft, Google a mnohé ďalšie, ktoré (napríklad IBM) používajú odhad softvérových projektov. Pre takéto firmy je totižto odhad nákladov kľúčový.

4.1.4 Analýza prípadov použitia

Hneď na úvod spomeniem, že všetky prezentované prípady použitia, ako bude zrejmé, sú popísané na rovnakej úrovni detailov. Prípady použitia podľa obrázka 4.1.2.1 sú pre prehľadnosť umiestené v prílohe A. Pre odhady nákladov si vystačíme s takýmto popisom aj keď v prílohe nie sú kompletne všetky UC z diagramu. Jedná sa o UC spojené so správou zamestnanou, služby, rezervácie, hostí. To však nie je problém, lebo všetky spomínané UC vychádzajú a sú čo do počtu scenárov totožné s UC spravuj izby, ktorý nájdeme v prílohe. Dôležité je však spomenúť, že kvôli rozsahu sú niektoré prípady použitia spojené do jedného komplexnejšieho. Ako však uvidíme pri výpočtoch tento spôsob popisu nie je stále na toľko komplexný, že by výrazne väčší počet prípadov bol zaradený do triedy komplexných.

4.1.5 Odhad nákladov metódou UCP

Nasleduje samotný postup výpočtu, ktorý vychádza s teoretickej časti. Najprv ohodnotíme a vypočítame faktory ovplyvňujúce projekt, neupravené váhy prípadov použitia a aktérov. Na záver finálny výpočet bodov prípadov použitia.

4.1.5.1 Výpočet technického faktoru

Podľa tabuľky dosadíme hodnoty pre projekt Hotel. Ohodnotená tabuľka vyzerá nasledovne.

Faktor	Názov	Multiplikátor	Ohodnotenie
T1	Distribuovaný systém	2	0
T2	Časová odozva systému	1	1
T3	Efektivita pre užívateľa	1	4
T4	Komplexné spracovanie aplikácie	1	1
T5	Znovupoužiteľnosť zdrojového kódu	1	1

T6	Náročnosť inštalácie	0,5	1
T7	Použiteľnosť	0,5	2
T8	Multiplatformnosť	2	2
T9	Možnosť zmeny	1	1
T10	Veľká súbežnosť	1	0
T11	Zabezpečenie špeciálnej bezpečnosti	1	0
T12	Závislosť od kódu tretích strán	1	2
T13	Školenie užívateľov	1	0
TFaktor			**14,75**

Tabuľka 4.1.4.1.1: Výpočet TFaktoru

$TCF = 0.6 + (0.01 * TFaktor)$

$TCF = 0.6 + (0.01 * 14{,}75)$

$\boldsymbol{TCF = 0,755}$

4.1.5.2 Výpočet environmentálneho faktoru

Podobný postup len budeme počítať environmentálny faktor EF.

Faktor	Názov	Multiplikátor	Ohodnotenie
E1	Znalosť projektu	1,5	5
E2	Skúsenosti s aplikáciou	0,5	0

E3	Skúsenosť s OO programovaním	1	5
E4	Schopnosti analyzovať	0,5	5
E5	Motivácia	1	5
E6	Stabilita požiadavkou	2	0
E7	Práca na polovičný úväzok	-1	0
E8	Náročnosť programovacieho Jazyka	-1	0
EFaktor			**20**

Tabuľka 4.1.4.2.1: Výpočet EFaktoru

$EF = 1{,}4 + (-0{,}03 * EFaktor)$

$EF = 1{,}4 + (-0{,}03 * 20)$

$\boldsymbol{EF = 0{,}8}$

4.1.5.3 Výpočet neupravených váh prípadov použitia

Znova sa vychádza z tabuľky a vzorca pre výpočet UUCW prezentovanej v teoretickej časti.

Trieda	Ohodnotenie	Počet
jednoduchý	5	6
priemerný	10	6
komplexný	15	1
UUCW		**105**

Tabuľka 4.1.4.3.1: Výpočet UUCW

4.1.5.4 Výpočet neupravených váh aktérov

Trieda	Ohodnotenie	Počet
jednoduchý	1	1
priemerný	2	0
komplexný	3	4
UAW		**13**

4.1.5.5 Finálny výpočet odhadu nákladov na projekt

Neupravené body prípadov použitia

$UUCP = UUCW + UAW$

$UUCP = 105 + 13$

$\boldsymbol{UUCP = 118}$

(upravené) Body prípadov použitia

$UCP = UUCP * TCF * EF$

$UCP = 118 * 0{,}755 * 0{,}8$

$\boldsymbol{UCP = 71{,}3\ bodov}$

V tomto momente máme spočítanú akúsi veľkosť projektu v jednotkách UCP. Nastáva moment, kedy túto hodnotu transformuje na časový údaj. Keďže tento vývojový tím nemá zatiaľ žiadne predošlé odhady, s ktorých by sa dala vypočítať ich vlastná doba na vytvorenie jedného bodu. Použijeme dva spôsoby. Jedným z nich bude podľa Karnera určená hodnota 20 hodín. Druhým bude podľa Schneidera a Wintersa [11] možnosť na základe environmentálnych faktorov určiť, či pre jeden bod treba tímu 20 alebo 28 hodín. Výslednú hodnotu určujúcu odhadovaný čas potrebný na vývoj označíme ako HE (*Hours of Effort*).

Karner

V tomto prípade použijeme hodnotu 20 hodín na jeden bod prípadov použitia.

$HE = UCP * 20$

$HE = 71{,}3 * 20$

$\boldsymbol{HE = 1425\ hodín}$

Schneider, Winters

Ako bolo rozobraté v teoretickej časti. Spočítame koľko z prvých šesť environmentálnych faktorov, teda E1 až E6 vrátane, má menšiu hodnotu ako 3 a koľko z faktorov E7 a E8 má väčšiu hodnotu ako 3. Ak tento počet bude menší ako 3, použijeme hodnotu 20 hodín, ak tento počet bude 3 alebo 4 použijeme hodnotu 28 hodín. V prípade, že tento počet bude väčší ako 4, projekt by mal byť pozastavený na základe UCP odhadov, nakoľko pre vývoj tohto projektu sú faktory príliš nepriaznivé.

Ako môžeme vidieť z tabuľky 4.1.4.2.1, faktory E1 až E6 obsahujú dve ohodnotenia menšie ako 3, konkrétne faktor E2 a E6. Faktory E7 a E8 nie sú ani jeden ohodnotený hodnotou vyššou ako 3. S toho vyplýva, že celková hodnota je dva, použijeme pre výpočet jedného bodu prípadu použitia 20 hodín ako v predošlom príklade. Výsledok sa teda nijako nezmení

$HE = UCP * 20$

$HE = 71{,}3 * 20$

$\boldsymbol{HE = 1425\ hodín}$

4.1.6 Zhodnotenie výsledkov odhadu

Keďže tento projekt je skutočným projektom, ktorý bol vypracovaný štvorčlenným tímom vývojárov ako školský projekt, po konzultácii s ich členom som previedol odhad vychádzajúci s UC modelu. Výsledný odhad príliš nesúhlasí so skutočnou dobou vývoja aplikácie. Ak počítame, že 4 členný tím pracoval na projekte priemerne 4 hodiny denne, čo vyplynulo s konzultácie s ich členom, nevychádza nám doba, ktorú sme odhadli v tomto projekte.

Odhad:

$$1425\ hodín/16\ hodín\ denne \sim \boldsymbol{90 dní}$$

Skutočnosť:

$$\boldsymbol{\sim 65\ dní}$$

Pri rozsahovo takto malom projekte sa ťažko predpovedá alebo hodnotí presnosť odhadu, nakoľko pri malom počte dní robí aj menšia nepresnosť veľký rozdiel a problém. Druhá

strana a často veľká nevýhoda a s toho plynúca nepresnosť odhadu, je zvolenie správnej hodnoty doby pre tvorbu jedného bodu prípadov použitia. Vo svojej práci Kirsten Ribu [9] uvádza, že táto doba sa môže pohybovať v intervale od 15 do 30 hodín. Ak by sme pre tento prípad použili hodnotu 15 hodín výsledok odhadu by takmer odpovedal skutočnej dobe vývoja, ako je znázornené v nasledujúcom výpočte.

$$HE = UCP * 15$$

$$HE = 71{,}3 * 15$$

$$\boldsymbol{HE = 1069\ hodín}$$

Potom časový odhad by vyšiel nasledovne:

$$1069\ hodín / 16\ hodín\ denne \sim \mathbf{67}\boldsymbol{dní}$$

Zostáva preto zatiaľ otázka, akú hodnotu zvoliť pre časový výpočet jedného bodu prípadov použitia v dobe, keď nemáme žiadne dáta s minulých projektov. Preto je určite vhodné, ak plánovanie a odhady berieme vážne uchovávať si pre budúce odhady napríklad údaje, koľko času tímu zabralo vývoj jedného bodu prípadov použitia.

4.2 Projekt 2

Nasledovný projekt bude prezentovať odhad nákladov pre vytvorenie modulu už pre zabehnutý systém. Bude prezentovaný diagram prípadov použitia, jednotlivé prípady použitia sa nachádzajú v prílohe A na konci práce. Projekt je fiktívnym projektom, ktorý som vytvoril pre účely tejto práce.

4.2.1 Zadanie projektu modul do IS banky

Banka potrebuje modul do svojho IS, ktorý bude doplňovať jeho funkcionalitu o operácie s účtami klientova banky. Modul bude mať za úlohu zaznamenávať klientov, ich účty a operácie nad nimi akými sú: vklad, výber v hotovosti, bezhotovostný prevod z účtu na účet. Prístup k IS má len pracovník banky, ktorý vykonáva všetky operácie. Systém musí byť schopný pravidelne, mesačne generovať výpisy z účtu. Modul by mal zachovať jednoduchosť ovládania aby pracovník mohol požiadavky prevádzať čo najrýchlejšie. Nesmie sa však narušiť spoľahlivosť systému. Všetky stanice, ktoré budú modul používať pracujú s operačným systémom MS Windows.

4.2.2 Analýza aktérov

Zo zadania vyplývajú nasledovné charakteristiky aktérov. Tieto sú dôležitou časťou pri samotných výpočtoch odhadov na náklady softvéru.

Aktéri

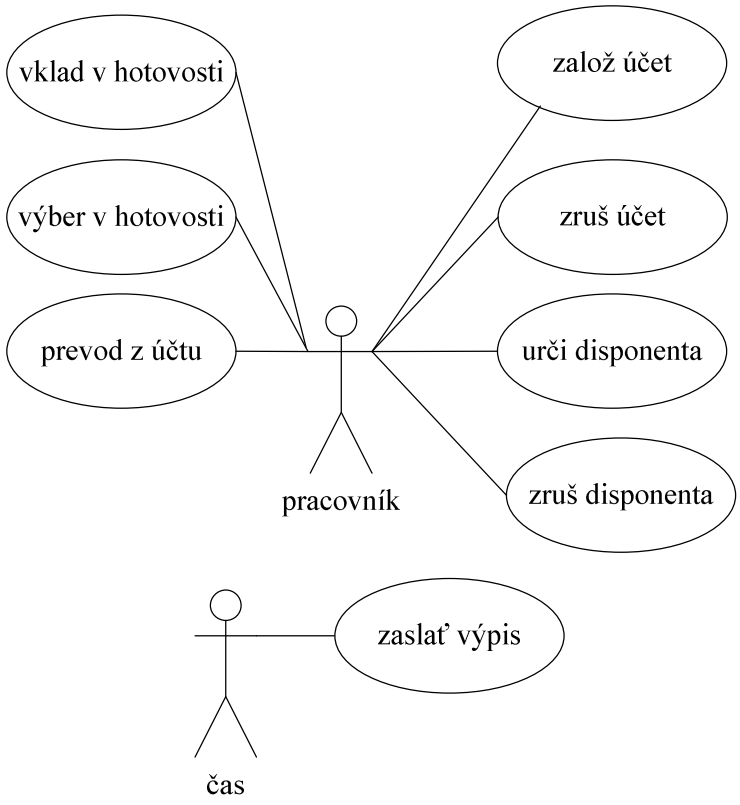
- **Pracovník:** jeho úlohou je plnenie všetkých operácií s účtami klientov podľa ich požiadavkou

Špeciálni aktéri

- **Čas:** jedná sa o program, ktorý mesačne vytvorí pre každého klienta banky výpis z účtu.

4.2.3 Diagram prípadov použitia

K vyššie špecifikovanému zadaniu prináleží nasledujúci diagram prípadov použitia



Obrázok 4.2.3.1: diagram prípadov použitia pre modul do IS banky

4.2.4 Analýza prípadov použitia

S požiadaviek zadávateľa a diagramu prípadov použitia vyplávajú scenáre a prípady použitia, ktoré sú umiestnené v prílohe A.

4.2.5 Odhad nákladov metódou UCP

Postupujeme podobne ako v predchádzajúcom príklade.

4.2.5.1 Výpočet technického faktoru

Faktor	Názov	Multiplikátor	Ohodnotenie
T1	Distribuovaný systém	2	0
T2	Časová odozva systému	1	4
T3	Efektivita pre užívateľa	1	4
T4	Komplexné spracovanie aplikácie	1	1
T5	Znovupoužiteľnosť zdrojového kódu	1	1
T6	Náročnosť inštalácie	0,5	1
T7	Použiteľnosť	0,5	2
T8	Multiplatformnosť	2	0
T9	Možnosť zmeny	1	4
T10	Veľká súbežnosť	1	0
T11	Zabezpečenie špeciálnej bezpečnosti	1	0

T12	Závislosť od kódu tretích strán	1	2
T13	Školenie užívateľov	1	0
TFaktor			**17,5**

$TCF = 0.6 + (0.01 * TFaktor)$

$TCF = 0.6 + (0.01 * 17{,}5)$

$\boldsymbol{TCF = 0,775}$

4.2.5.2 Výpočet environmentálneho faktoru

Podobný postup len budeme počítať environmentálny faktor EF.

Faktor	Názov	Multiplikátor	Ohodnotenie
E1	Znalosť projektu	1,5	5
E2	Skúsenosti s aplikáciou	0,5	3
E3	Skúsenosť s OO programovaním	1	5
E4	Schopnosti analyzovať	0,5	5
E5	Motivácia	1	5
E6	Stabilita požiadavkou	2	0
E7	Práca na polovičný úväzok	-1	0
E8	Náročnosť programovacieho Jazyka	-1	0

EFaktor	**21,5**

Tabuľka 4.1.4.2.1: Výpočet EFaktoru

$EF = 1{,}4 + (-0{,}03 * EFaktor)$

$EF = 1{,}4 + (-0{,}03 * 21{,}5)$

$\boldsymbol{EF = 0{,}755}$

4.2.5.3 Výpočet neupravených váh prípadov použitia

Znova sa vychádza z tabuľky a vzorca pre výpočet UUCW prezentovanej v teoretickej časti.

Trieda	**Ohodnotenie**	**Počet**
jednoduchý	5	8
priemerný	10	0
komplexný	15	0
UUCW		**40**

Tabuľka 4.2.5.3.1: Výpočet UUCW

4.2.5.4 Výpočet neupravených váh aktérov

Trieda	**Ohodnotenie**	**Počet**
jednoduchý	1	1
priemerný	2	0
komplexný	3	1
UAW		**4**

4.2.5.5 Finálny výpočet odhadu nákladov na projekt

Neupravené body prípadov použitia

$$UUCP = UUCW + UAW$$
$$UUCP = 40 + 4$$
$$\boldsymbol{UUCP = 44}$$

(upravené) Body prípadov použitia

$$UCP = UUCP * TCF * EF$$
$$UCP = 44 * 0{,}775 * 0{,}755$$
$$\boldsymbol{UCP = 25{,}7\ bodov}$$

Z dosiahnutého výpočtu môžeme ďalej ako v prvom projekte, určiť skutočné náklady na projekt formou hodín. Budem postupovať rovnako, aby sme mohli určiť rozdiely týchto projektov. Spočítame teda čas potrebný na vývoj označený HE.

Karner

V tomto výpočte použijeme hodnotu 20 hodín na jeden bod prípadov použitia.

$$HE = UCP * 20$$
$$HE = \mathbf{25{,}7} * 20$$
$$\boldsymbol{HE = 514\ hodín}$$

Schneider, Winters

Podobne postupujeme aj pri určení jednej z dvoch možností, ako prezentujú Schneider a Winters. Dostávame hodnotu 20 hodín, keďže medzi environmentálnymi faktormi E1 až E6, vrátane nich je jeden faktor, E6 ohodnotených menšiu hodnotou ako 3 a z faktorov E7 a E8 nemá ani jeden väčšiu hodnotu ako 3. To znamená použiť 20 hodín na jeden prípad použitia podobne ako v prvom príklade. Znova dostávame rovnaký výsledok, ako ten podľa Karnera, keďže použijeme tú istú hodnotu pre vývoj jedného prípadu použitia teda:

$$HE = UCP * 20$$
$$HE = \mathbf{25{,}7} * 20$$
$$\boldsymbol{HE = 514\ hodín}$$

4.2.6 Zhodnotenie výsledkov odhadu

Keďže tento projekt je len fiktívnym projektom, výsledok odhadu sa nedá porovnať so skutočnosťou. Poskytuje však možnosť poukázania rozdielu odhadu pri dvoch rôznych typoch projektov. Budeme predpokladať, že na projekte pracuje dvojčlenný tím 4 hodiny denne.

Odhad:

$$514\ hodín/8\ hodín\ denne \sim \mathbf{64}\ \boldsymbol{dn}\text{í}$$

V prípade že by sme na projekt pridelili rovnaký počet členov tímu dostávame hodnotu odhadu:

$$514\ hodín/16\ hodín\ denne \sim \mathbf{32}\ \boldsymbol{dn}\text{í}$$

Môžeme si všimnúť, ako výrazne ovplyvňuje klasifikácia prípadov použitia do skupín podľa komplexnosti finálny odhad. Keďže oba príklady majú skoro rovnaký počet prípadov použitia, líšia sa teda o jeden, pritom druhý projekt je zaťažený viac technickým ale menej environmentálnym faktorom. Tento výsledok je pritom výrazne nižší ako v prvom projekte, kde bol odhad približne 90 dní s rovnakým počtom hodín pre vývoj jedného bodu prípadov použitia. Tento fakt vlastne poukazuje na istú medzeru v UCP, ktorou nepochybne je klasifikácia prípadov použitia do skupín podľa komplexnosti. Ako bolo spomenuté v prvom projekte, prípady použitia boli písané viac komplexnejším spôsobom, mali by teda obsahovať viacej transakcií. No nebolo to veľmi viditeľné. Alebo bola možnosť rozpísať tieto scenáre do viacero prípadov použitia, čím by sa zväčšil ich počet, čo by malo zasa veľký vplyv na celkový dopad odhadu. Pritom ani v jednom prípade by nebolo porušené pravidlo písania UC na rovnakej úrovni detailu popisu. Zostáva teda otázka, vytvárať viacero prípadov použitia s menej scenármi alebo vytvoriť jeden komplexný UC?

5 ZHODNOTENIE METÓDY

V tejto kapitole si dovolím zhrnúť celkový dopad používania metódy UCP pre odhad nákladov pre vývoj softvérových projektov. Tak ako to už býva, všetko má svoje výhody a nevýhody, metóda UCP nie je výnimkou.

Výraznou výhodou, ako je zrejmé s praktickej časti je jednoduchosť výpočtov a použite tejto metódy. Odhad je založený na primitívnych matematických operáciách, preto môže byť proces veľmi ľahko automatizovaný a tým pádom sa ušetrí čas pri zisťovaní odhadov. Proces samotný môže byť aj výrazne presnejší, ak sa bude uchovávať akýsi priemer určujúci, koľko treba danej organizácii na vypracovanie jedného bodu prípadov použitia. Tento fakt nijako vážne organizáciu nezaťaží, ale poskytne jej možnosť pomocou UCP určiť presnejšie odhady, čo vidím ako veľkú výhodu. Určite ako bolo možné a si všimnúť UCP nám poskytuje možnosť použiť akejsi jednotky, ktorá skutočne meria veľkosť projektu. Tento odhad nám umožňuje oddeliť veľkosť projektu od doby trvania, ktorá sa dá zistiť už spomínaných štatistických údajov s predchádzajúcich projektov. Avšak hlavnou výhodou je, že táto metóda sa dá použiť v ranných fázach životného cyklu. Nepotrebujeme vedieť žiadne dátové závislosti, štruktúry ako napríklad u FPA. Tá sa síce podobá na UCP, ale asi len toľko, že svoje odhady zakladá z užívateľského (ako UCP) pohľadu, je nezávislá od použitia programovacieho jazyka, použitej metodológie, a možností samotného vývojového tímu. Stačí vytvorenie UC modelu a odhady sa môžu začať. Samozrejme otázkou je aký UC model vytvoríme, či je dostatočne vierohodný a podobne, viac o nevýhodách a problémoch vzápätí.

Metóda má však aj radu nevýhod. Ako bolo spomínané viackrát v texte, už samotná tvorba prípadov použitia je veľkou záťažou pri vývoji softvéru. Síce UC sú produktom analýzy a sú vhodné pre ucelene myšlienky o projekte, no nie je však jednoduché udržovať popis UC stále na rovnakej úrovni detailu, hlavne ak prípady použitia píše v priebehu času viacero členov tímu. Ak by sa nám však tento faktor podarilo predsa eliminovať nastáva moment, kedy treba vyjadriť čas pre vývoj jedeného bodov prípadov použitia v prípade, že máme vierohodnú veľkosť určenú UCP. Ako bolo prezentované v príkladoch, existuje niekoľko prístupov, no stále sú medzi nimi značné rozdiely a nie je exaktne určené, ktorý dodá najvierohodnejší odhady. Čiastočne sa tento problém rieši použitím vlastných štatistík, no najprv by sme museli nejaké mať, čo pri začiatku nemáme, vybudujeme ich až časom. Ďalej, ako mnohokrát bolo spomínané, závislosť odhadov na UC modeli znemožní

odhad ak nemáme vytvorený kompletný UC model projektu, teda všetky prípady použitia musia byť nadefinované na rovnakej úrovni detailu popisu ich scenárov. S toho plynie ďalší problém, pri zisťovaní váh prípadov použitia vychádzame s delenia ich komplexnosti na základe počtu transakcií. Transakciu sa teda rozumie výmena medzi aktérom a systémom. Aktér niečo požaduje systém mu odpovedá. Je teda veľmi dôležité dodržať buď spôsob písania UC. V prípade keď jeden krok v UC nezodpovedá jednej transakcii, čo je veľmi časté prihliadnuc na prezentované projekty, je dôležité správne pochopenie scenára a rozdelenie ho do transakcií. Samotnou otázkou aj je či UCP ako jednota nie sú príliš všeobecná a tým pádom vznikajú nepresnosti v časových odhadoch. Tento fakt je riešený v knihe [12], rozobratím UC na menšie časti, teda scenáre a ponúkať ich ako jednotku určujúcu veľkosť, s tým však súvisí kompletná zmena štruktúry samotného výpočtu.

Prezentoval som isto len pár výhod a nevýhod metódy UCP, dôslednejšou analýzou by sme určite dospeli k viacerým. Považujem ich však za kľúčové a veľmi dôležité. Po zhodnotení jednotlivých výhod a nevýhod sa ľahko dopracujeme k možným vylepšeniam, tie však vedú v istých prípadoch k zásadným zmenám modelu a možno k vytvoreniu inej metódy, ktorá by sa snažila tieto nedostatky riešiť. Preto by v tomto smere nemal prestať výskum v odhadoch nákladov na softvérové projekty.

ZÁVER

Na záver už len veľmi krátko zhodnotím celkový dopad práce, moje vlastné názory. Táto téma je podľa mňa veľmi aktuálna a potrebuje neustále zlepšovanie. V práci som sa snažil poukázať na výhody, nevýhody a poukázať na chyby alebo problémy, s ktorými sa pri odhadoch nákladov na projekt stretávame. Praktická časť bola výhradne zameraná na odhady metódou UCP. Porovnanie a komplexné riešenie problémov dnes používaných metód by znamenalo omnoho rozsiahlejšie rozpracovanie tejto témy, čo môže byť možným rozšírením tejto práce. Celkové spracovanie hodnotím ako úspešné, podarilo sa mi rozpracovať a na príkladoch dvoch rôznych projektov vysvetliť princíp odhadov nákladov softvérových projektov pomocou UCP. Nepodarilo sa mi však nájsť iné metódy, ktoré by boli podobne ako UCP založené na scenároch a využívali by OO analýzu a návrh ako stavebný kameň pre svoje výpočty. V tomto ohľade však existuje viacero článkov a zdrojov, ktoré sa týmto problémom zaoberajú, čo hodnotím ako prínos aj v rámci mojej práce.

ZÁVER V ANGLIČTINE

At the end, very brief evaluation of the full impact of work with my own opinions. In my view, this subject is very present and needs constant improvement. At this thesis I have tried to highlight the advantages, disadvantages and point out errors or problems with which we have to deal in project cost estimations. The practical part was exclusively focused on estimates using UCP method. Comparison and complex problem-solving solution of methods used today, would mean a much more extensive elaboration of this topic, which may be the possible extension of this thesis. I evaluate this processing as successful, I managed to elaborate and on two examples of different projects explain the principle of cost estimates of software projects using UCP. I could not find other methods that would be like based on scenarios and use the OO analysis and design as a building block for its calculations like UCP. However, in this consideration there are numerous articles and resources which deal with this problem, which I evaluate as a contribution to my thesis.

ZOZNAM POUŽITEJ LITERATÚRY

[1] BOEHM, B. W. *Software Engineering Economics*. USA, Upper Saddle River, Prentice-Hall 1981.

[2] SEDLÁČKOVÁ, J. *Cenové odhady softwarových projektů*. Masarykova univerzita v Brně, Fakulta informatiky, Brno, 2005. Diplomová práce.

[3] KŘENA, B.; KOČÍ, R. *Úvod do softwarového inženýrství*. Vysoké učení technické v Brně, Fakluta informačních technologií, Brno, 2006. Studijní opora.

[4] STELLMAN, Andrew ; GREENE, Jennifer. *Building better software* [online]. May 3, 2009 [cit. 2011-06-03]. Dostupné z WWW: <http://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/>

[5] COCKBURN, Alistar. *Writing Effective Use Cases*. 1st edition.: Addison-Wesley Professional, October 15, 2000. 304 pp. ISBN 0201702258.

[6] KARNER, G. *Metrics for Objectory*. University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21, December 1993. Diploma thesis.

[7] ALEXANDER, Alvin. *How to Determine Your Software Application Size Using Function Point Analysis.* [online]. 1998 [cit. 2011-06-03]. Dostupné z WWW: <http://www.devdaily.com/FunctionPoints/>.

[8] SEHLHORST, Scott . *Software Cost Estimation With Use Case Points* [online]. 2007 [cit. 2011-06-03]. Dostupné z WWW: http://tynerblain.com/blog/2007/02/12/software-cost-estimation-ucp-1/>.

[9] RIBU, Kirsten. *Estimating Object-Oriented Software Projects with Use Cases*. University of Oslo, Department of Informatics. 2001. Master of Science Thesis

[10] TÍNES, Lukáš ; MARTINÁK, Lukáš ; BARTOŠ, Peter ; HORNICKÝ Pavol. *Analýza a návrh informačních systémů: Hotel.* Vysoké učení technické v Brně, Fakluta informačních technologií, Brno, 2010. Školský projekt.

[11] SCHNEIDER, Geri ; WINTERS, Jason P. . *Applying Use Cases: A Practical Guide*. : Addison-Wesley Professional, September 15, 1998. 188 pp. ISBN 0201309815.

[12] COHN, Mike . *Agile Estimating and Planning*.: Prentice Hall, November 11, 2005. 368 pp. ISBN 9780131479418.

PRÍLOHA A

Prípady použitia k projektu 1

Tabuľky jednotlivých prípadov použitia k projektu 1, Hotel Slovan. Prípady použitia sú vytvorené podľa [10].

Identifikátor	UC01		
Názov	Prípad použitia: Spravuj izby		
Popis	Umožní správcovi pridať, upraviť alebo zrušiť izbu		
Priorita	2 – stredná	**Frekvencia**	občas
Vstupné	Správca je prihlásený do systému		
Výstupné	Izba je vytvorená, upravená alebo zrušená		
Užívatelia	správca		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém zobrazí zoznam izieb	
	2	Systém ponúkne možnosť pridať, upraviť alebo zrušiť izbu	
	3	Ak správca vyberie pridať novú izbu	
	3.1	Systém zobrazí okno s možnosťou zadať informácie	
	3.2	Správca potvrdí pridanie novej izby	
	3.3	Systém pridá izbu a zobrazí ju v zozname izieb	
	4	Ak správca vyberie možnosť upraviť	
	4.1	Systém zobrazí informácie o danej izbe, ktoré je možno zmeniť	
	4.2	Správca potvrdí zmeny	
	4.3	Systém zmeny uloží a zobrazí ich v zozname izieb	
	5	Ak správca vyberie možnosť zrušiť izbu	
	5.1	Systém overí, či k tejto izbe existuje rezervácia	
	5.2	Ak je izba rezervovaná	
	5.2.1	Systém zobrazí varovanie a neumožní izbu zrušiť	
	5.3	Systém zobrazí okno so zrušením izby	
	5.4	Správca potvrdí zrušenie	
	5.5	Systém vymaže informácie o izbe	

Alternatívny scenár	**Krok**	**Činnosť**
	-	-
Výnimky	-	

Tabuľka A.1: Prípad použitia UC01: Spravuj izby

Identifikátor	UC02		
Názov	Prípad použitia: Vytvor pobyt		
Popis	Umožní recepčnému vytvoriť pobyt, ak sa jedná o pobyt z rezervácie, tak zároveň vymaže rezerváciu		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Recepčný je prihlásený v systéme		
Výstupné	Pobyt je vytvorený s priradeným hosťom a rovno vytvorená platba za pobyt		
Užívatelia	recepčný		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém zobrazí okno s otázkou o aký typ sa jedná (z rezervácie / priamo)	
	2	Recepčný vyberie možnosť vytvorenia z rezervácie [alt. scenár A]	
	3	Systém zobrazí zoznam rezervácií	
	4	Recepčný vyberie rezerváciu	
	5	Systém zobrazí detaily o pobyte	
	6	Recepčný zvolí „Pridať hosťa“	
	7	Systém zobrazí okno s možnosťou zadať informácie o hosťovi	
	8	Recepčný vyplní informácie	
	9	Systém potvrdí pobyt a platbu za pobyt	
Alternatívny scenár	**Krok**	**Činnosť**	
	A2	Systém zobrazí voľné izby	
	A3	Recepčný zadá „Pridať hosťa“	
	A4	Systém zobrazí okno pre vyplnenie informácií o hosťovi a pobyte	
	A5	Recepčný zadá informácie o hosťovi a pobyte	

	A6	Systém vytvorí pobyt a platbu
Výnimky	UC02E1: Žiadna voľná izba	

Tabuľka A.2: Prípad použitia UC02: Vytvor pobyt

Identifikátor	UC02E1		
Názov	Prípad použitia: Vytvor pobyt bez rezervácie žiadna voľná izba		
Popis	Systém informuje recepčného, že nie je žiadna voľná izba		
Priorita	2 – stredná	**Frekvencia**	občas
Vstupné	Recepčný je prihlásený v systéme		
Výstupné	Je zobrazené informácia o nemožnosti rezervovať pobyt		
Užívatelia	recepčný		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém informuje recepčného o nedostupnosti voľnej izby a je znemožnené vytváranie pobytu	
	2	Recepčný potvrdí prijatie informácie	
	3	Systém zruší vytvorenie pobytu	
Alternatívny scenár	**Krok**	**Činnosť**	
	-	-	
Výnimky	-		

Tabuľka A.3: Prípad použitia UC02E1: Vytvor pobyt bez rezervácie žiadna voľná izba

Identifikátor	UC03		
Názov	Prípad použitia: Zobraz pobyty		
Popis	Umožní recepčnému zobraziť aktuálne pobyty		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Recepčný je prihlásený v systéme		
Výstupné	Výpis aktuálnych pobytov		
Užívatelia	recepčný		

Hlavný scenár	**Krok**	**Činnosť**
	1	Systém zobrazí zoznam pobytov s možnosťou filtrovania
	2	Ak recepčný zvolí konkrétny pobyt
	2.1	Systém zobrazí podrobné okno s informáciami o pobyte
	3	Recepčný ukončí prehliadanie pobytov
Alternatívny scenár	**Krok**	**Činnosť**
	-	-
Výnimky	-	

Tabuľka A.4: Prípad použitia UC03: Zobraz pobyty

Identifikátor	UC04		
Názov	Prípad použitia: Ukonči pobyt		
Popis	Umožní recepčnému ukončiť pobyt		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Recepčný je prihlásený v systéme a má zoznam pobytov z UC03		
Výstupné	Pobyt je ukončený		
Užívatelia	recepčný		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém zobrazí základné informácie o pobyte a celkovú sumu	
	2	Recepčný potvrdí ukončenie pobytu	
Alternatívny scenár	**Krok**	**Činnosť**	
	-	-	
Výnimky	-		

Tabuľka A.5: Prípad použitia UC04: Ukonči pobyt

Identifikátor	UC05		
Názov	Prípad použitia: Predĺž pobyt		
Popis	Umožní recepčnému predĺžiť pobyt		
Priorita	2 – stredná	**Frekvencia**	zriedkavo
Vstupné	Recepčný je prihlásený v systéme a má zoznam pobytov z UC03		
Výstupné	Pobyt je predĺžený		
Užívatelia	recepčný		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém zobrazí okno s možnosťou zadania nového termínu ukončenia pobytu	
	2	Recepčný zadá nový termín ukončenia	
	3	Systém skontroluje, či na tento termín a izbu už nie je rezervácia [alt. scenár A]	
	4	Systém zmení termín ukončenia pobytu	
	5	Recepčný potvrdí zmeny termínu	
Alternatívny scenár	**Krok**	**Činnosť**	
	A3	Systém vypíše informáciu o nemožnosti predĺženia pobytu	
Výnimky	-		

Tabuľka A.6: Prípad použitia UC05: Predĺž pobyt

Identifikátor	UC06		
Názov	Prípad použitia: Prijmi platbu		
Popis	Umožní prijať platbu za vykonané služby		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Recepčný je prihlásený v systéme a má zoznam pobytov z UC03		
Výstupné	Platba je prijatá		
Užívatelia	recepčný		
Hlavný scenár	**Krok**	**Činnosť**	

	1	Systém otvorí okno s platbami
	2	Recepčný zvolí prijať platbu
	3	Systém otvorí okno s informáciami o prijatí platby, vytlačí účet za služby a označí platby za uhradené
	4	Recepčný potvrdí uhradenie platby
Alternatívny scenár	**Krok**	**Činnosť**
	-	-
Výnimky	-	

Tabuľka A.7: Prípad použitia UC06: Prijmi platbu

Identifikátor	UC07		
Názov	Prípad použitia: Prijmi požiadavku na službu		
Popis	Umožní prijať požiadavku od zákazníka		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Recepčný je prihlásený v systéme a má zoznam pobytov z UC03		
Výstupné	Požiadavka je prijatá		
Užívatelia	recepčný		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém otvorí okno so zoznamom poskytovaných služieb	
	2	Recepčný zvolí zákazníkom vybranú službu a potvrdí výber	
	3	Systém uloží službu medzi ostatné služby ktoré treba vykonať	
Alternatívny scenár	**Krok**	**Činnosť**	
	-	-	
Výnimky	-		

Tabuľka A.8: Prípad použitia UC07: Prijmi požiadavku na službu

Identifikátor	UC08		
Názov	Prípad použitia: Preber požiadavku na službu		
Popis	Umožní personálu prebrať vykonanie služby		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Personál je prihlásený v systéme		
Výstupné	Požiadavka je prebraná		
Užívatelia	personál		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém otvorí okno s požiadavkami čakajúce na vykonanie	
	2	Personál potvrdí prebranie služby	
	3	Systém označí službu ako vykonanú	
Alternatívny scenár	**Krok**	**Činnosť**	
	-	-	
Výnimky	-		

Tabuľka A.9: Prípad použitia UC08: Preber požiadavku na službu

Identifikátor	UC09		
Názov	Prípad použitia: Potvrď vykonanie služby		
Popis	Umožní personálu prebrať vykonanie služby		
Priorita	2 – stredná	**Frekvencia**	často
Vstupné	Personál je prihlásený v systéme, vykonanie služby bolo dokončené		
Výstupné	Vykonanie služby je potvrdené		
Užívatelia	personál		
Hlavný scenár	**Krok**	**Činnosť**	
	1	Systém otvorí okno s službami, ktoré sa majú vykonať	
	2	Personál vyberie „vykonané služby“	
	3	Systém zobrazí služby ktoré sú označené ako vykonané	
	4	Personál vyberie službu, ktorú už vykonal a potvrdí výber	

	5	Systém označí službu ako vykonanú a vytvorí platbu za službu
Alternatívny scenár	**Krok**	**Činnosť**
	-	-
Výnimky	-	

Tabuľka A.10: Prípad použitia UC09: Potvrď vykonanie služby

Prípady použitia k projektu 2

Tabuľky jednotlivých prípadov použitia k projektu 2, modul do IS.

ID	UC01
názov	Založ účet
popis	Klient (nový / existujúci) požaduje založenie nového účtu
aktér	pracovník
Hlavný scenár	1. Pracovník zadá meno klienta
	2. Ak systém vypíše, že klient nie je zatiaľ klientom banky
	2.1. Pracovník vytvorí nový záznam o klientovi a vyplní jeho informácie vo formuláre
	3. Ak systém nájde viacej mien, ponúkne ich zoznam
	3.1 Pracovník vyberie správneho klienta
	4. Pracovník zadá údaje o novom účte
Alternatívny scenár	-

Tabuľka A.11: Prípad použitia UC01: Založ účet

ID	UC02
názov	Zruš účet
popis	Klient požiada o zrušenie účtu
aktér	pracovník
Hlavný scenár	1. Pracovník zadá meno klienta
	2. Systém zobrazí okno s možnými operáciami
	3. Pracovník zvolí možnosť „zrušiť účet“
	4. Systém zobrazí okno s možnosťou pre poslať zostatok na iný účet
	5. Ak pracovník potvrdí prevod na iný účet
	5.1. Systém vykoná akciu podľa UC03
	6. Pracovník vráti zostatok a v systéme potvrdí vrátenie zostatku
	7. Systém zruší účet klienta
Alternatívny scenár	-

Tabuľka A.12: Prípad použitia UC02: Zruš účet

ID	UC03
názov	Prevod z účtu na účet
popis	Klient požiada o prevod čiastky na iný účet
aktér	pracovník
Hlavný scenár	1. Pracovník zadá rovno číslo účtu klienta
	2. Systém zobrazí okno s informáciami o účte klienta
	3. Pracovník zvolí možnosť „prevod z účtu“ a zadá požadovanú sumu
	4. Ak systém zistí, že zostatok na účte je menší ako požadovaná suma
	4.1. Vypíše informáciu o nemožnosti previesť sumu a ponúkne pre prevod zostatok
	5. Pracovník zadá číslo cieľového účtu
	6. Systém prevedie transakciu prevodu
Alternatívny scenár	-

Tabuľka A.13: Prípad použitia UC03: Prevod z účtu na účet

ID	UC04
názov	Vklad v hotovosti
popis	Klient požiada vloženie sumy na účet
aktér	pracovník
Hlavný scenár	1. Pracovník zadá číslo cieľového účtu
	2. Systém zobrazí okno so zadaním čiastky

	3. Pracovník prijme hotovosť a potvrdí vloženie čiastky
	4. Systém zobrazí okno so správu o vložení
Alternatívny scenár	4A. Vloženie nebolo úspešné, systém nahlási chybu a vynuluje vklad
	5A. Pracovník sumu neprevezme

Tabuľka A.14: Prípad použitia UC04: Vklad v hotovosti

ID	UC05
názov	Výber v hotovosti
popis	Vlastník alebo disponent požaduje výber v hotovosti
aktér	pracovník
Hlavný scenár	1. Pracovník zadá číslo účtu
	2. Systém overí oprávnenie k výberu
	2.1 Klient nemá oprávnenie prípad použitia končí
	3. Pracovník zadá požadovanú sumu k výberu
	4. Systém zobrazí okno so správu o vložení
Alternatívny scenár	4A. Vloženie nebolo úspešné, systém nahlási chybu a vynuluje vklad
	5A. Pracovník sumu neprevezme

Tabuľka A.15: Prípad použitia UC05: Výber v hotovosti

ID	UC06
názov	Urči disponenta
popis	Vlastník požaduje určenie disponent účtu
aktér	pracovník
Hlavný scenár	1. Pracovník zadá číslo účtu
	2. Systém zobrazí okno s možnosťou "pridaj disponenta"
	3 Pracovník zadá informácie o disponentovi
	3. Systém informácie spracuje
Alternatívny scenár	-

Tabuľka A.16: Prípad použitia UC06: Urči disponenta

ID	UC07
názov	Zruš disponenta
popis	Vlastník požaduje zrušenie disponenta účtu
aktér	pracovník
Hlavný scenár	1. Pracovník zadá číslo účtu
	2. Systém zobrazí okno s možnosťou "zruš disponenta"
	3 Pracovník potvrdí zruší disponenta
	3. Systém informácie spracuje
Alternatívny scenár	-

Tabuľka A.17: Prípad použitia UC07: Zruš disponenta

ID	UC08
názov	Zaslať výpis
popis	Systém zašle výpisy vlastníkom účtu
aktér	čas
Hlavný scenár	1. Systém vytvorí výpis a rozošle ich vlastníkom
Alternatívny scenár	-

Tabuľka A.18: Prípad použitia UC08: Zaslať výpis