

Algoritmické metody odhadování software

Filip Bednařík

Bakalářská práce
2015



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Filip Bednařík**

Osobní číslo: **A12261**

Studijní program: **B3902 Inženýrská informatika**

Studijní obor: **Informační technologie v administrativě**

Forma studia: **prezenční**

Téma práce: **Algoritmické metody odhadování software**

Téma anglicky: **Algorithmic Methods for the Estimation of Software Size**

Zásady pro vypracování:

1. Seznamte se s algoritmickými přístupy k odhadování velikosti softwaru.
2. Zaměřte se na metodiku funkčních bodů.
3. Uveďte výhody a nevýhody jednotlivých přístupů.
4. Vypracujte postup výpočtu.
5. Navrhněte způsob realizace postupu výpočtu v tabulkovém editoru nebo jiným vhodným způsobem.

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: tištěná/elektronická

Seznam odborné literatury:

1. Boehm, B. W., et al.: COCOMO II Model Definition Manual. USA, Los Angeles, University of Southern California 1999.
2. International Function Points User Group: Function Point Counting Practices Manual: Release 4.1.1999.
3. Sedláčková, J.: Cenové odhady softwarových projektů. Masarykova univerzita v Brně, Fakulta informatiky, Brno, 2005. Diplomová práce.
4. Anda, B. Comparing Use Case based Estimates with Expert Estimates. Proc. of Empirical Assessment in Software Engineering, Keele, United Kingdom, April 8-10, 2002.
5. 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>.
6. SILHAVY, Radek; SILHAVY, Petr; PROKOPOVA, Zdenka. Requirements Based Estimation Approach for System Engineering Projects. In: Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering. Springer International Publishing, 2015. p. 467-472.

Vedoucí bakalářské práce:

Ing. Radek Šilhavý, Ph.D.

Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce:

6. února 2015

Termín odevzdání bakalářské práce:

21. května 2015

Ve Zlíně dne 6. února 2015


doc. Mgr. Milan Adámek, Ph.D.
děkan




Ing. Miroslav Matýšek, Ph.D.
ředitel ústavu

Prohlašuji, že

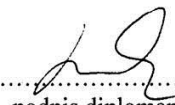
- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/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 diplomovou/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 – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s příjím-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/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 diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/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 diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraňá do IS/STAG jsou totožné.

Ve Zlíně

18.5 2015


.....
podpis diplomanta

ABSTRAKT

Ve své bakalářské práci se zaměřuji na algoritmické metody odhadování softwaru. Cílem teoretické části práce je vysvětlení důležitých pojmů, charakteristika nejznámějších metod pro odhadování, kterými jsou COCOMO II a metoda funkčních bodů (FPA). V praktické části věnuji pozornost výhodám a nevýhodám odhadovacích metod. Součástí je vzorový příklad, na kterém provedu výpočet pomocí metody FPA. Následně importuji všechna získaná data a výpočty do tabulkového prostředí aplikace Excel.

Klíčová slova: odhadování, COCOMO II, FPA, projekt, produkt, životní cyklus, doba vývoje, funkcionalita

ABSTRACT

When i was writing my final work, I was focused on Algorithmic Methods for the Estimating Software. The object of the theoretical part is explanation of all major concepts, characteristics of the best-known methods for estimating we known them as COCOMO II and function point analysis (FPA). In my Practic part i devote my attention to the advantages and disadvantages of estimation methods. Also includes example model that I can show calculation using the method FPA. Subsequently, I will import all data and calculations that i get into the Excel spreadsheet environment.

Keywords: estimation, COCOMO II, FPA, project, product, product life-cycle, development time, functionality.

Rád bych poděkoval panu Ing. Radkovi Šilhavému, Ph.D. za cenné rady, odborné vedení a vstřícnost při konzultacích. V neposlední řadě také děkuji všem, kteří mi byli oporou při vypracování bakalářské práce.

OBSAH

ÚVOD.....	8
I TEORETICKÁ ČÁST.....	9
1 PROJEKT (TERMINOLOGIE).....	10
1.1 PROJEKT.....	10
1.2 PRODUKT	10
1.3 PROJEKTOVÝ TROJÚHELNÍK (TROJIMPERATIV)	10
1.4 ŽIVOTNÍ CYKLUS PROJEKTU	11
1.5 METODIKY VÝVOJE SOFTWARE	12
1.5.1 Tradiční metodiky	12
1.5.2 Agilní metodiky	12
2 METODY ODHADOVÁNÍ.....	14
2.1 ODHADOVÁNÍ.....	14
2.1.1 Přesnost odhadu	14
3 ALGORITMICKÉ METODY	16
3.1 COCOMO 81.....	16
3.2 COCOMO II.....	16
3.2.1 Modely odhadů.....	17
3.2.2 Odhad doby vývoje	25
3.3 ODHAD FPA.....	25
3.3.1 Neupravené funkční body (UFP)	26
3.3.2 Upravené funkční body	27
3.3.3 Odhad doby realizace	29
II PRAKTICKÁ ČÁST	31
4 POROVNÁNÍ ALGORITMICKÝCH METOD	32
5 ODHAD POMOCÍ METODY FPA	34
5.1 VÝPOČET NEUPRAVENÝCH FUNKČNÍCH BODŮ	35
5.2 VÝPOČET UPRAVENÝCH FUNKČNÍCH BODŮ	42
5.3 ZJIŠTĚNÍ CELKOVÝCH NÁKLADŮ	51
ZÁVĚR	52
SEZNAM POUŽITÉ LITERATURY.....	53
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	55
SEZNAM OBRÁZKŮ	56
SEZNAM TABULEK.....	57
SEZNAM PŘÍLOH.....	59

ÚVOD

V dnešní době je odhadování softwarových produktů nezbytnou součástí každé softwarové firmy na území České republiky, ale i v zahraničí. Při procesu odhadování je analyzováno velké množství požadavků, rizik a cílů. Správná kalkulace odhadů na náklady, dobu vývoje a četnost vývojářského týmu může takové firmě ušetřit spoustu nákladů i značné množství času. Detailní spolupráce se zadavatelem projektu garantuje zvýšenou kvalitu konečného produktu. V opačném případě může dojít k degeneraci kvalitativní hodnoty, nespokojenosti zákazníka a uživatelů. Dle mého názoru, pro úspěšnost projektu a spokojenost zákazníka, je důležité umět stanovit podrobné požadavky a nepodceňovat úvodní odhad, který nám v konečné fázi může přinést ovoce. Především z tohoto důvodu, důležitosti odhadu, jsem si zvolil příslušné téma.

Podstatné vědomosti, které jsem získal z předmětu Tvorba a analýza softwaru v závěrečném semestru bakalářského studia, mi poskytly úvodní náhled do této problematiky. Podrobně se ve své práci zaměřuji na některé algoritmické metody, určené k odhadování softwarových projektů. Velkou pozornost věnuji metodám COCOMO II a funkčních bodů, které považuji za jedny z nejdůležitějších.

V praktické části porovnávám dané metody a zdůrazňuji jejich hlavní výhody a nevýhody. Další z cílů mé práce je zaměřit se na metodu funkčních bodů, kterou aplikuji na vzorovém příkladu. Příklad ilustruji na fiktivním softwarovém produktu, u kterého stanovím hlavní požadavky na funkcionalitu. Neposledním cílem je ohodnotit softwarový produkt a provést výpočet funkčních bodů znázorněný v tabulkovém editoru Excel. V konečné fázi uplatním nominální hodnotu funkčních bodů k výpočtu celkových odhadů.

I. TEORETICKÁ ČÁST

1 PROJEKT (TERMINOLOGIE)

1.1 PROJEKT

Definice projektu podle PMI *“Projekt je dočasné úsilí s cílem vytvořit unikátní produkt nebo službu.”* [1]

Projekt je řízený proces se svým začátkem a koncem a předem stanovenými pravidly. Je rovněž základním stavebním prvkem projektového řízení.

Časové vytyčení projektu se ve většině případů uvádí jako nezbytná součást projektu a to proto, aby nám jasně stanovila, co je začátkem a koncem práce. Aktivitu bez předem určeného konce a výstupu nemůžeme v žádném případě řadit mezi projekty, nýbrž procesy. Cílem projektového snažení je vytvoření unikátního produktu (předmětu, služby), který je definován určitými atributy.

1.2 Produkt

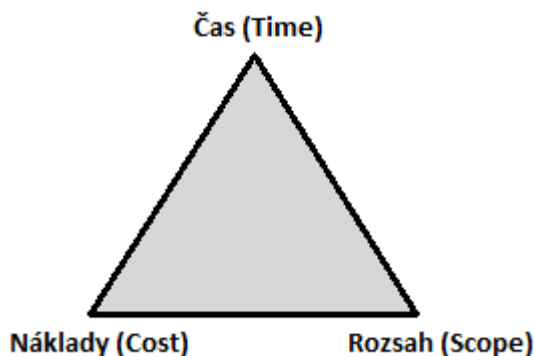
Ke správnému porozumění pojmu projekt je nezbytné si definovat co to produkt je:

Produkty jsou výsledky činnosti nebo procesu, do nichž se může zahrnout služba, hardware, zpracované materiály, software a jejich kombinace. [2]

Všeobecně produktem může být jakýkoliv výstup projektu, například informační strategie, definice uživatelských požadavků, poptávkový dokument, datový model, model procesů apod.. Některé produkty lze aplikovat do výchozích postupů dalších projektů nebo je lze použít při jiných aktivitách, které už nejsou řízeny formou projektů.

1.3 Projektový trojúhelník (trojimperativ)

Jednou z několika nezbytných součástí projektového managementu je i projektový trojúhelník, který znázorňuje omezení na úkor kvality. Základní myšlenkou po čas dosahování cílů projektu je znázorňování vztahu mezi dostupnými zdroji a kvalitou hotového produktu. Jednotlivé strany trojimperativu spolu navzájem korespondují a při vykonání určité změny na jedné ze tří stran se pro dosažení cíle změní i zbylé dvě strany (např. změna harmonogramu v podobě zpoždění nebo prodloužení doby realizace se odrazí na nákladech a rozsahu celého projektu.).



Obr. 1. Projektový trojúhelník

- Náklady – jde o plánovaný rozpočet, stanovující kolik bude realizace projektu stát. V některých situacích může docházet k navýšení nebo snížení nákladů. Proto by se mělo v raném stádiu vývoje odpovědět na otázky typu: Kdo bude mít na starost financování celého projektu? Jaká bude jeho celková cena? Je zadavatel schopen v případě navýšení nákladů přistoupit na zvýšení výsledné ceny a hodnoty projektu?
- Rozsahem rozumíme nejen obsažnost projektu, který je zapotřebí k dokončení softwarového produktu, ale i rozměr produktu zahrnující kvalitu a potřebné funkce,
- Čas, neboli „deadline“ je předem stanovený datum vydání hotového produktu. V tomto případě je nesmírně důležitá komunikace mezi zadavatelem a dodavatelem. Pakliže je vzájemná domluva nedostačující, dochází k oddálení termínu dodání. [3]

Udržení stran trojúhelníku v rovnováze se považuje za ideální stav. K tomu je zapotřebí dobře odhadnout situaci, činit kompromisy mezi cíli vztahujícími se k nákladům, rozsahu a času, také se snažit předcházet možným komplikacím v průběhu životního cyklu projektu. [4]

1.4 Životní cyklus projektu

Během vývoje se projekt nachází v různých etapách. Tyto se spolu navzájem pojí a tvoří tak celek, jenž nazýváme životní cyklus projektu.

Životní cyklus projektu je rozdělen do několika propojených projektových fází, které jsou kontrolovány vedením a poskytují následné úkony výkonné organizace. [5]

Etapy životního cyklu jsou navazující, občas i překrývající se časové sekvence znázorňující aktuální stav projektu. Pro přechod z jedné sekvence do druhé je potřeba splnit předem stanovené podmínky příslušné sekvence. Pojmenování jednotlivých fází se liší podle autora či publikace, ovšem charakteristika zůstává stejná.

1. zadání
2. analýza
3. návrh
4. implementace
5. testování a provoz

Podrobné definování a hlubší poznání výše uvedených fází životního cyklu lze vyhledat v příslušných publikacích o projektovém managementu. [6]

1.5 Metodiky vývoje software

Metodologie softwarového vývoje obsahuje určitá pravidla, postupy a nástroje k dosažení plně funkčního výsledku v podobě softwaru.

1.5.1 Tradiční metodiky

Jedná se o starší metodiky při vývoji softwaru, které byly kvůli své malé flexibilitě a poměrně velké složitosti nevyhovující k uspokojení potřebných požadavků při vývoji nových projektů. Jako první tyto problémy zaznamenaly především drobné projekty a malé týmy zejména v dokumentacích, revizích a schvalování. Tyto zmíněné problémy nepomohla vyřešit ani patřičná úprava těchto metodik, a proto bylo nutné učinit nekompromisní změnu v návrhu vývoje softwaru, a to vznik agilních metodik, kterým se budu podrobně věnovat v následující části.

Existuje několik zástupců řídících se touto metodikou, jako například vodopádový model, který se považuje za nejstarší a dnes již není moc používán. Naopak spirálový model vhodný pro větší a komplexní projekty či Rational Unified Process členěný do několika na sebe navazujících cyklů. [14]

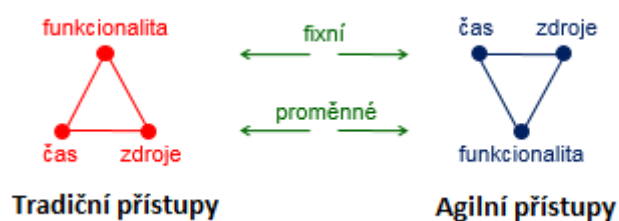
1.5.2 Agilní metodiky

V tomto případě jde o novější softwarový vývoj, při kterém je kladen důraz na zjištění požadavků ze strany zákazníka a rychlé zavedení produktu do provozu. Je nezbytné udržovat stálou komunikaci, jak se zákazníkem, tak i se členy týmu, kteří se na vývoji podílí. Vzhledem k velké konkurenci je majoritním úkolem každé firmy dodat hotový software s minimálními náklady v co nejkratším možném čase.

Rozlišujeme dva různé druhy časů. První, celkový čas, nám začíná při zadávání vstupních požadavků ze strany zákazníka a končí při dodání kompletně zhotoveného softwaru do rukou

zadavatele. Druhým a ekonomicky výhodnějším časovým úsekem je předání částečně fungujícího softwaru, který může zákazník aplikovat při své práci a obdržet za to určitý užitek v podobě příjmu. Tento počáteční zisk tak sníží další náklady potřebné pro dokončení zbylých částí produktu. Významnou výhodu můžeme sledovat i na druhé straně, kdy se ve formě zpětné vazby vrací užitečné informace k vývojovému týmu a tím se usnadní a urychlí cesta za dokončením softwaru. [14]

V předchozích odstavcích byla popsána agilní metodika. Nyní uvedu hlavní rozdíly oproti tradičním přístupům, jenž jsou znázorněny na obrázku z přednášky. [7]



Obr. 2. Porovnání tradičního a agilního přístupu

Na obrázku (Obr. 2) je znázorněna důležitost funkcionality u jednotlivých přístupů. Agilní přístupy označují funkcionalitu jako proměnnou hodnotu, kterou není potřeba v plné míře dosáhnout. Důležitějším úkolem je zde splnění dodacího termínu a uplatnění předem stanovených nákladů než funkčnost celého projektu.

Tradiční přístupy oproti agilním zaměřují fixní a proměnné položky. V tomto případě je na prvním místě zařazena implementace funkčních částí na úkor výskytu určitých rizik vedoucích ke zdražení projektu nebo oddálení termínu.

2 METODY ODHADOVÁNÍ

V této kapitole se budu podrobně zabírat problematikou týkající se odhadovacích metod. Většina zákazníků vyžaduje předběžné stanovení celkových nákladů, času a úsilí již před zahájením projektu. V závislosti na délce projektu a množství požadavků se uplatňují různé druhy odhadovacích metod. Nejdříve si definujeme, co to dohadování je.

2.1 Odhadování

Odhadováním se rozumí získání stanovené částky nákladů a doby trvání, jak u softwarových, tak i u jiných projektů. Pod slovem náklady si v tomto případě můžeme představit několik různých položek, které tvoří finanční hodnotu projektu. Jelikož je při realizaci softwaru nejdůležitější vývojářský personál, je proto jasné, že největší a zároveň i nejdůležitější položkou budou náklady na mzdy zaměstnanců. Nesmíme opomenout provozní náklady vynaložené k chodu kanceláří, oddělení či budov, jenž jsou rovněž zahrnuty v nákladech. Provozními náklady jsou myšleny výdaje za energii, daně, poplatky mzdové a ostatní osobní náklady. Při použití externího softwaru nebo hardwaru je nutno počítat s výdaji za licenci, která může být dočasná (pronájem licence) nebo trvalá (zahrnuta v ceně počítače). [8]

K úspěšnému dokončení projektu je při odhadování jedním z důležitých faktorů i jeho přesnost.

2.1.1 Přesnost odhadu

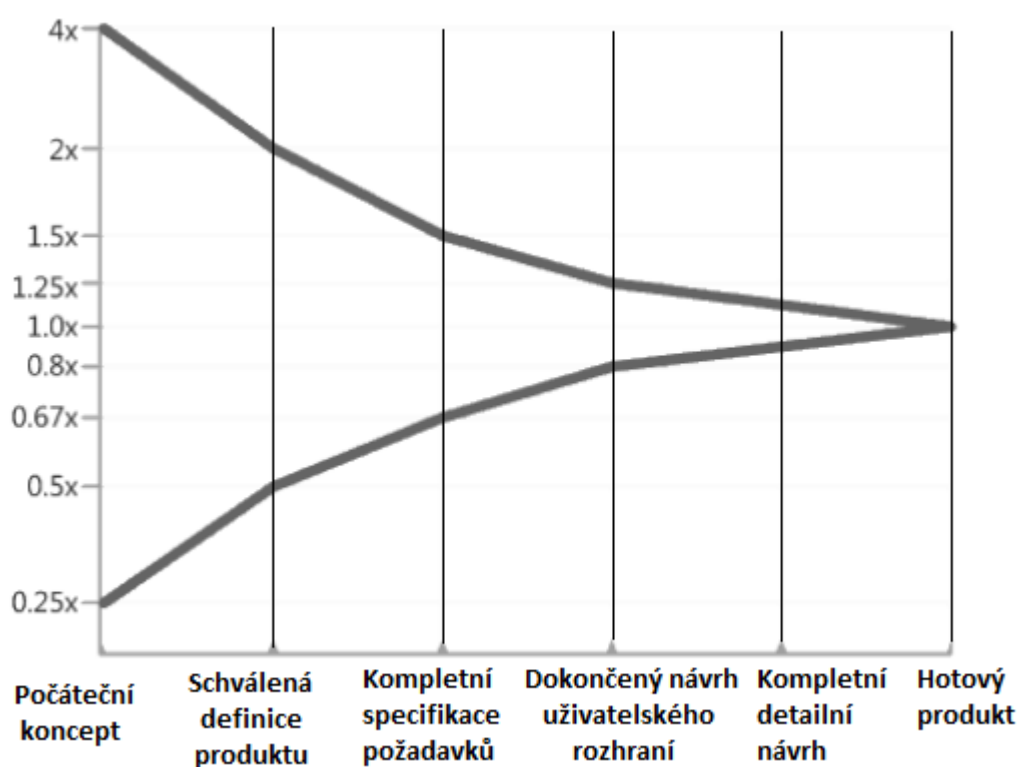
Definice vhodného odhadu není jednoznačně stanovená ani dle expertů popisujících tuto problematiku. Všeobecné příčiny špatného odhadu vycházejí především z nedostatku poskytnutých informací o zadaném projektu ale i z nedostatečné míře zkušeností pověřené osoby, která může provést nepřesný odhad. Podle [8] se pravděpodobnost úspěšného odhadu zvyšuje s nabírajícím množstvím zkušeností a informací průběhem celého životního cyklu projektu.

Společnosti mohou za přítomnosti přesných odhadů vytěžit výhody, které zvýší kvalitu celého projektu:

- Vzrostlá kvalita – předcházením nereálného odhadu se lze vyhnout nežádoucím změnám na kvalitě produktu,
- Efektivnější rozpočtování – jednodušší plánování finančního rozpočtu,
- Snazší pochopení stavu – možnost kdykoliv zjistit aktuální stav vývoj,

- Časné rozpoznání rizika – předběžná (průběžná) kontrola a odhalení nepřesných odhadů, která zamezí např. růstu nákladů nebo časové prodlevě,
- Striktně daný vývojový tým – složený z určitých osob pracujících na projektu od jeho zadání až po implementaci a závěrečné předání,
- Přeplánování aktivit během projektu – při nesprávně odhadnutých softwarových aktivitách.

Při absenci přesných odhadů dochází tedy k nárůstu časových i finančních rezerv a v nejhorším případě až k selhání celého projektu. Ve většině případů k tomu dochází i z nedostatku informací, ať už o projektu samotném nebo o organizaci a vedení práce týmu. Nemalým přínosem pro kvalitu odhadování napomáhají znalosti a zkušenosti osob pověřených ke stanovení odhadů. Názornou ukázkou je obrázek (Obr. 3), jenž přesně vystihuje průběh životním cyklem projektu v jednotlivých fázích vývoje (osa x) a hodnotu odchylky (osa y). [8]



Obr. 3. Kužel přesnosti odhadu

3 ALGORITMICKÉ METODY

Z názvu může být patrné, že se jedná o určitý druh metod, které společně využívají matematické modely k produkci odhadů. Tyto metody se používají hlavně u středních a větších projektů, při kterých je potřeba podrobné plánování nejen kvůli rostoucí obtížnosti, ale i kvůli většímu počtu členů v týmu s převládající konformitou.

3.1 COCOMO 81

Tento algoritmický model uvedl k životu americký softwarový inženýr Barry W. Boehm roku 1981. Metodu lze rozdělit do tří základních vývojových skupin (Organic, Semi-detached a Embedded) podle druhu projektu, dostupného množství nákladu a času realizace. O několik let byl nahrazen novější verzí, jenž se používá pro odhadování do dnes. [10]

Organické projekty (Organic): Na relativně malých projektech pracuje jen několik málo zkušených členů vývojového týmu. Řadíme sem projekty, u kterých předpokládáme malé náklady, krátkou dobu realizace, jednoduché požadavky apod..

Přechodné projekty (Semi-detached): Na pomyslné stupnici složitosti se jedná svým rozsahem a složitostí o průměrné projekty nebo části projektu. Vyžaduje se zde spolupráce většího počtu členů s dostatečnou úrovní komunikace za výskytu složitějších požadavků ze strany uživatele.

Složité projekty (Embedded): Řadíme sem především projekty s obtížnými požadavky jako jsou např. termín dodání, špatná komunikace se zákazníkem, špatně specifikované požadavky apod.. [9]

3.2 COCOMO II

COCOMO II (The Constructive Cost Model II) představil Barry Boehm v roce 1995 jako algoritmickou metodu pro zjištění nákladů softwarových projektů. Tento model navazoval na předcházející model COCOMO z roku 1981 a splnil potřebné důvody nahrazení, jež byla nedostačující plnění potřebných funkcí pro danou dobu. [9]

Základní rovnice pro modely sloužící k odhadu nákladů je odvozena a upravena z předcházejícího modelu COCOMO. Má následující tvar:

$$PM = A \times (KSLOC)^B \quad (3.1)$$

kde:

PM (person months) jednotka člověkoměsíc vyjadřuje množství času jedné osoby, která vynaloží úsilí pro vývoj softwaru v jednom měsíci,

A – konstantní hodnota 2.94 podle [13],

KSLOC – udává počet řádků kódu v tisících převedením neupravených funkčních bodů (UFP), jejichž význam je definován v kapitole 3.3.

B – konstantní hodnota podle Boehma u organických projektů je rovna 1.05, u přechodných 1.12 a u složitých 1.20. Význam přiblížím ve vzorci (3.9).

3.2.1 Modely odhadů

COCOMO II zahrnuje podle [9] několik základních modelů závisejících na porozumění produktu a projektových jednotek:

1. *Application Composition Model* úspěšně odhadováním vystihuje náklady a námahy, které se týkají výroby prototypů s vysokým rizikem např. uživatelské rozhraní, softwarový systém, výkonnost nebo splatnost použitých technologií.
2. *Early Design Model* lze aplikovat v raném stádiu vývoje produktu, při kterém ještě není složení výsledné architektury stanoveno.
3. *Post-Architecture model* odhadů nákladů používáme jedině v případech, kdy jsme si naprosto jistí složením, důkladným pochopením rysů a prostředím softwarového produktu.

Application Composition Model (ACM)

Je vhodnou volbou při znázornění potřebné námahy k vývoji systému. Celková námaha je stanovena na základě objektových bodů (tzv. Objective points – OP), které nám udávají konečnou velikost softwaru. Použití termínu objektové body zahrnuje i definici tzv. objektů, podle nichž je počet objektových bodů stanovený. Těmito body jsou počty obrazovek, počty informujících zpráv i množství 3GL komponentů. Každý objekt je charakterizovaný jako jednoduchý, střední a obtížný v závislosti na hodnotách příslušných rozměrů. Tuto charakteristiku můžeme sledovat v tabulce (Tab. 1). [9] [13]

Tab. 1. Složitost objektových bodů u ACM modelu

Počet objektů	Množství a zdroj datových tabulek		
	< 3	3 – 7	> 8
< 3	jednoduchý	jednoduchý	střední
3 – 7	jednoduchý	střední	obtížný
> 8	střední	obtížný	obtížný

Výše uvedenou tabulku je nutno převést do kvantitativních (číselných) jednotek. Hodnoty v tabulce (Tab. 2) znázorňují relativní úsilí potřebné k realizaci.

Tab. 2. Vyjádření námahy pro objekty modelu ACM

Typ objektů	Míra complexity		
	jednoduchý	střední	obtížný
obrazovka	1	2	3
zpráva	2	5	8
3GL komponenty			10

Množství objektových bodů získáme součtem všech hodnot objektových instancí. Tento součet můžeme poté aplikovat u závěrečného výpočtu člověkoměsíce podle vzorců (3.2) a (3.3):

$$NOP = \frac{OP \times (100 - R)}{100} \quad (3.2)$$

$$PM = \frac{NOP}{PROD} \quad (3.3)$$

kde:

NOP udává počet objektových bodů,

R představuje míru použitelnosti kódu,

PROD je míra produktivity vyjádřena tabulkou (Tab. 3).

Tab. 3. Míra produktivity pro model ACM

Zkušenosti, schopnosti a zralost vývojářů	Míra produktivity				
	velmi nízká	nízká	nominální	vysoká	velmi vysoká
PROD	4	7	13	25	50

Early Design Model (EDM)

Používá se zejména při počátečních fázích softwarových projektů, kdy je obtížné stanovit velikost vyvíjeného produktu, vlastnosti cílové platformy, přístup vývojového týmu a podrobnou specifikaci procesů. Tento model používá pro výpočet odhadu upravený vzorec (3.2), u kterého kalkuluje mimo jiné i se sedmi různými atributy produktu. Každý z těchto atributů pro EDM odpovídá kombinaci atributů pro *Post-Architecture Model (PAM)* znázorněné v tabulce (Tab. 4).

Tab. 4. Kombinace EDM faktorů s PAM modelem

EDM faktor	Protější kombinace k PAM modelu
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

Význam sedmi atributů produktu podle [9]:

- RCPX (product reliability and complexity) – složitost a spolehlivost produktu
- RUSE (required reuse) – míra opětovného použití
- PDIF (platform difficulty) – složitost platformy
- PERS (personal capability) – schopnosti personálu

- PREX (personnel experience) – zkušenosti personálu
- FCIL (facilities) - zařízení
- SCED (schedule) – plánování

ED model používá tedy pro výpočet úsilí následující rovnici:

$$PM = A \times (KSLOC)^B \times EM \quad (3.4)$$

$$EM = \prod_{i=1}^7 EM_i + PM \quad (3.5)$$

kde:

A je konstanta = 2.94 podle Boehma,

$KSLOC$ udává počet řádků kódu v tisících převedením neupravených funkčních bodů (UFP). Tabulka (Tab. 5) obsahuje převedené jednotky z UFP na SLOC.

B je exponent, který přibližně specifikuje následující model PAM,

EM_i je určený sedmi různými projektovými atributy v závislosti na stupni hodnocení. Tento multiplikátor podrobněji znázorňuje tabulka (Tab. 6).

Tab. 5. Převod funkčních bodů do řádků kódu

Programovací jazyk	SLOC/UFP
Ada	71
AI Shell	49
APL	32
Assembly	320
C	128
C++	29
Fortran 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

Tab. 6. Nominální hodnoty faktorů pro EDM model

Faktory	Hodnota						
	Extrémně nízká	Velmi nízká	Nízká	Nominální	Vysoká	Velmi vysoká	Extrémně vysoká
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE			0.95	1.00	1.07	1.15	1.24
PDIF			0.87	1.00	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62
SCED		1.43	1.14	1.00	1.00	1.00	

Post-Architecture model (PAM)

PAM je z uvedených tří modelu nejdetailnější. Používá se pro zjištění odhadů rozsahu v závěrečném stádiu životního cyklu softwaru, kdy je již známa jeho architektura. Podle [9] používá stejný vzorec jako EDM se změnou hodnoty indikující počet faktorů:

$$EM = \prod_{i=1}^{17} EM_i \quad (3.6)$$

kde:

EM_i je multiplikátor zastupující součin sedmnácti atributů, které jsou uvedeny níže.

$$PM = A \times [KSLOC]^B \times EM \quad (3.7)$$

$$SIZE = Size \times \left(1 + \frac{BRAK}{100}\right) \quad (3.8)$$

Pro výpočet parametru b , který jsem použil při výpočtu úsilí u předešlé metody odhadu, použiji níže uvedený vztah:

$$b = 1.01 + \frac{1}{100} \times \sum_{j=1}^5 SF_j \quad (3.9)$$

kde:

b je exponent, jehož hodnota podle [15] je úzce spjata s výkonností a velikostí produktu. Mohou nastat celkem tři situace kdy:

- $b < 1$: Stav, při kterém projekt nevykazuje žádné ztráty. Platí přímo úměrný vztah. Pokud je zvýšená výnosnost produktu, zvýší se i jeho velikost,
- $b = 1$: V tomto případě jsou ztráty a zisk v rovnováze. Jedná se o lineární model, který se vyskytuje spíše u malých projektů,

- $b > 1$: Situace, jenž vykazuje značné ztráty. Ve většině případů je to dopadem zhoršené komunikace u velkých projektů anebo špatné propojení jednotlivých částí tvořící výsledný produkt.

SF (Scale factor) – proměnná zastupující jeden z pěti faktorů. Tyto určují různé situační části projektu a jsou rozepsány v tabulce (Tab. 7).

Tab. 7. Nominální hodnoty faktorů pro PAM model

Faktory	Hodnoty faktorů pro EDM a PAM					
	Velmi nízký	Nízký	Nominální	Vysoký	Velmi vysoký	Extrémně vysoký
PREC	4.05	3.24	2.43	1.62	0.81	0.00
FLEX	6.07	4.86	3.64	2.43	1.21	0.00
RESL	4.22	3.38	2.53	1.69	0.84	0.00
TEAM	4.94	3.95	2.97	1.98	0.99	0.00
PMAT	4.54	3.64	2.73	1.82	0.91	0.00

Vlastnosti těchto faktorů podle [9] nyní podrobně rozepíši:

1. **PREC** (Precedentedness) – návaznosti na předchozí projekty. Čím více je produkt s již dříve vyvinutými projekty totožný, tím větší je numerická hodnota faktoru.
2. **FLEX** (Development flexibility) – flexibilita vývoje. Vyjadřuje míru, do jaké se může produkt přizpůsobit požadavkům a externí specifikaci rozhraní.
3. **RESL** (Architecture / Risk resolution) – rozhodnutí architektury/rizika. Kombinuje odstraňování rizik a celistvost produktového designu.
4. **TEAM** (Team cohesion) – koheze týmu. Představuje míru soudržnosti týmu (uživatelé, zákazníci, správci, vývojáři atd) podílejícího se na vytváření produktu.
5. **PMAT** (Process maturity) – míra vyspělosti procesu, kterou určuje Capability Maturity Model (CMM) vydaný organizací softwarového inženýrství. Rozlišujeme celkem pět úrovní vyspělosti a osmnáct různých oblastí tzv. Key Process Areas - KPA. Ke zjištění míry vyspělosti musí cílová organizace spojit svoje dovednosti KPA s vhodnou úrovní vyspělosti CMM. Vztahy KPA k CMM přehledně znázorňuje tabulka (Tab. 8).

Tab. 8. Zařazení jednotlivých oblastí do úrovně vyspělosti

Úroveň CMM	Key process area
Level 1 (Initial)	Žádný
Level 2 (Repeatable)	Requirements Management Software Project Planning Software Project Tracking and Oversight Software Subcontract Management Software Quality Assurance Software Configuration Management
Level 3 (Defined)	Organization Process Focus Organization Process Definition Training Program Integrated Software Management Software Product Engineering Intergroup Coordination Peer Reviews
Level 4 (Managed)	Quantitative Process Management Software Quality Management
Level 5 (Optimizing)	Defect Prevention Technology Change Management Process Change Management

Atributy z předešlého modelu (EDM) odpovídají atributům v modelu PAM kde je nutné pro definování jejich numerických hodnot znát informace o softwarovém produktu, o jeho architektuře, vývojářském týmu a o kladných nefunkčních požadavcích. Hodnoty těchto multiplikátů pro PAM můžeme vidět v následující tabulce (Tab. 9). [9]

Tab. 9. Hodnoty multiplikátorů pro model PAM

Faktory	Hodnota					
	Velmi nízká	Nízká	Nominální	Vysoká	Velmi vysoká	Extrémně vysoká
RELY	0.82	0.92	1.00	1.10	1.26	
DATA		0.90	1.00	1.14	1.28	
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.06	1.23	
TIME			1.00	1.11	1.29	1.63
STOR			1.00	1.11	1.17	1.46
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.42	1.19	1.00	0.85	0.71	
PCAP	1.34	1.15	1.00	0.88	0.76	

PCON	1.29	1.12	1.00	0.90	0.81	
APEX	1.22	1.10	1.00	0.88	0.81	
PLEX	1.19	1.09	1.00	0.91	0.85	
LTEX	1.20	1.09	1.00	0.91	0.84	
TOOL	1.17	1.09	1.00	0.90	0.78	
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	

Boehm [9] uvádí existenci čtyř základních skupin, jenž rozdělují multiplikátory podle oblasti uplatnění:

Softwarové atributy:

RELY (Required Software Reliability) – požadovaná spolehlivost softwaru

DATA (Database Size) – velikost databáze

CPLX (Product Complexity) – míra složitosti produktu

RUSE (Developed for Reusability) – míra opětovného použití

DOCU (Documentation Match to Life-Cycle Needs) – potřebná dokumentace

Hardwarové atributy:

TIME (Execution Time Constraint) – míra požadavků na dobu odezvy

STOR (Main Storage Constraint) – míra použitelnosti paměti

PVOL (Platform Volatility) – nestálost platformy

Lidské atributy:

ACAP (Analyst Capability) – míra schopnosti analytiků

PCAP (Programmer Capability) – míra kvality programátorů

PCON (Personnel Continuity) – míra personální kontinuity

APEX (Applications Experience) – míra zkušeností programátorů s obdobnými aplikacemi

PLEX (Platform Experience) – míra zkušenosti s danou platformou

LTEX (Language and Tool Experience) – míra zkušeností v programovacím jazyce

Projektové atributy:

TOOL (Use of Software Tools) – míra použití nástrojů pro vývoj

SITE (Multisite Development) – míra vícemístného vývoje

SCED (Required Development Schedule) – doba realizace neboli časový plán

3.2.2 Odhad doby vývoje

Kromě odhadnutí potřebného úsilí k vývoji softwaru je pro vývojáře potřeba si stanovit předběžnou dobu, nezbytnou k realizaci. K odhadnutí podle [9] jsou zapotřebí následující vztah:

$$TDEV = [A \times PM^{(0.33+0.2 \times (B-1.01))}] \times \frac{SCED\%}{100} \quad (3.10)$$

TDEV je časová hodnota udávaná v kalendářních měsících,

PM je hodnota potřebného úsilí kterou jsme získali u Early design modelu v jednotkách člověkoměsíc,

SCED% zastupuje procentuální multiplikátor komprese/expanze časového plánu, který je k nahlédnutí v tabulce (Tab. 10).

Tab. 10. Míra komprese/expanze SCED faktoru

Faktor	Míra SCED faktoru					
	Velmi nízká	Nízká	Nominální	Vysoká	Velmi vysoká	Extrémně vysoká
SCED	75% z nominální hodnoty	85%	100%	130%	160%	

3.3 Odhad FPA

Model pro algoritmické odhadování softwaru FPA – Function Point Analysis, představil roku 1979 americký softwarový inženýr Allan Albrecht. Tato metoda je jedinečná díky používání tzv. funkčních bodů, jenž nevycházejí z množství řádků tak, jak tomu bylo u metody COCOMO II, nýbrž zohledňují funkcionalitu celého programu.

Model FPA nepopisuje konkrétní postup jak dosáhnout výsledku odhadu ale nabízí řadu několika možných postupů závislých na druhu konečného produktu, jenž budeme vytvářet. Při realizace je uživatelem nejdříve vytvořen detailní popis požadavků, které následně vývojáři „konvertují“ do programovacího jazyka. [16]

3.3.1 Neupravené funkční body (UFP)

Na začátku jsou všechny požadavky od uživatele zpracovány, blíže specifikovány a zařazeny podle [17] do dvou základních kategorií funkcí :

1. Transakční s funkčními body:

- External inputs (Externí vstupy - EI)
Jde o elementární proces, při kterém dochází ke změně interních dat aplikace. Průběh procesu zpracovává vstupní data na cestě z vnějšího do vnitřního prostředí. Používají se především pro kontrolování vnitřních logických jednotek.
- External outputs (Externí výstupy - EO)
Jedná se rovněž o elementární proces jako v předchozím případě u EI se změnou směru toku uživatelských a řídicích dat. Ty putují z vnějšího prostředí do vnitřního. Výstupem se v této situaci myslí různá hlášení, zprávy, grafy a jiné. Názorným příkladem takového výstupu může být situace, kdy uživatel chce potvrdit uhrazení objednávky ale systém mu to neumožní z důvodu nedostatku finančních prostředků na účtě. Sdělení které uživatel obdrží od systému je externím výstupem.
- External inquiries (Externí dotazy - EQ)
Vnější dotazy jsou elementární procesy, které svým působením nemění vnitřní strukturu programu. EQ je kombinací externích vstupů a výstupů, u nichž je uživatel vyzván k zadání vstupu.

2. Datové, kde jsou zahrnuty:

- Internal logical files (Interní logické soubory - ILF)
Jsou uživatelem dostupné záznamy v rámci aplikace, které jsou generovány a používány aplikací pomocí externích vstupů. Pod záznamy si můžeme představit např. většinou logické entity, pracovní soubory, tabulky,
- External interface files (Soubory externího rozhraní - EIF)
Řadíme sem společné databáze a jiné záznamy. Jelikož jsou tyto soubory sdíleny více aplikacemi, pak je aplikace mohou používat a nebo na ně odkazovat.

Pro výpočet celkové hodnoty neupravených funkčních bodů platí, že nejdříve všech pět výše vypsanych funkčních bodů vynásobíme příslušnou váhou k nahlédnutí v tabulce (Tab. 11), poté výsledné hodnoty podle [16] sečteme. Získáváme vztah:

$$UFP = \sum_{i=1}^5 B_i \times m_i \quad (3.11)$$

kde:

B_i je jeden z funkčních typů (EI, EO, EQ, ILF, EIF)

m_i je multiplikátor indikující váhu pro příslušný funkční bod podle tabulky (Tab. 11).

Tab. 11. Váhové multiplikátory pro jednotlivé funkční typy

Funkční typ	Míra complexity		
	Nízká	Průměrná	Vysoká
EI	X 3	X 4	X 6
EO	X 4	X 5	X 7
EQ	X 3	X 4	X 6
ILF	X 7	X 10	X 15
EIF	X 5	X 7	X 10

3.3.2 Upravené funkční body

Po vypočítání neupravených funkčních bodů je následujícím krokem k určení celkového nákladu metodou FPA zjištění upravených funkčních bodů (AFP). Hodnotu neupravených funkčních bodů násobíme tzv. adjustačním faktorem (VAF), ve kterém je zohledněno 14 hlavních systémových charakteristik. Těmto vlastnostem je uživatelem přiřazována hodnota (od 0-nejnižší význam do 5-nejvyšší význam) v závislosti na míře důležitosti. [17]

Vztahy pro výpočet upravených funkčních bodů podle [16] jsou:

$$AFP = UFP \times VAF \quad (3.12)$$

$$VAF = (TDI \times 0.01) + 0.65 \quad (3.13)$$

$$TDI = \sum_{i=1}^{14} DI_i \quad (3.14)$$

kde:

UFP je počet neupravených funkčních bodů

VAF (value adjustment factor) je hodnota adjustačního faktoru

DI (degree influence) je číselný ukazatel míry důležitosti jednotlivé charakteristiky

TDI (Total degree influence) – celkový součet všech měr důležitostí

Hlavní systémové charakteristiky

Jak již bylo uvedeno v předešlém odstavci, existuje 14 základních systémových otázek charakterizující celkovou funkčnost a komplexitu aplikace, na které je nutné správně odpovědět. Mezi tyto otázky podle [18] řadíme:

1. Datová komunikace (*Data Communications*)

Kolik komunikačních zařízení máme k dispozici při převodu informací mezi aplikacemi nebo systémem?

2. Distribuované zpracování dat (*Distributed Data Processing*)

Jak jsou rozděleny parametry a probíhající funkce?

3. Výkon (*Performance*)

Je uživatel spokojený s časem odezvy?

4. Plné využití konfigurací (*Heavily Used Configuration*)

Jak často se používá cílová platforma, na které poběží aplikace?

5. Rychlost transakcí (*Transaction Rate*)

Jak často se provádějí transakce (denně, týdně, měsíčně apod.)

6. Vstup dat online (*Online Data Entry*)

Jaké procento z celkového počtu dat je vloženo online?

7. Účinnost koncového uživatele (*End-User Efficiency*)

Byla aplikace určena pro koncového uživatele?

8. Aktualizace online (*Online Update*)

Kolik interních logických souborů je aktualizováno online?

9. Celkové zpracování (*Complex Processing*)

Má aplikace rozsáhlé matematické nebo logické zpracování?

10. Znovupoužitelnost (*Reusability*)

Uspokojuje tato aplikace jednu nebo více uživatelských potřeb?

11. Snadnost instalace (*Installation Ease*)

Jak je obtížná konverze a instalace aplikace?

12. Jednoduchost provozu (*Operational Ease*)

Jak efektivní jsou startovací, zálohovací a obnovovací funkce?

13. Orientovaná na více míst (*Multiple Sites*)

Je aplikace navržena, vyvinuta a podporována více webovými místy?

14. Jednoduchost změn (*Facilitate Change*)

Přizpůsobí se aplikace některým změnám?

3.3.3 Odhad doby realizace

Díličními výsledky z předchozích kapitol lze pomocí metody FPA odhadnou přibližná velikost softwaru a přibližná doba potřebná k zhotovení produktu. Nejdříve je zapotřebí si definovat vztah pro výpočet hodnoty použitých řádků zdrojového kódu na jeden funkční bod (SLOC/AFP). Platí následující vzorec podle [11]:

$$SLOC = AFP * SLOC/UFP \quad (3.15)$$

Kde:

SLOC/UFP je faktor udávající převedené funkční body do počtu řádků kódu podle Tab. 5.,

AFP zastupuje upravené funkční body.

K zjištění přibližné doby odhadu je potřeba si nejdříve vypočítat hodnotu vynaloženého úsilí daného projektu. Podle [12] platí několik různých vztahů, které závisí na dostupných datech, jenž jsme získali v průběhu odhadování:

$$E = c \times AFP + b \quad (3.16)$$

$$E = AFP / (AFP/m) \quad (3.17)$$

$$E = c \times AFP^b \quad (3.18)$$

kde:

b je konstantní hodnota, kterou jsem blíže specifikoval u vzorce (3.1),

c je konstantní hodnota 2.5,

AFP/m je zástupcem odhadu přibližných funkčních bodů za jeden měsíc.

Nyní již máme veškeré potřebné údaje k výpočtu přibližné doby realizace v kalendářních měsících. Podle ISBSG platí následující vztah:

$$TDEV = 0.38 \times E^{0.37} \quad (3.19)$$

kde:

$TDEV$ je hodnota v kalendářních měsících.

II. PRAKTICKÁ ČÁST

4 POROVNÁNÍ ALGORITMICKÝCH METOD

Jedním z cílů mé bakalářské práce bylo zjistit výhody a nevýhody jednotlivých algoritmických metod odhadování software.

Cocomo 81

Přestože je tento model nejstarší ze všech algoritmických metod odhadování, tak i on má několik svých unikátních výhod. Těmito výhodami jsou:

- možnost přidání některých jedinečných faktorů a jejich aplikace při výpočtu,
- hodí se pro malé, střední i velké projekty,
- díky dlouholetým zkušenostem garantuje velkou vyváženost.

Hlavní nevýhodou u COCOMO 81 a taky hlavním důvodem jejího nahrazení novou metodou COCOMO II je neumožnění znovupoužití některých částí kódu. Podle mého názoru existuje i několik dalších důležitých vlastností, které nezahrnuje:

- ignoruje potřebnou dokumentaci,
- nezahrnuje vlastnosti zákazníků a dovednosti vývojářů,
- nespolupracuje s vývojovým prostředím.

COCOMO II

Jelikož byl tento model používán po několik desítek let, je zřejmé, že výhod bude o poznání více než nevýhod. Mezi výhody bych v tomto případě zařadil:

- znovupoužitelnost kódu a aplikace funkčních bodů,
- jednoduchost a objektivnost,
- standard při výpočtech u řady firem,
- lehká dostupnost základních informací týkajících se projektu.

Jednu z nejdůležitějších dysfunkcí, na které jsem narazil během podrobné studie COCOMO II modelu, je použití vodopádového modelu při vývoji softwaru. Jelikož se jedná o nejstarší model životního cyklu softwaru, tak se domnívám, že byl použit z důvodů dostupnosti v tehdejší době, kdy model COCOMO II vznikl. Vodopádový model zamezuje pokračování v další fázi vývoje, aniž by byla dokončena předchozí fáze. Tato nevýhoda se vztahuje především na rozsáhlejší projekty, u kterých může uživatel v průběhu vývoje měnit požadavky. Mezi další nevýhody patří:

- vynaložení velkého množství času a práce při odhadování,

- nevhodnost pro malé projekty díky své složitosti.

Rozdíly mezi metodami COCOMO

V následujícím odstavci se pokusím rozvést hlavní rozdíly u jednotlivých metod.

1. Mladší model COCOMO II používá jednotku KSLOC pro odhadování velikosti softwaru, zatímco vývojově starší model COCOMO 81 počítá s KDSI (DSI – Delivered Source Instructions). V praxi to vypadá tak, že například při použití známé syntaxe if-then-else se počítá jako jeden celek u SLOC, ale jinak je tomu u DSI, kde tato hodnota roste o několik nominálních jednotek.
2. Přidání některých faktorů: DOCU, PVOL, LTEX, PLEX, RUSE, PCON, SITE
3. Odebrání faktorů: VIRT, TURN, LEXP, MODP, VEXP
4. Nové rozdělení modelu do tří vývojových fází (ACM, EDM a PAM)

FPA

Největší výhodou je bezesporu spolupráce s funkčními body, které byly touto metodou uvedeny v platnost. Výhoda spočívá v možnosti použít jakýkoliv libovolný programovací jazyk a pomocí tabulky č. 5 převést softwarové části kódu na funkční body, sloužící k realizaci celého produktu. Existuje několik dalších výhod:

- Libovolná velikost programu (množství kódu),
- Nezávislost na zvoleném programovacím jazyku a použité technologii,
- Přesnost a spolehlivost výsledku na rozdíl od metod, které používají metriku kódu LOC,
- Možnost použít FPA metodu v raném počátku životního cyklu vývoje produktu.

Z hlediska dysfunkcí považuji za největší nevýhodu subjektivní hodnocení, jenž metoda FPA vyžaduje. Ve velké míře záleží na pohledu odhadujícího analytika, jak se k zadanému úkolu postaví a jakým způsobem ho ohodnotí. Mezi nevýhody mimo jiné patří:

- Výpočet je často velmi obtížný
- Kvalita výstupu je přehlížena ve velké míře

5 ODHAD POMOCÍ METODY FPA

Jedním z cílů mé bakalářské práce bylo zaměřit se na metodu funkčních bodů, a proto jsem si tuto metodu vybral ve své praktické části, ve které budu ilustrovat tuto problematiku na názorném příkladu.

Aplikace StoreQ

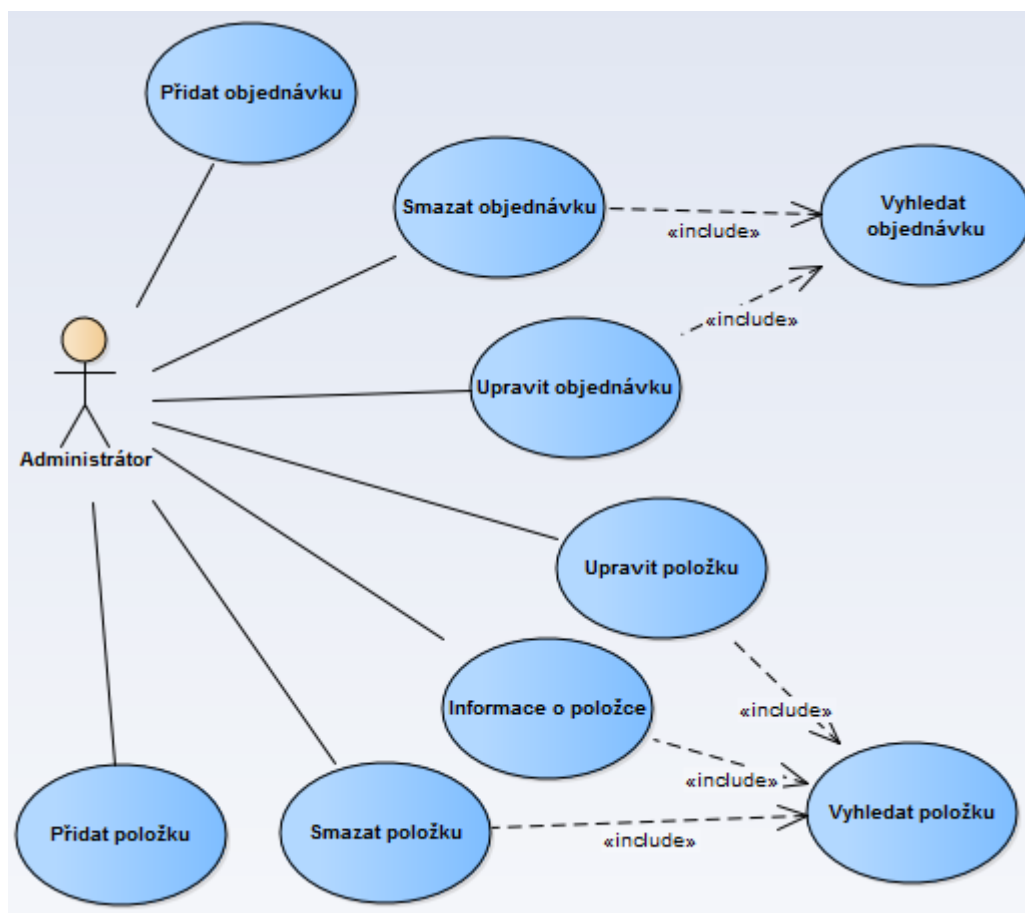
Modelový případ StoreQ bude evidovat interní záznamy malé firmy o stavech produktů na skladě. Rozhraní bude mít funkci nahlížet do záznamů a provádět editaci, smazání či přidání záznamů. Dále bude možnost seřadit jednotlivé záznamy podle času uskladnění a aktuálního počtu na skladě. Nezbytnou funkcí bude řízení objednávek, které půjdou opět editovat, přidávat či mazat, měnit stav (objednané, vyřizuje se, odeslané, převzato). Vše se bude odehrávat v reálném čase. Veškerá funkčnost je dostupná po přihlášení do systému. Monitoruje i sklady

Funkcionalita aplikace StoreQ:

- zobrazování, editování a zrušení objednávek a položek na skladě,
- zobrazování u vyhledaných položek, ve kterém skladě se nachází a jejich počet,
- možnost přidat položku i objednávku,
- seřazení podle času uskladnění, počtu a lokaci v rámci aplikace.

Sklady monitorované aplikací StoreQ:

- centrální sklad – Zlín,
- externí sklad – Otrokovice,
- externí sklad – Vizovice.



Obr. 4. Use Case model aplikace StoreQ

5.1 Výpočet neupravených funkčních bodů

Prvním krokem ke zjištění celkových nákladů a předběžné odhadnutí doby vývoje je kalkulace neupravených funkčních bodů. Výslednou hodnotu zjistíme součtem následujících třech datových funkcí:

Record element type (RET) – Je uživatelem rozpoznatelná podmnožina datových elementů, mezi které patří interní logické soubory nebo soubory externího rozhraní. Rozlišujeme dva druhy podmnožin:

Libovolné podmnožiny: Uživatel má na výběr, jestli použije jednu nebo žádnou z podmnožin během elementárního procesu, jenž přidá, či vytvoří nová data.

Povinné podmnožiny: Uživatel musí použít alespoň jednu podmnožinu, nemá na výběr jak v předchozím případě.

File type referenced (FTR) – Je interní nebo externí logický soubor, který je vedený transakční funkcí.

Data element type (DET) – Je unikátní uživatelem rozpoznatelné datové pole, které se neopakuje. DET může být nejenom kvantitativní, ale i kvalitativní. Kvantitativními daty rozumíme taková data, která můžeme číselně vyjádřit. Na druhé straně kvalitativní data jsou taková, jež číselně vyjádřit nemůžeme např. (text, fotografie, zvukový záznam apod.). Tento datový typ se používá u datových a transakčních funkcí.

Dále je nutné si osvojit význam některých prvků:

Elementární proces – je nejmenší jednotkou aktivity, přesto je pro uživatele smysluplná a významná.

Použitelnost výše zmíněných datových funkcí se mění v závislosti na komponentech, na které je chceme aplikovat. Komponenty se v tomto případě rozumí datové a transakční rozdělení (popsáno podrobně v kapitole 3.3.1.). Následující tabulka (Tab. 12) nám ukáže, kdy se jednotlivé funkce používají:

Tab. 12. Použití datových funkcí

Komponent	Datová funkce		
	RET	DET	FTR
Externí vstupy		Ano	Ano
Externí výstupy		Ano	Ano
Externí dotazy		Ano	Ano
Interní logické soubory	Ano	Ano	
Soubory externího rozhraní	Ano	Ano	

Podle IPFUG existují určitá pravidla pro počítání datových funkcí. Datový element (DET) reprezentuje v programu vždy nějaký druh pole (check button, command button, radio button etc.), které je rozpoznatelné uživatelem. Abychom došli ke správnému výsledku počítání datovaných funkcí DET, podle [20] musíme dodržet následující pravidla:

- Spočítat DET pro každé unikátní uživatelem rozpoznatelné a neopakovatelné pole, které je udržované nebo získané z ILF nebo EIF během vykonání elementárního procesu
- Pokud dvě aplikace udržují anebo odkazují na stejné ILF/EIF je nutné započítat pouze takové DET, které používá aplikace k odhadnutí velikosti ILF/EIF
- Spočítat DET pro každou část dat potřebných uživatelem k ustálení vztahu s ILF nebo EIF (spočítat cizí klíče)

- Spočítat DET pro každé uživatelem rozpoznatelné a neopakovatelné pole procházející přes hranice aplikace nebo nacházející se uvnitř aplikace. Dále je nutné určit, jakým způsobem byla data vytvořena
- Pokud je DET nacházející se uvnitř aplikace stejná jako DET, jenž prošlo směrem dovnitř přes hranice, počítá se pouze jednou jako elementární proces
- Pokud nastane situace, při které je pole rekurzivní, pak je toto pole započteno jako jedna datová funkce DET. Stejná situace platí i při vykonání více úkonů, majících stejnou funkcionalitu
- Jako DET se nepočítá: literály, stránkování proměnných nebo datové prvky generované systémem

RET funguje na mateřském principu. Smyslem je vždy přiřadit jeden druh záznamu k tomu druhému (např. v bankovníctví, kdy evidování čísel bankovních účtů bez přiřazení uživatele k číslu je irelevantní). I datová funkce RET má několik pravidel pro počítání:

- Spočítat RET u každé libovolné nebo povinné podmnožiny pro ILF nebo EIF
- Pokud neexistují žádné takové podmnožiny, pak se počítá ILF nebo EIF jako jedno RET

Poslední z datových funkcí je FTR, umožňující prostřednictvím vnějšího výstupu upravovat obsah interních logických souborů i souborů externího rozhraní. Funkce se řídí podle následujících pravidel:

- Spočítat FTR při každém čtení ILF nebo EIF během vykonávání elementárního procesu
- Spočítat FTR pro každý interní logický soubor, jenž je udržovaný během vykonávání elementárního procesu

Externí vstupy

Uživatelská data, která vstupují z vnějšího prostředí do vnitřního systému, se nazývají externí vstupy. Jestliže některé externí vstupy přidávají, upravují nebo mažou data ve vnitřním logickém souboru, pak tyto vstupy považujeme za transakce externího vstupu.

Všechny externí vstupy jsem analyzoval na základě jednotlivých případů užití a jejich funkcionality. Vycházel jsem z Use Case diagramu znázorněném na obrázku (Obr. 4). Následně jsem vypsál všechny operace externích vstupů, jenž aplikace StoreQ obsahuje a znázornil jsem je v tabulkách (Tab. 12) s příslušnou komplexitou funkcí (Tab. 13). [19]

Operace s objednávkami:

- Přidání objednávky – Při zhotovení každé objednávky bude vyžadováno vyplnění osmi polí (Jméno, Příjmení, Ulice, Město, PSČ, E-mail, Telefon a Objednané položky), které budou následně aplikací zkontrolovány, zda jsou vyplněná správně. K dispozici bude i zobrazení celkové ceny a ceny s příslušnou DPH. Dále bude mít zadavatel možnost objednávku uložit nebo ukončit,
- Upravení objednávky – Při editaci již existující objednávky jsem vycházel ze stejných datových polí jako u přidávání objednávky,
- Smazání objednávky – Pokud uživatel zvolí možnost smazat objednávku, zobrazí se pole se jménem, příjmením, objednanými položkami a potvrzovací zprávou. Následně bude aplikací vyzván, aby smazání potvrdil nebo zrušil.

Operace s produkty:

- Přidání produktů na prodejnu – U přidání položek na prodejnu bude uživatel vyzván k vyplnění čtyřech polí (název, počet kusů, popis, cena), jenž budou zkontrolovány pro správné vyplnění. Dále bude mít uživatel možnost objednávku uložit nebo ukončit
- Upravení produktů na prodejně – Při upravení evidovaných produktů na prodejně se uživateli zobrazí stejná datová pole jako u přidání produktu
- Smazání produktů na prodejně – Pokud uživatel zvolí možnost smazat produkt, zobrazí se pole s názvem produktu, počtem kusů, popisem a potvrzovací zprávou. Následně bude aplikací vyzván, aby smazání potvrdil nebo zrušil

Tab. 12. Externí vstupy pro aplikaci StoreQ

Operace	Počet DET	Počet FTR
Upravení objednávky	12	2
Smazání objednávky	5	2
Přidání objednávky	12	2
Upravení položky na prodejně	6	1
Smazání položky na prodejně	5	1
Přidání položky na prodejnu	6	1

Tab. 13. Komplexita funkce EI

FTR	DET		
	1 - 4	5 - 15	Více než 15
Méně než 2	Nízký	Nízký	Průměrný
2	Nízký	Průměrný	Vysoký
Více než 2	Průměrný	Vysoký	Vysoký

Externí výstupy

Obsahují data putující z vnitřního prostředí systému do vnějšího prostředí. Pomocí příkazů nebo jiného druhu datových výstupu se mohou aktualizovat data interních logických souborů. Hlavním zdrojem pro vytvoření takovýchto zpráv a výstupů jsou informace obsažené v jedné nebo několika interních logických souborech a souborech externího rozhraní.

Všechny externí výstupy jsem analyzoval na základě jednotlivých případů užití a jejich funkcionality. Vycházel jsem z Use Case diagramu znázorněném na obrázku (Obr. 4). Následně jsem vypsál všechny operace externích výstupů, které aplikace StoreQ obsahuje a znázornil jsem je v tabulkách (Tab. 14) s příslušnou komplexitou funkcí (Tab. 15). [19]

Zobrazení objednávek:

- Detail objednávky – Uživateli se zobrazí osm polí spolu s čistou a navýšenou cenou stejně jako u přidání objednávky zahrnuté v externích vstupech. V tomto případě vynecháme datové pole pro kontrolu správného vyplnění. Dále do celkové hodnoty DET nebude pro svoji duplicitu započteno tlačítko Uložit

Zobrazení produktů:

- Detail produktu – Všechny produkty se zobrazí s následujícími poli: Název produktu, Počet kusů, Popis a Cena. Doplnující informace o aktuální dostupnosti ve třech skladech budou zobrazovat jednotlivá pole ke každému skladu. Data každého skladu jsou uložena v externím souboru. Stejně jako u detailu objednávek, tak i zde nebude kalkulováno tlačítko Uložit

Tab. 14. Externí výstupy pro aplikaci StoreQ

Operace	Počet DET	Počet FTR
Detail objednávky	10	2
Detail položky	7	4

Tab. 15. Komplexita funkcí EO a EQ

FTR	DET		
	1 - 5	6 - 19	Více než 19
Méně než 2	Nízký	Nízký	Průměrný
2-3	Nízký	Průměrný	Vysoký
Více než 3	Průměrný	Vysoký	Vysoký

Externí dotazy

Přenosem externích dotazů je reagování na vstup formou generování konkrétního výstupu, při kterém se nemění chování systému. Tyto vstupní a výstupní procesy nezahrnují žádné matematické operace, neupravují data a nezasahují do interních logických souborů během plnění dotazu.

Všechny externí dotazy jsem analyzoval na základě jednotlivých případů užití a jejich funkcionality. Vycházel jsem z Use Case diagramu znázorněném na obrázku (Obr. 4). Následně jsem vypsál všechny operace externích dotazů obsažené v aplikaci StoreQ a znázornil jsem je v tabulkách (Tab. 16) s příslušnou komplexitou funkcí (Tab. 15). [19]

- Vyhledání položky na skladech – Uživatel bude mít možnost vyhledat položku pomocí identifikačního čísla, textu nebo ceny. Bude mít na výběr z možností prohledat alespoň jeden sklad. Před zobrazením výsledků je lze seřadit podle dostupného počtu kusů, podle abecedy anebo podle identifikačního čísla. K odeslání dotazu bude sloužit tlačítko Hledej
- Vyhledání objednávky – V tomto případě bude uživatel povinen vyplnit alespoň jedno z polí: Číslo objednávky, Telefonní číslo nebo E-mail zadavatele. Veškeré údaje budou odeslány tlačítkem Hledej

Tab. 16. Externí dotazy pro aplikaci StoreQ

Operace	Počet DET	Počet FTR
Vyhledání položky na skladech	8	4
Vyhledání objednávky	4	1

Interní logické soubory

Jde o všechny logické, uživatelem rozpoznatelné skupiny dat nebo informací obsažené v mezích aplikace. Pokud jsou logické skupiny i informace pro uživatele viditelné, pak je započítáme jako ILF. Má logickou strukturu a je uložen v souboru tak, jak ho vidí koncový uživatel.

Všechny entity interních logických souborů uložené v aplikaci StoreQ jsem analyzoval a znázornil jsem je v tabulkách (Tab. 17) s příslušnou komplexitou funkcí (Tab. 18). [19]

- Entita: Objednávka – Obsahem entity je následujících jedenáct datových prvků: Identifikační číslo objednávky, Jméno, Příjmení, Ulice, Objednávka, Město, PSČ, E-mail, Telefon, Cena a Cena s DPH. Celou entitu lze rozdělit do dvou podskupin: Osobní údaje a informace o objednávce. Z toho důvodu RET nabývá hodnoty dva
- Entita: Položka – Obsahem entity jsou tyto datové prvky: Identifikační číslo produktu, Název, Počet kusů, Popis a Cena. Všechny tyto prvky jsem zařadil do jedné skupiny RET

Tab. 17. Interní logické soubory pro aplikaci StoreQ

Operace	Počet DET	Počet RET
Objednávka	11	2
Položka	4	1

Tab. 18. Komplexita funkce ILF a EIF

RET	DET		
	1 - 19	20 - 50	Více než 50
1	Nízký	Nízký	Průměrný
2 až 5	Nízký	Průměrný	Vysoký
Více než 5	Průměrný	Vysoký	Vysoký

Soubory externího rozhraní

Funkce zahrnuje všechny logické skupiny uživatelských dat řízených jinou aplikací. Tato potřebná a používaná data, jenž se vyskytují za hranicí aplikace, jsou používána pouze podle aktuálních potřeb.

Všechny entity souborů externího rozhraní uložené v aplikaci StoreQ jsem analyzoval a znázornil jsem je v tabulkách (Tab. 19) s příslušnou komplexitou funkcí (Tab. 18). [19]

- Entita: DPH – Bude obsahovat tři procentuální hodnoty daní: 21 %, 15 % a 0 %.
- Entita: Sklad – Každý sklad bude ukazovat Název, Počet kusů, Popis a Cenu.

Tab. 19. Soubory externího rozhraní pro aplikaci StoreQ

Operace	Počet DET	Počet RET
DPH	3	1
Sklad Zlín	4	1
Sklad Otrokovice	4	1
Sklad Vizovice	4	1

Celkový počet neupravených funkčních bodů

K dosažení počtu neupravených funkčních bodů jsem dosadil počet transakčních a datových funkcí do vzorce (3.11). Příslušné váhové multiplikátory pro funkční typy jsou k nahlédnutí v tabulce (Tab. 11). Následný výpočet jsem provedl pomocí programu Excel. Celkový počet neupravených funkčních bodů jsem vypsál v tabulce (Tab. 20).

Tab. 20. Neupravené funkční body pro StoreQ

Komponent	Počet UFP
EI	21
EO	12
EQ	9
ILF	14
EIF	20
Celkem UFP	76

5.2 Výpočet upravených funkčních bodů

Dalším krokem k získání celkového počtu funkčních bodů je kalkulace a následné sečtení adjustačních faktorů. Každému z těchto faktorů jsem přiřadil hodnotu podle stupně vlivu na systém. Součtem všech nominálních hodnot jsem vynásobil součet neupravených funkčních bodů z předešlé kapitoly.

Pro aplikaci StoreQ je zde čtrnáct systémových charakteristik, jenž nabývají hodnoty 0 až 5 podle stupně vlivu:

1. Datová komunikace (*Data Communications*) – 3

Veškerá data a informace použítá v aplikaci pro odeslání nebo přijetí jsou přes komunikační prostředky. Přenos dat nebo výměna informací mezi dvěma systémy nebo zařízeními probíhá za přítomnosti protokolů obsahující konvence nebo pravidla pro přenos. Stupeň hodnocení pro aplikaci StoreQ znázorňuje tabulka (Tab. 21).

Tab. 21. Datová komunikace

Hodnocení (DI)	Popis
0	Aplikace používá dávkové zpracování nebo je stand-alone.
1	Aplikace je dávková, ale používá vzdálená vstupní data nebo vzdálený výpis.
2	Aplikace je dávková, ale používá vzdálená vstupní data a vzdálený výpis.
3	Aplikace zahrnuje on-line sběr dat nebo TP front-end k dávkovému postupu anebo dotazovacímu systému.
4	Aplikace je více než front-end, ale podporuje jen jeden typ TP komunikačních protokolů.
5	Aplikace je více než front-end a podporuje více než jeden typ TP komunikačních protokolů.

2. Distribuované zpracování dat (*Distributed Data Processing*) – 3

Distribuovaná data nebo vykonávání funkcí jsou vlastnosti uvnitř softwarových hranic.

U aplikace StoreQ probíhá příjem dat nikoliv odesílání, proto je směr přenosu jednosměrný a je on-line. Tabulka (Tab. 22) určuje stupeň charakteristiky.

Tab. 22. Distribuované zpracování dat

Hodnocení (DI)	Popis
0	Aplikace nepodporuje přenos dat nebo funkce mezi systémovými komponenty.
1	Aplikace připravuje data pro zpracování s použitím jiných systémových komponentů (např. tabulkové editory).
2	Data jsou připravena pro přenos, ale přesun a zpracování je součástí jiného systémového komponentu.
3	Distribuované zpracování je jednosměrné a přenos dat je on-line.
4	Distribuované zpracování je obousměrné a přenos dat je on-line.

5	Procesní funkce jsou vykonané dynamicky na nejvhodnějších systémových komponentech.
---	---

3. Výkon (*Performance*) – 1

Uživatelé stanovené a schválené požadavky týkající se celkové výkonnosti aplikace v závislosti na její používání. V mém případě požadavky při návrhu aplikace StoreQ nevyžadovaly žádnou zvýšenou pozornost. Výkon jsem ohodnotil pomocí tabulky (Tab. 23).

Tab. 23. Výkon

Hodnocení (DI)	Popis
0	Uživatelé nebyly stanovené žádné speciální požadavky.
1	Požadavky na výkon a design byly uživatelem podány, vzaty v úvahu, ale nevyžadovaly žádnou zvláštní námahu.
2	Doba odezvy a propustnost jsou rozhodující v čase špičky. Nebyly žádné zvláštní požadavky na design pro využití CPU. Deadline zhotovení se odkládá o jeden den.
3	Doba odezvy a propustnost je rozhodující po celý den. Nebyly žádné zvláštní požadavky na design pro využití CPU. Deadline zhotovení je omezený.
4	Požadavky uživatele na výkon jsou přesně stanovené a vyžadují analýzu výkonnosti ve fázi návrhu.
5	Při návrhu nebo během vývoje byly použity nástroje pro analýzu výkonnosti ke splnění uživatelských požadavků na výkon.

4. Plné využití konfigurací (*Heavily Used Configuration*) – 0

Při používání HW je nutné vzít v úvahu např. velikost uložště nebo výkonnost procesoru apod.. Z úvodních požadavků uživatele bylo zřejmé, že nepředpokládá žádné zvláštní výhrady pro konfigurační komponenty. Podrobné popsání charakteristiky znázorňuje tabulka (Tab. 24).

Tab. 24. Plné využití konfigurací

Hodnocení (DI)	Popis
0	Nejsou zahrnuta žádná explicitní ani implicitní provozní omezení.
1	Existují jistá provozní omezení, ale nejsou omezující. K eliminaci těchto omezení je zapotřebí jen malého úsilí.

2	Jsou zohledněná existující omezení na bezpečnost a načasování.
3	Jsou zohledněné specifické požadavky na procesor k určité části aplikace.
4	Zohlednění zvláštních omezení aplikace vede k úpravám v centrálním nebo dedikovaném procesoru.
5	Speciální omezení na aplikaci, jenž jsou součástí distribuovaných systémových komponentů.

5. Rychlost transakcí (*Transaction Rate*) – 3

Pro každého uživatele je velmi důležitá rychlost přenosu dat tak, aby se k uživateli dostala v co nejmenším možném čase. Rychlost je závislá na architektuře systému, ale ovlivňuje ji i design, instalace a podpora aplikace. Tabulka (Tab. 25) představuje úroveň charakteristiky.

Tab. 25. Rychlost transakcí

Hodnocení (DI)	Popis
0	Nepředpokládá se žádná špička při používání aplikace.
1	Předpokládá se periodické opakování špičky (např. měsíční, čtvrtletní, sezónní).
2	Předpokládá se periodická týdenní špička.
3	Předpokládá se periodická denní špička.
4	Uživatelské požadavky na rychlost transakce jsou vysoké a vyžadují analýzu výkonnosti v navrhovací fázi.
5	Uživatelské požadavky na rychlost transakce jsou vysoké a vyžadují analýzu výkonnosti v navrhovací fázi. Vyžadují také použití nástrojů pro analýzu výkonnosti během návrhu, vývoje a instalace.

6. Vstup dat on-line (*Online Data Entry*) – 3

Procenta znázorňují procentuální část dat, která přicházejí on-line přes hranice aplikace směrem dovnitř. Tato data jsou součástí většiny obrazovek softwarových aplikací. Aplikace StoreQ počítá přibližně s 20 %, jenž jsou transakcí interaktivních datových vstupů podle tabulky (Tab. 26).

Tab. 26. Vstup dat online

Hodnocení (DI)	Popis
0	Všechny transakce jsou zpracovány dávkově.
1	1 % až 7 % transakcí jsou interaktivní datové vstupy

2	8 % až 15 % transakcí jsou interaktivní datové vstupy
3	16 % až 23 % transakcí jsou interaktivní datové vstupy
4	24 % až 30 % transakcí jsou interaktivní datové vstupy
5	Více než 30 % transakcí jsou interaktivní datové vstupy

7. Účinnost koncového uživatele (*End-User Efficiency*) – 3

Charakteristika popisuje stupeň splnění aktivit koncového uživatele znázorněné v tabulce (Tab. 27). Při vývoji aplikace se její tvůrci zaměřují na tvorbu on-line designových funkcí důležitých pro spokojenost koncového uživatele. Designovými funkcemi je myšleno následujících 16 aktivit: navigační pomoc (funkční klávesy, dynamicky vytvořená menu apod.), menu, on-line pomoc a její dokumentace, automatický pohyb kurzoru, scrollování, vzdálený tisk, předurčené funkční klávesy, dávkovost online transakcí, zobrazená data vybraná kurzorem, vizuální úpravy textu, dokumentace on-line transakcí, rozhraní myši, vyskakovací (pop-up) okna, minimum obrazovek k dosažení obchodních funkcí, podpora ve dvou jazycích, podpora ve více než dvou jazycích.

Tab. 27. Účinnost koncového uživatele

Hodnocení (DI)	Popis
0	Ani jedna z 16ti aktivit.
1	Jedna až tři aktivity.
2	Čtyři až pět aktivit.
3	Šest a více aktivit bez uvedení zvláštních požadavků ovlivňujících účinnost.
4	Šest a více aktivit doprovázených vysokými požadavky, které jsou dostatečně silné natolik, že vyžadují použití jiných faktorů (např. snížení počtu úderů do klávesnice, využití šablon nebo maximalizace výchozího nastavení).
5	Šest a více aktivit doprovázené vysokými požadavky, jenž vyžadují použití nástrojů a procesů k prokázání, že byly stanovené cíle splněny.

8. Aktualizace on-line (*Online Update*) – 4

Jedná se především o aktualizaci vnitřních logických souborů aplikace, kde stupeň aktualizací je zobrazen v tabulce (Tab. 28). Rozdíl v jednotlivých stupních hodnocení

je v objemu aktualizací a jak velkou část systému aktualizace zahrnují. Většina aplikací vyžaduje průběžnou aktualizaci, což ve výsledku znamená konstantní vytíženost zdrojových dat.

Tab. 28. Aktualizace online

Hodnocení (DI)	Popis
0	Žádné.
1	Zahrnuta aktualizace jednoho až tří online souborů. Objem aktualizací je nízký a obnovení jednoduché.
2	Zahrnuta aktualizace čtyř a více online souborů. Objem aktualizací je nízký a obnovení jednoduché.
3	Zahrnuta aktualizace vnitřních logických souborů.
4	Součástí systému je speciálně navržená a implementovaná ochrana proti ztrátě dat.
5	Součástí systému jsou automatizované postupy a objem aktualizací je vysoký.

9. Celkové zpracování (*Complex Processing*) – 1

Komplexní zpracování je charakteristika popisující stupeň zpracování komponentů podle tabulky (Tab. 29). K dispozici jsou následující komponenty: kontrola citlivých informací nebo bezpečné zpracování dat, rozsáhlé logické zpracování, rozsáhlé matematické zpracování, opětovné zpracování neúplných transakcí, komplexní zpracování vstupních a výstupních operací.

Tab. 29. Celkové zpracování

Hodnocení (DI)	Popis
0	Žádný z komponentů.
1	Jeden libovolný komponent.
2	Libovolné dva komponenty.
3	Libovolné tři komponenty.
4	Libovolné čtyři komponenty.
5	Libovolných pět komponentů.

10. Znovupoužitelnost (*Reusability*) – 1

Charakteristika podle tabulky (Tab. 30) popisuje, do jaké míry lze opětovně použít kód při vývoji jiných aplikací.

Tab. 30. Znovupoužitelnost

Hodnocení (DI)	Popis
0	Žádný znovupoužitelný kód.
1	Znovupoužitelný kód pouze v rámci aplikace.
2	Méně než 10 % se používá na několik potřeb uživatele.
3	10 % a více se používá na několik potřeb uživatele.
4	Aplikace byla dokumentována pro budoucí znovupoužití. Aplikace je přizpůsobena uživatelem na úrovni zdrojového kódu.
5	Aplikace byla dokumentována pro budoucí znovupoužití. Aplikace je přizpůsobena k modifikaci parametrů uživatelem.

11. Snadnost instalace (*Installation Ease*) – 1

Vlastnost je zaměřena na jednoduchost instalace a konverzi, jenž byly poskytnuty a testovány během testovací fáze aplikace. Tento atribut podrobně popisuje tabulka (Tab. 31).

Tab. 31. Snadnost instalace

Hodnocení (DI)	Popis
0	Uživatel nestanovil žádné zvláštní požadavky.
1	Uživatel nestanovil žádné zvláštní požadavky, ale vyžaduje speciální nastavení instalace.
2	Konverzní a instalační požadavky byly stanoveny uživatelem. Dopad požadavků se nepovažuje za důležitý.
3	Konverzní a instalační požadavky byly stanoveny uživatelem. Dopad požadavků se považuje za důležitý.
4	Stejně jako položka 2 navíc s automatizovanými nástroji na instalaci
5	Stejně jako položka 3 navíc s automatizovanými nástroji na instalaci

12. Jednoduchost provozu (*Operational Ease*) – 2

Při tomto atributu záleží na množství manuálních zásahů, které by měly být po instalaci a konfiguraci aplikace co nejmenší. Spouštění, zálohování a obnovovací postupy byly vystaveny kontrole během testovací fáze vývoje. Důkladnější kontrola těchto postupů vede k minimalizaci míry jednoduchosti, kterou popisuje tabulka (Tab. 32).

Tab. 32. Jednoduchost provozu

Hodnocení (DI)	Popis
0	Uživatel nestanovil žádné zvláštní požadavky a souhlasil s běžným způsobem zálohování.
1 - 4	Na aplikaci platí některé z níže uvedených položek. Každá odrážka má hodnotu jednoho bodu, pokud tomu není jinak. <ul style="list-style-type: none"> • Je poskytnuto efektivní spuštění, obnova a záloha, ale je nutný zásah operátora. • Je poskytnuto efektivní spuštění, obnova a záloha, ale není nutný zásah operátora. • Aplikace minimalizuje potřebu použití pásek. • Aplikace minimalizuje potřebu použití papír.
5	Aplikace je určena pro provoz bez dozoru. Operace bez dozoru nezahrnují zapnutí a vypnutí systému. Automatické zotavení je součástí aplikace

13. Orientovaná na více míst (*Multiple Sites*) – 2

Existují aplikace kompatibilní s určitým HW nebo fungující pouze za přítomnosti určitého SW. Atribut popsáný v tabulce (Tab. 33) se soustředí na takové aplikace, jenž byly speciálně navrženy a vyvinuty tak, aby podporovaly různá HW nebo SW prostředí.

Tab. 33. Orientovaná na více míst

Hodnocení (DI)	Popis
0	Požadavky nevyžadují potřebu více než jedné uživatelské oblasti.
1	Požadavky více oblastí byly zohledněny v návrhu. Aplikace funguje jen za stejných HW a SW podmínek.
2	Požadavky více oblastí byly zohledněny v návrhu. Aplikace funguje jen za podobných HW a SW podmínek.

3	Požadavky více oblastí byly zohledněny v návrhu. Aplikace funguje jen za odlišných HW a SW podmínek.
4	Dokumentace a podporovaný plán jsou testovány tak, aby podporovaly aplikaci definovanou podle 2. úrovně ve více oblastech.
5	Dokumentace a podporovaný plán jsou testovány tak, aby podporovaly aplikaci definovanou podle 3. úrovně ve více oblastech.

14. Jednoduchost změn (*Facilitate Change*) – 4

Atribut popisuje míru udržování a správu aplikace, která je definována pěti různými charakteristikami. Každá z těchto charakteristik má hodnotu jedna (pokud není stanoveno jiné pravidlo) a jsou rozděleny do dvou skupin. Ke stanovení hodnoty podle počtu charakteristik obsažených v aplikaci slouží tabulka (Tab. 34).

Flexibilní dotazy:

- Součástí aplikace je příslušenství jednoduchého dotazování a zpráv (počítá se jako jedna položka).
- Součástí aplikace je příslušenství složitějšího dotazování a zpráv (počítá se jako dvě položky).
- Součástí aplikace je příslušenství komplexního dotazování a zpráv (počítá se jako tři položky).

Kontrolní data:

- Kontrolní data jsou uchovávána v tabulkách, které jsou udržované uživatelem pomocí interaktivních procesů. Veškeré změny proběhnou následující den.
- Kontrolní data jsou uchovávána v tabulkách, které jsou udržované uživatelem pomocí interaktivních procesů. Veškeré změny proběhnou okamžitě. (počítá se jako dvě položky).

Tab. 34. Jednoduchost změn

Hodnocení (DI)	Popis
0	Žádná z charakteristik.
1	Jedna položka z charakteristik.
2	Dvě položky z charakteristik.
3	Tři položky z charakteristik.
4	Čtyři položky z charakteristik.

5 | Pět položek z charakteristik.

K výpočtu adjustačního faktoru slouží vzorec (3.13), do kterého doplním celkový součet všech čtrnácti měr důležitosti.

$$VAF = (TDI \times 0.01) + 0.65 = (31 \times 0.01) + 0.65 = 0.96$$

Nyní použiji výslednou hodnotu adjustačního faktoru pro výpočet počtu upravených funkčních bodů, který je dán vzorcem (3.12).

$$AFP = UFP \times VAF = 76 \times 0.96 = 72.96$$

5.3 Zjištění celkových nákladů

Pro zjištění celkových nákladů použiji platné vztahy z kapitoly (3.3.3). Nejdříve si spočítám počet zdrojových řádků kódu pomocí vzorce (3.15):

$$SLOC = AFP * \frac{SLOC}{UFP} = 72.96 * 128 \cong 9339$$

V rovnici jsem dosadil za faktor udávající počet převedených funkčních bodů na počty řádku hodnotu 128, která podle tabulky (Tab. 5.) zastupuje programovací jazyk C.

Nyní spočítám hodnotu vynaloženého úsilí. Použiji dva vzorce (3.16) a (3.18) pro přesnější výsledek. Podle [12] je vztah definovaný vzorcem (3.17) vhodný v případě nedostatečného množství získaných údajů, a proto jej ve svém příkladu nebudu aplikovat.

$$E = c \times AFP + b = 2.5 \times 72.96 + 1.05 = 183.45$$

$$E = c \times AFP^b = 2.5 \times (72.96)^{1.05} \cong 226.03$$

Výpočtem průměru dvou získaných hodnot zpřesním svůj výpočet:

$$E = \frac{183.45 + 226.03}{2} = 204.74$$

Následujícím výpočtem stanovím přibližnou dobu potřebnou k realizaci produktu podle vztahu (3.19), výsledek je v kalendářních měsících:

$$TDEV = 0.38 \times E^{0.37} = 0.38 \times 204.74^{0.37} \cong 3$$

ZÁVĚR

Při realizaci rozsáhlých softwarových produktů se nevyhneme důkladné analýze a odhadu nezbytných vývojových nákladů. Odhadovací metody se dnes již dobře adaptují na různé druhy projektů a lze je uplatnit v jakékoli fázi životního cyklu projektu.

Zkoumané metody se liší v úvodních vstupech, kde je zapotřebí si ujasnit, kolik řádků zdrojového kódu bude během vývoje napsáno. Tak je tomu u metody COCOMO II. Oproti tomu metoda funkčních bodů vyžaduje znalosti týkající se funkcionality produktu. Mezi těmito metodami je úzká komunikace, jenž pramení z možnosti převést řádky zdrojových kódů na funkční body a naopak.

Cílem mé práce bylo charakterizovat algoritmické metody a zároveň jsem věnoval zvláštní pozornost funkčním bodům. V praktické části jsem si zvolil jednoduchý příklad, na kterém jsem demonstroval všechny znalosti získané z teoretické části mé práce. Zjistil jsem, že existují neupravené a upravené funkční body potřebné při výpočtu celkových nákladů. Součástí výpočtu neupravených funkčních bodů je i návrh a zpracování případu užití formou Use Case diagramu. Následujícím krokem bylo všechny tyto požadavky převést do externích vstupů, výstupů a dotazů dále interních logických souborů a souborů externího rozhraní. Všechny tyto komponenty jsem ve své práci ohodnotil a výslednou hodnotu aplikoval při výpočtu funkčních bodů. Nezbytnou součástí mého výpočtu bylo i zjištění adjustačního faktoru, jenž vycházel z kalkulace upravených funkčních bodů. Posledním z cílů bylo provést výpočet ve vhodném tabulkovém editoru. Ve své práci jsem si ke splnění tohoto úkolu vybral prostředí aplikace Excel, kde jsem provedl výpočty a vypsál veškerá data potřebná pro výpočet.

SEZNAM POUŽITÉ LITERATURY

- [1] SVOZILOVÁ, Alena. *Projektový management*. 1. vyd. Praha: Grada, 2006, 353 s. Expert (Grada). ISBN 80-247-1501-5.
- [2] BRUCKNER, Tomáš. *Tvorba informačních systémů: principy, metodiky, architektury*. 1. vyd. Praha: Grada, 2012, 357 s. Management v informační společnosti. ISBN 978-80-247-4153-6.
- [3] CHATFIELD, Carl S a Timothy D JOHNSON. *Microsoft Project 2000*. Redmond, Wash.: Microsoft Press, c2000, xix, 424 p. Step by step (Redmond, Wash.). ISBN 0735609209.
- [4] SCHWALBE, Kathy. *Řízení projektů v IT: kompletní průvodce*. Vyd. 1. Brno: Computer Press, 2011, 632 s. ISBN 978-80-251-2882-4.
- [5] *A guide to the project management body of knowledge: PMBOK guide*. 3rd ed. Newtown square, Pa.: Project Management Institute, Inc., c2004, viii, 390 s. ISBN 1930699506-.
- [6] POLÁK, Jiří, Antonín CARDA a Vojtěch MERUNKA. *Umění systémového návrhu: objektově orientovaná tvorba informačních systémů pomocí původní metody BORM*. 1. vyd. Praha: Grada, 2003, 195 s. Management v informační společnosti. ISBN 80-247-0424-2.
- [7] *Softwarové inženýrství – agilní metodiky-2* [online]. [cit. 2015-05-01]. Dostupný z WWW: <<http://moodle.vscht.cz/mod/resource/view.php?id=1813&redirect=1>>.
- [8] MCCONNELL, Steve. *Odhadování softwarových projektů: jak správně určit rozpočet, termín a zdroje*. Vyd. 1. Brno: Computer Press, 2006, 317 s. ISBN 80-251-1240-3.
- [9] Boehm, B. W., et al.: *COCOMO II Model Definition Manual*. USA, Los Angeles, University of Southern California 1999.
- [10] BOEHM, Barry W. *Software engineering economics*. Englewood Cliffs, N.J.: Prentice-Hall, c1981, xxvii, 767 p. ISBN 0138221227.
- [11] HAVALDAR, Padmaja. *Cost Estimation* [online]. 2003 [cit. 2015-05-11]. Dostupné z: <http://people.cis.ksu.edu/~padmaja/Project/CostEstimate.htm>
- [12] KRÁL, Jaroslav. *Informační systémy: specifikace : realizace : provoz*. 1. vyd. Veletiny: Science, c1998, 356 s. ISBN 80-86083-00-4

- [13] YAHYA, Majed Al, Rodina AHMED a Sai LEE. *Impact of CMMI Based Software Process Maturity on COCOMO II's Effort Estimation* [online]. 2010, 138 s. [cit. 2015-05-05].
- [14] I. Sommerville, *Software Engineering*. Upper Saddle River, NJ, USA: Addison-Wesley, 2011.
- [15] JAWADEKAR, Waman S. *Software engineering: principles and practice*. New Delhi: Tata McGraw-Hill, 2004. ISBN 9780070583719.
- [16] Function Point Modeler. *Function Point Modeler* [online]. 2009 [cit. 2015-05-11]. Dostupné z: <http://www.functionpointmodeler.com/>
- [17] GROUP, International Function Point Users a Mary Bradley] [CHAIRPERSON. *Function point counting practices manual*. Release 4.1. Westerville, OH: IFPUG, 1999. ISBN 0963174274.
- [18] Computer Science and Electrical Engineering. *Computer Science and Electrical Engineering* [online]. 2015 [cit. 2015-05-11]. Dostupné z: <http://www.csee.umbc.edu/>
- [19] ALVIN, Alexander. *How to Determine Your Application Size Using Function Points* [online]. 2004 [cit. 2015-05-19]. Dostupné z: <http://conferences.embarcadero.com/article/32094>
- [20] D. Garmus and D. Herron, *Function Point Analysis*. Upper Saddle River, NJ, USA: Addison-Wesley, 2000.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PMI Project management institute

3GL Third-generation programming language

SEZNAM OBRÁZKŮ

<i>Obr. 1. Projektový trojúhelník</i>	<i>11</i>
<i>Obr. 2. Porovnání tradičního a agilního přístupu</i>	<i>13</i>
<i>Obr. 3. Kužel přesnosti odhadu</i>	<i>15</i>
<i>Obr. 4. Use Case model aplikace StoreQ</i>	<i>35</i>

SEZNAM TABULEK

<i>Tab. 1. Složitost objektových bodů u ACM modelu</i>	<i>18</i>
<i>Tab. 2. Vyjádření námahy pro objekty modelu ACM</i>	<i>18</i>
<i>Tab. 3. Míra produktivity pro model ACM</i>	<i>19</i>
<i>Tab. 4. Kombinace EDM faktorů s PAM modelem.....</i>	<i>19</i>
<i>Tab. 5. Převod funkčních bodů do řádků kódu</i>	<i>20</i>
<i>Tab. 6. Nominální hodnoty faktorů pro EDM model</i>	<i>20</i>
<i>Tab. 7. Nominální hodnoty faktorů pro PAM model</i>	<i>22</i>
<i>Tab. 8. Zařazení jednotlivých oblastí do úrovně vyspělosti</i>	<i>23</i>
<i>Tab. 9. Hodnoty multiplikátorů pro model PAM</i>	<i>23</i>
<i>Tab. 10. Míra komprese/expanze SCED faktoru</i>	<i>25</i>
<i>Tab. 11. Váhové multiplikátory pro jednotlivé funkční typy</i>	<i>27</i>
<i>Tab. 12. Externí vstupy pro aplikaci StoreQ</i>	<i>38</i>
<i>Tab. 13. Komplexita funkce EI.....</i>	<i>39</i>
<i>Tab. 14. Externí výstupy pro aplikaci StoreQ.....</i>	<i>39</i>
<i>Tab. 15. Komplexita funkcí EO a EQ</i>	<i>40</i>
<i>Tab. 16. Externí dotazy pro aplikaci StoreQ</i>	<i>40</i>
<i>Tab. 17. Interní logické soubory pro aplikaci StoreQ</i>	<i>41</i>
<i>Tab. 18. Komplexita funkce ILF a EIF</i>	<i>41</i>
<i>Tab. 19. Soubory externího rozhraní pro aplikaci StoreQ</i>	<i>42</i>
<i>Tab. 20. Neupravené funkční body pro StoreQ</i>	<i>42</i>
<i>Tab. 21. Datová komunikace</i>	<i>43</i>
<i>Tab. 22. Distribuované zpracování dat</i>	<i>43</i>
<i>Tab. 23. Výkon</i>	<i>44</i>
<i>Tab. 24. Plné využití konfigurací</i>	<i>44</i>
<i>Tab. 25. Rychlost transakcí.....</i>	<i>45</i>
<i>Tab. 26. Vstup dat online</i>	<i>45</i>
<i>Tab. 27. Účinnost koncového uživatele</i>	<i>46</i>
<i>Tab. 28. Aktualizace online.....</i>	<i>47</i>
<i>Tab. 29. Celkové zpracování.....</i>	<i>47</i>
<i>Tab. 30. Znovupoužitelnost</i>	<i>48</i>
<i>Tab. 31. Snadnost instalace</i>	<i>48</i>
<i>Tab. 32. Jednoduchost provozu</i>	<i>49</i>

<i>Tab. 33. Orientovaná na více míst</i>	<i>49</i>
<i>Tab. 34. Jednoduchost změn</i>	<i>50</i>

SEZNAM PŘÍLOH

Příloha P1 - StoreQ

PŘÍLOHA P I: STOREQ

Příloha je součástí CD odevzdaného společně s bakalářskou prací.