ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE
Provozně ekonomická fakulta
Katedra informačního inženýrství

Obor informační management

METODY ODHADU SLOŽITOSTI VÝVOJE MODERNÍHO SOFTWARU

Doktorská disertační práce

Doktorand: **Ing. Zdeněk Struska**
Školitel: Prof. RNDr. Jiří Vaníček, CSc.

Borovany 2008

Úvodem této práce bych chtěl poděkovat svému školiteli Prof. RNDr. Jiřímu Vaníčkovi, CSc. za směřování práce, dále za pomoc a připomínky v průběhu tvorby této práce. Mé díky patří i Doc. Vojtěchu Merunkovi, PhD. za odborné připomínky týkající se metody BORM.

Poděkovat bych chtěl také Katedře informačního inženýrství Provozně-ekonomické fakulty ČZU v Praze za zázemí poskytnuté v průběhu studia.

Největší díky však patří mým rodičům za podporu v průběhu studií.

Duben 2008

OBSAH

1 Úvod

Pokud chce softwarová firma uspět v současném světě se svými produkty, musí včas a přesně reagovat na potřeby trhu. Aby byla společnost na trhu úspěšná musí nabídnout produkt, který bude konkurenceschopný nejen kvalitou ale také cenou.

Pro odhad svých šancí na úspěch, musí znát náklady, za které je schopna své produkty vyrábět nebo nabízet. Tato teze platí pro všechny firmy obecně, tzn. i pro ty, které se zabývají tvorbou softwaru. Stejně jako si výrobní firma před začátkem výroby nového či reorganizace výroby existujícího produktu sestavuje, na kolik jí daná změna přijde a jak to ovlivní cenu budoucího výrobku, musí i firma vytvářející software provést předběžnou kalkulaci svých nákladů souvisejících s tvorbou či zdokonalením softwaru.

Pro tyto účely existují metody, které nákladovost softwaru určují zprostředkovaně prostřednictvím odhadu složitosti. Přínosné by bylo znát co nejpřesnější hodnoty, co nejdříve. To je ovšem v oblasti odhadu složitosti, který se provádí v počáteční fázi vývoje IS, kdy máme omezené vstupní informace, nemožné.

Pro lepší porozumění odborným výrazům a vhodnému postupu zavedení nové terminologie jsou anglicky psané názvy ve složených závorkách.

1.1 Cíle disertační práce

Předkládaná disertační práce je zaměřena na oblast odhadu složitosti informačních systémů (IS). Cílem práce není úplný popis všech existujících metod, které pro odhad složitosti vývoje IS existují, ale vyzdvižení především těch, které jsou dle autora této práce nejperspektivnější. Dále pak návrh nové metody pro odhad složitosti systémů vyvíjených metodou BORM, jednou z perspektivních metodik.

Oblast odhadu složitosti je jednou z podceňovaných součástí vývoje IS. Právě tento odhad lze použít při rozhodování o dalším osudu vyvíjeného softwaru. Je nutné si uvědomit, že tyto metody nabízí první orientační odhad v úvodní fázi vývoje softwaru na základě existujících vstupů. Zcela zřejmě nelze od metod očekávat exaktně přesné odhady, které se v průběhu

projektu nezmění. Je tomu naopak, protože odhad složitosti je možné v průběhu fází životního cyklu IS neustále zpřesňovat.

Cíle předkládané práce jsou

1. Zmapovat oblast odhadu složitosti vývoje IS.
2. Představit a popsat nejvýznamnější existující metody pro odhad složitosti.
3. Interpretovat a srovnat představené metody odhadu složitosti.
4. Na základě získaných poznatků vyvinout novou metodu odhadu složitosti IS navržených v metodě BORM.

Součástí disertační práce jsou dvě přílohy. První z nich rozšiřuje popis metody BORM z kapitoly 3.4.5 Úvod do metody BORM, druhá představuje podrobný popis metody Analytického hierarchického rozkladu použité pro vývoj koeficientů a vah nové metody v kapitole 4.1 Postup nastavení vah a koeficientů metody BORMp.

1.2 Metodika dosažení cílů práce

Metodika disertační práce je založena na studiu odborné literatury a aparátu kvantitativních metod a systémové analýzy.

Práce autora se sestávala z následujících kroků:

1. Studium odborné literatury.
2. Identifikace nejvýznamnějších skupin metod pro odhad pracnosti.
3. Ověření kvality výstupů vybraných metod odhadu pracnosti.
4. Identifikace prostoru pro autorovu další vědeckou činnost ve vybrané oblasti.
5. Návrh konceptu nové metody odhadu pracnosti.
6. Sběr dat pro praktické ověření navrhovaného výpočtu nové metody.
7. Kvantifikace klíčových proměnných metody a návrh koeficientů nové metody.
8. Ověření výsledků nové metody na datech z praktických projektů.

1.2.1 Přínosy disertační práce

Disertační práce obsahuje původní myšlenky a postupy představující autorův přínos do problematiky odhadu složitosti IS.

V kapitole 3.4 Odhady založené na modelu představuje a popsuje autor práce softwarové nástroje podporujících představené metody. Tyto nástroje umožňují provádět rychlé odhady složitosti navrhovaných systémů.

V kapitole 3.5 Zhodnocení technik odhadu složitosti vývoje provádí autor zhodnocení představených metod a naznačuje použitelnost popsaných metod na různé projekty tvorby IS.

V kapitole 4. zachycuje autor jednotlivé kroky vývoje nové metody, která je pojmenována BORM Points (BORMp), v kapitolách:

1. 4.1 Postup nastavení vah a koeficientů metody BORMp – autor zde popisuje nastavení koeficientů a vah pro vzorce používané v metodě BORMp pomocí kombinace expertní metody a metody Analytického hierarchického rozkladu.
2. 4.2 Návrh výpočtu složitosti metodou BORMp – autor představuje novou metodu určenou pro odhad složitosti IS navržených v metodě BORM.
3. 4.3 Testování metody BORMp – autor zde popisuje testování navržené metody na konkrétních projektech a zhodnocení získaných výsledků.

1.2.2 Terminologie

Předložená práce je vypracována v českém jazyce a snaží se udržet české odborné terminologie. Autor však narazil na určité nesrovnalosti a proto v zájmu uchování přesnosti vyjádření používá některé anglické termíny bez jejich překladu. Je to zejména případ termínů, které nemají podle názoru autora český ekvivalent a nebo je jejich český překlad významově totožný s jiným pojmem. Jde například o pojmy „Function Points“ a „Feature Points“, které mají odlišný význam, ale do českého jazyka se překládají stejně jako funkční body.

1.3 Vývoj a stav řešené problematiky

Pro klasický návrh IS jsou celkem podrobně popsány metody odhadu pracnosti, naproti tomu pro objektový návrh zatím nikde souhrnně zpracovány nejsou. Samozřejmě již existuje mnoho dílčích teoretických prací, které jednotlivě dokazují účelnost objektově orientovaného datového modelu v databázových systémech. Přesto se zatím v oblasti metod analýzy a návrhu objektových databázových aplikací vesměs používají jen postupy původně určené pro práci s relačními systémy a nebo jen intuitivní přístupy založené na zkušenosti s imperativními objektově orientovanými programovacími jazyky. Předpokládá se využití refaktoringu,

návrhových vzorů a agilních metodik. Bohužel pro objektový datový model zatím není žádná všeobecně uznávaná a používaná technika nebo metoda návrhu [Merunka 2004].

Na základě předpovídaného vývoje v oblasti návrhu databázových systému se dá očekávat, že objektově orientovaný datový model by mohl postupně nahrazovat model relační. Jednou z významných nevýhod objektového prostředí je chybějící standardizace, která negativně ovlivňuje větší rozšíření objektově orientovaného modelu v praxi.

Na nastalou situaci by měly včas reagovat obory, které přímo souvisejí s některou z fází návrhu IS pomocí objektově orientovaných metod. Pokud si představíme časovou posloupnost jednotlivých fází, pak prvním krokem je vždy vymezení požadavků. Právě na startovní čáře tvorby IS bývá velmi přínosné a důležité odhadnout budoucí cílový stav projektu. Ač se dá zcela důvodně pochybovat o přesnosti těchto odhadů, přesto existují metody, které v počátcích vývoje mohou tvůrcům softwaru částečně napomoci.

První metodou, která se pokusila odhadnout složitost IS, byla metoda IBM Function Points. Na ní úzce navazuje její současná platforma označovaná podle skupiny, která ji vytvořila, jako IFPUG (International Function Points Users Group). Nejnovější verzí metody IBM Function Points se stala metoda COSMICS-FPP, která byla poprvé představena v roce 1999. Poslední jmenovanou metodou vycházející z původní metody IBM Function Points ze sedmdesátých let minulého století je metoda označovaná jako SPR Function Points. Právě ta se stala základem pro expanzi metody Function Points do objektového prostředí. Výsledkem je metoda označovaná jako SPR Feature Points.

Metoda, která se pouze inspirovala způsobem výpočtu IBM Function Points, avšak razila nový směr v této vědecké oblasti je metoda Use Case Points (UCP). Vznikla na počátku devadesátých let 20. století, za autora je považován Gustav Karner [Karner 1993].

V práci je také představena metoda COCOMO, jedná se o v současnosti široce známou a uznávanou metodu pro odhad pracnosti. Její první verze, tzv. COCOMO 1.1 odhaduje pracnost na základě objemu programu. Důležitým vstupem metody COCOMO 1.1 v úvodních etapách odhadu složitosti je objem programu získaný jako výsledek metody Function Points. Naproti tomu nová verze tzv. COCOMO 2.0 nabízí uživateli sadu nástrojů umožňujících provést odhad pracnosti bez použití metody Function Points nebo jí podobné.

Aby práce věrohodným způsobem odrážela postupy, které jsou pravidelně doporučovány a využívány nejen v teoretické ale i praktické rovině, jsou zde popsány také metody odhadu složitosti na základě předcházejících zkušeností a tzv. expertní odhady. Jde patrně v současnosti převažující metody, které firmy v Česku a ve světě v praxi využívají.

Novou a pro tuto práci nejvýznamnější metodou, kterou autor této práce navrhuje, je metoda BORMp (BORMp), která vychází z obecných zásad odhadu složitosti vývoje IS, rozvíjí je však o speciální ideje a techniky pro systémy vyvíjené metodou BORM. Metoda BORMp úzce navazují na metodu UCP, zároveň se pokouší rozšířit vstupy metody o nový tzv. zákaznický faktor, který by měl zprostředkovat dopad zákazníků do projektu nově navrhovaného softwaru.

1.4 Motivace

V současné době schází v české vědě kvalitní literatura, která by shrnovala problematiku technik odhadu složitosti vývoje IS a podrobně se tímto tématem zabývala. Většina publikací v českém jazyce se pouze odkazuje na existenci některé v této práci uvedené metody, aniž by ji hlouběji rozváděla. Jedním ze světlých ostrůvku je práce prof. Vaníčka [Vaníček 2004], která je určena pro posluchače oboru Informatika na Provozně-ekonomické fakultě České zemědělské univerzity. Prof. Vaníček zde představuje metodu COCOMO a koncept metody Function Points, která se stala zdrojem inspirace pro tvůrce metod následujících.

Motivací autora příspěvku je předložit strukturovaný přehled nejvýznamnějších metod a technik používaných pro odhad složitosti vývoje IS. Dále naznačit hierarchii vývoje vybraných metod včetně podrobného postupu výpočtu u každé z nich.

Důležitou částí této práce jsou kapitoly popisující vývoj metody BORMp. Ta představuje krok do zcela neprobádané oblasti techniky odhadu složitosti a stává se první metodou odhadu složitosti pro metodiku BORM (Business and Object Relation Modelling). Autor této práce se domnívá, že na Katedře informačního inženýrství Provozně-ekonomické fakulty České zemědělské univerzity je ideální prostředí pro tuto činnost, protože jeden z autorů metody BORM je jejím členem, tzn. dá se říci, že metoda je zde „domácí“.

Do budoucnosti je velkou motivací autora této práce zaměřit se na zdokonalování v práci navržené metody BORMp, takovým způsobem aby nezanikla, ale stala se předmětem výzkumu dalších vědeckých pracovníků. Dále byla aktivně prezentována na odborných konferencích

a seminářích a dostala se do podvědomí odborné veřejnosti. Stejně tak aby se stala součástí softwarových nástrojů, které jsou určeny k modelování skutečnosti prostřednictvím metodiky BORM.

2 Metody tvorby IS

Neustálé zkracování vývojových, realizačních i produkčních cyklů je důsledkem prorůstání softwarových aplikací do všech složek lidských aktivit. Podle Cardy [Carda, Merunka, Polák 2003] jsou v oblasti tvorby softwaru zřetelné usilovné snahy po dokonalejším, spolehlivějším, efektivněji vytvořeném programovém vybavení a jeho zajištění v nebývalých rozměrech, vytvářeným s přiměřenými náklady. Existuje zde nepochybně paralela s výrobními odvětvími a nabízí se tu srovnání s průmyslovou revolucí z přelomu 18. a 19. století. Ta měla za důsledek přesun objemu výroby od drobných řemeslníků k prvním velkovýrobcům, což bylo doprovázeno novou technologií i organizací práce.

Proto je logické, že společně s rozvojem praktického používání OOP došlo k rozvoji metod analýzy a návrhu IS využívajících objektovou technologii, na které postupně navazovaly také metody odhadu složitosti určené pro objektové prostředí.

Zpočátku se používaly klasické metody založené na strukturovaném přístupu a role nástrojů OOP byla omezena jen do oblasti implementace. Tento přístup se však neosvědčil. Přibližně od konce 80. let bylo vyvinuto několik vzájemně odlišných metodologií pro objektově orientovanou analýzu a návrh. [Drbal 1996], [Wilkie 1993]. Dle Merunky [Merunka, Pergl, Picka 2005] mezi ně patří především:

1. OMT - Object Modelling Technique, jejímž autory jsou J. Rumbaugh, M. Bláha, W. Premerlani, F. Eddy a W. Lorensen. Metoda byla poprvé publikována nakladatelstvím Prentice Hall v knize "Object-Oriented Modelling and Design" v roce 1991. Technika je zajímavá tím, že je v podstatě hybridní technikou, zahrnující jak vybrané objektové, tak i klasické nástroje. Nejčastěji se používala pro návrh databázově orientovaných aplikací s objektovým klientem a relačním serverem. [Blaha, Premerlani 1995], [Rumbaugh et al. 1991].

2. Coad – Yourdonova metoda, jejímiž autory jsou P. Coad a E. Yourdon (autor velmi používané metody klasické strukturované analýzy a návrhu). Metoda byla poprvé

publikována v časopise American Programmer v roce 1989 a posléze v knihách obou autorů nakladatelství Yourdon Press. Z uvedených technik je nejsnazší co do počtu používaných pojmů. [Yourdon 1994].

3. OOSD - Object-Oriented Software Development, jejímž autorem je G. Booch. Metoda byla poprvé publikována v roce 1991 v knize nakladatelství Benjamin/ Cummings "Object-Oriented Design with Applications". Tato poměrně velmi komplexní metodologie byla určena pro týmový vývoj rozsáhlých aplikací v jazyce C++ a Smalltalk, a ze všech zde vyjmenovaných metod pokrývá nejvíce objektových vlastností. [Booch 1994].

4. OOSE - Object-Oriented Software Engineering autora Ivara Jacobsona. Metoda je velmi kvalitně popsána ve stejnojmenné knize. (Alternativní název metody je „Objectory“) Vzhledem k tomu, že metoda má původ ve skandinávské škole, je velmi zajímavá pro aplikace z oblasti simulace a řízení. Jacobsonova metoda se jako první začala zabývat problematikou získávání a modelování informací v prvních fázích životního cyklu ještě před budováním konceptuálního diagramu. Úvodní technika této metody - „Use Case“ byla adoptována i do ostatních objektových metodologií a je dodnes používána. [Jacobson 1992].

5. Object-Oriented Structured Design notation autorů A.I. Wassermanna, P.A. Pirchera a R.J. Mullera, publikovaná poprvé v časopise IEEE Computer č. 23(3) 1990. Metoda je zajímavá především notací používaných diagramů, která ovlivnila následné metodologie.

6. OOER notation autorů K. Gormana a J. Choobineha, poprvé publikovaná na 23. mezinárodní konferenci o informatice (computer science) na Havaji v létě 1991. Metoda používá notaci ER diagramů v klasické Chenově syntaxi, rozšířené o objektové vlastnosti. Je výhodná pro použití v objektově orientovaných databázích.

7. Unifikovaný modelovací jazyk UML podporují od roku 1996 autoři G. Booch, J. Rumbaugh a I. Jacobson pod záštitou firmy Rational Inc. Metoda začala být publikována průběžně na Internetu a v sérii knih, vydávaných firmou Rational Inc, která dodává také vlastní CASE nástroj Rational Rose. UML je dnes doporučovaným průmyslovým standardem pro notaci (způsob kreslení) diagramů a je stále ve vývoji.

Představuje sjednocení myšlenek původních metod svých autorů na platformě OMT. Ve srovnání s původní OMT je patrný posun směrem k větší míře podpory objektových vlastností a určitý odklon od původních „hybridních“ vlastností OMT - například zrušení datového modelování pomocí DFD. Je třeba mít na paměti, že UML je ve skutečnosti jen jazyk, tedy návrh standardu k zakreslování nejrůznějších objektových diagramů, který se navíc stále vyvíjí, mění a doplňuje. [UML 2004].

8. Metoda J. Martina a J. Odella. Metoda byla publikována v sérii knih nakladatelství Martin Books, a představuje jako UML pokus o sjednocení objektových zkušeností předchozích metod. Používá velké množství nejrůznějších diagramů a pojmů. [Martin, Odell 1995].

Dále Merunka [Merunka, Pergl, Picka 2005] zmiňuje, že kromě výše uvedených metod vzniklo ještě několik vesměs velmi kvalitních technik nebo i jen nástrojů, jejichž rozsah však byl omezen na několik publikací nebo na výuku softwarového inženýrství na vysokých školách, anebo se jedná o firemní know-how.

2.1.1 Transformace uživatelských požadavků

I když je tvorba objektového modelu usnadněna výše uvedenými metodami a nástroji, které říkají co a jak lze při modelování použít, jde o velmi obtížný a přitom klíčový problém. Zatím není k dispozici jednoznačný a všeobecně uznávaný návod, který by posloužil k efektivní transformaci zadání problému do formální podoby modelu. Velkou roli zde hraje zkušenost návrháře a jeho schopnost komunikovat se zadavateli.

Značným problémem je rozpoznání objektů a jejich vlastností při modelování zadání pro vytvářený systém, včetně jeho odhadu složitosti. K řešení těchto problémů byly vyvíjeny různé různě složité techniky. Tou nejjednodušší – až naivní, a také samozřejmě nejméně účinnou – je návod vycházející z jazykové analýzy textu zadání (podstatná jména = objekty, slovesa = metody, přídavná jména = atributy). [Rumbaugh et al. 1991]

Mezi sofistikovanější metody, které byly pro tento účel vyvinuty, patří podle Merunky [Merunka 2004]:

1. Object Behavioral Analysis, která slouží k získávání prvotní představy o objektovém modelu. Jde o iterativní techniku používající především tzv. modelující karty. Touto technikou začíná projekt tvorby IS a jejím výstupem je objektová informace dostatečná k sestrojení prvotního modelu. [Rubin, Goldberg 1992].

2. Behavioral Constraints [Goldberg 1995], která slouží k transformaci prvotního objektového modelu na konceptuální metodou „filtrace“ skrze formálně vyjádřená pravidla, která omezují použití jednotlivých možných hierarchií mezi objekty.

3. Applying Patterns - využívání vzorů (poprvé publikováno P. Coadem na Internetu, dále například [Beck 1997]). Technika slouží ke stanovení vhodné struktury objektů v konceptuálním modelu pomocí opětovného používání objektových vzorů. Ty jsou vlastně formalizované části znalostí z předchozích projektů, které jsou v podobě konceptuálních podgrafů a jsou schopny se vzájemně kombinovat a v nových podmínkách hrát nové role. Používání této techniky spočívá ve vytváření konceptuálního modelu jako orientovaného grafu metodou skládání a substitucí podgrafů představujících jednotlivé vzory.

4. Structural Transformations [Nierstrasz, Papathomas 1990, Shiver, Wegner 1987, Goldberg 1995], jenž je zvláštním případem předchozí techniky pro potřeby postupné transformace objektového modelu od modelu popisujícího zadání k modelu popisujícímu řešení.

2.2 Zajištění jakosti pro odhad složitosti IS

Jakost produktu je definována jako souhrn podstatných vlastností produktu, které určují míru uspokojení daných (obecně očekávaných) {impied} a stanovených {stated} potřeb {needs} uživatele produktu, v případě užití produktu stanoveným způsobem [ČSN ISO 8402].

Aby bylo podle Vaníčka [Vaníček 2004] dosaženo požadované jakosti, je nutné chápat jakost ne pouze z pohledu subjektu, který si IS objednává, ale také z pohledu všech ostatních zainteresovaných stran. I ty budou se systémem pracovat, proto je důležité jejich požadavky zohlednit.

Faktorem ovlivňujícím jakost vytvářeného IS je cena. Je důležité, aby byla nalezena rovnováha mezi cenou a jakostí. Pokud budeme chtít vytvářet vysoce jakostní produkt je pravděpodobné, že budeme muset zaplatit nemalou částku k jeho dosažení. V případě nízkých nároků na jakost to platí opačně.

Pokud bude požadováno, aby vyvíjený software byl na trhu úspěšný, je nutné do požadavků na IS zapracovat požadavky prodejců, jejichž cílem je maximalizovat zisk, i zákazníků, kteří vstupují na trh maximalizovat svůj užitek nákupem produktu s nejlepším poměrem mezi jakosti a cenou [Vaníček 2004].

Každý zákazník preferuje rozdílný způsob výběru produktu. Někteří se spoléhají na své schopnosti a „šťastnou ruku“, jiní dávají na zkušenosti a rady vybraného nezávislého subjektu. Ať již si zákazník vybere první nebo druhou možnost měl by věnovat pozornost tomu, zda vybraný produkt vyhovuje tuzemským či mezinárodním normám. Ty jsou z dílny mezinárodního technického výboru ISO/IEC JTC1 Informační technologie, který je společným technickým výborem obou normalizačních organizací – ISO – někdy chápané jako zkratka International Standard Organisation a International Electromechanical Commisson (IEC).

Je-li produkt v souladu s normou, splňuje požadavky dané normy. To samozřejmě neznamená, že splňuje požadavky uživatele. Normy na jakost softwarových produktů mají většinou tu povahu, že nutí dodavatele produktu, aby zveřejnil všechny relevantní informace nutné pro to, aby ten, kdo produkt poptává mohl vyhodnotit, zda úroveň dosažené jakosti splňuje jeho požadavky, vycházející z jeho potřeb [Vaníček 2004]. Normy pro jakost připouštějí většinou nabízet i levné produkty s nízkou úrovní jakosti. Tyto produkty se však nesmí tvářit jinak.

Vaníček [Vaníček 2004]chápe pod pojmem **norma** {standard} technickou normu. Ta je vymezena jako společně dohodnutý předpis pro technický nebo technicko-ekonomický stav nějaké entity nebo průběh nějakého jevu za daných podmínek [Vaníček 2004].

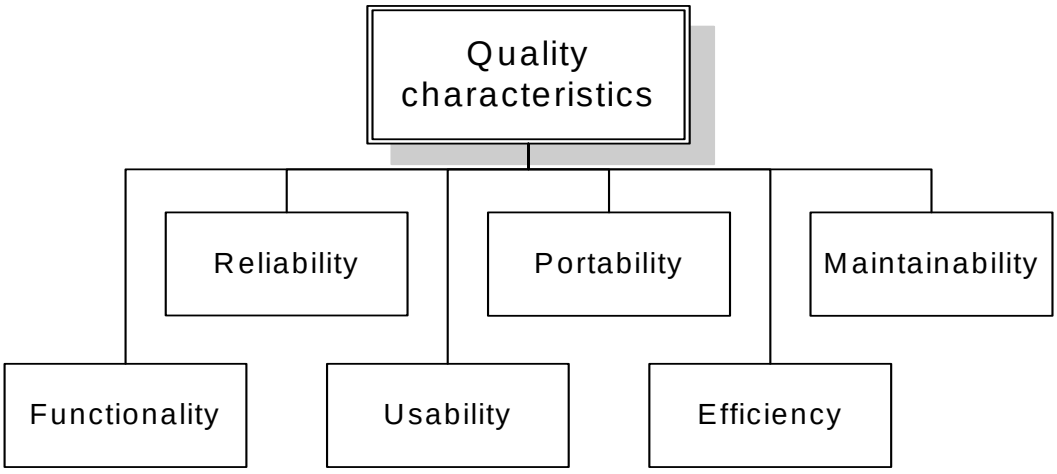
2.2.1 Charakteristiky jakosti použitelné pro odhadování složitosti IS

Před začátkem tvorby IS je vhodné se dobře seznámit s jednotlivými charakteristikami jakosti, které mohou významným způsobem ovlivnit budoucí spokojenost zákazníka s naším produktem a eliminovat neočekávané náklady na pozdější změnu či úpravu vyvíjeného produktu. Jejich definice vycházejí z mezinárodní normy ISO/IEC 9126-1 Informační technologie – Jakost

softwarového produktu – Model jakosti [ISO/IEC 9126], byla též přejata do soustavy evropských i českých norem jako ČSN ISO/IEC 9126-1.

Autoři Struska a Vaníček [Struska, Vaníček 2005a] považují za zřejmé, že každý zákazník může mít rozdílné požadavky na vyvíjený produkt, ty pramení z podnikových strategií, cílů, organizačních struktur a oblasti podnikání. Podstatný rozdíl v požadavcích bude mezi softwarem pro jadernou elektrárnu, kde bude velký důraz kladen na bezpečnost a bezporuchovos. Naproti tomu software pro terminál hromadné dopravy, kde si cestující bude vyhledávat informace týkající se pražské hromadné dopravy. U něj bude důraz kladen na použitelnost a příjemné uživatelské prostředí, ve kterém se uživatelé bez problémů zorientují.

Aby bylo možné stanovit úrovně jakosti odděleně podle skutečných potřeb, bylo dohodnuto rozdělit jakost softwarových produktů na šest kategorií nazvaných charakteristiky jakosti, které se rozpadají do dalších podcharakteristik.



Obr. 1. Charakteristiky jakosti [Struska, Vaníček 2005a]

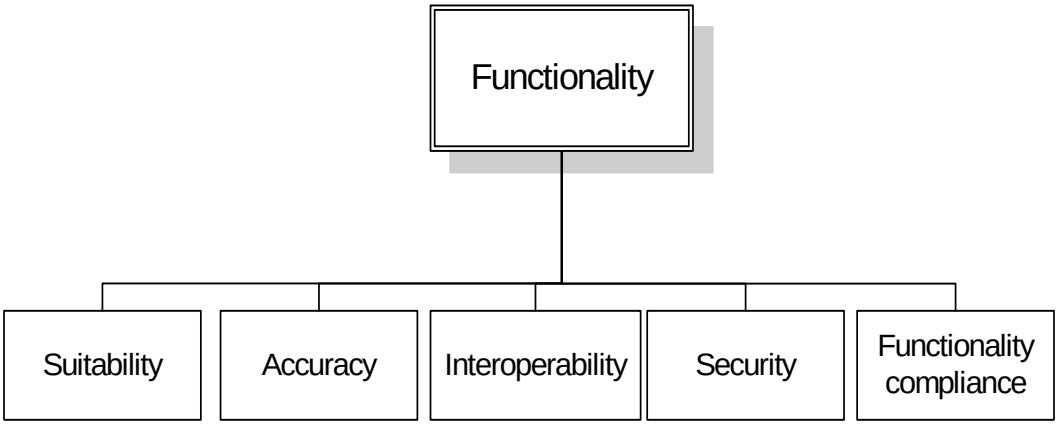
2.2.1.1 Funkčnost {Functionality}

Funkčnost je vymezena jako schopnost IS či softwarového produktu obsahovat funkce, které zabezpečují předpokládané nebo stanovené potřeby uživatele při používání systému za stanovených podmínek. Tato charakteristika tedy zjišťuje, zda jsou funkce vůbec zabezpečeny, nikoliv jak jsou zabezpečeny [Vaníček 2004].

Mezi podcharakteristiky funkčnosti patří [ISO/IEC 9126]:

- *Funkční přiměřenost* {suitability} - Funkční přiměřenost je vymezena jako schopnost poskytovat funkce pro zajištění specifikovaných úloh a cílů uživatele.

- *Přesnost* {accuracy} - Přesnost je vymezena jako schopnost poskytnout správné a požadované výsledky s potřebnou úrovní přesnosti.
- *Schopnost spolupráce* {interoperability} - Schopnost spolupráce je vymezena jako schopnost spolupracovat s jedním nebo několika jinými specifikovanými systémy.
- *Bezpečnost* {security} - Bezpečnost je vymezena jako schopnost chránit informace a data tak, aby neautorizovaná osoba nebo systém neměla možnost je číst či modifikovat a přitom autorizovaným subjektům nebyla odepřena stanovená úroveň přístupu k datům.
- *Shoda ve funkčnosti* {functionality compliance} - Shoda funkčnosti je vymezena jako schopnost pracovat ve shodě s normami, standardy, zákony, konvencemi a zvyklostmi obvyklými v prostředí, v kterém je systém či produkt využíván.



Obr. 2. Podcharakteristiky funkčnosti [Struska, Vaníček 2005a]

2.2.1.2 Bezporuchovost {Reliability}

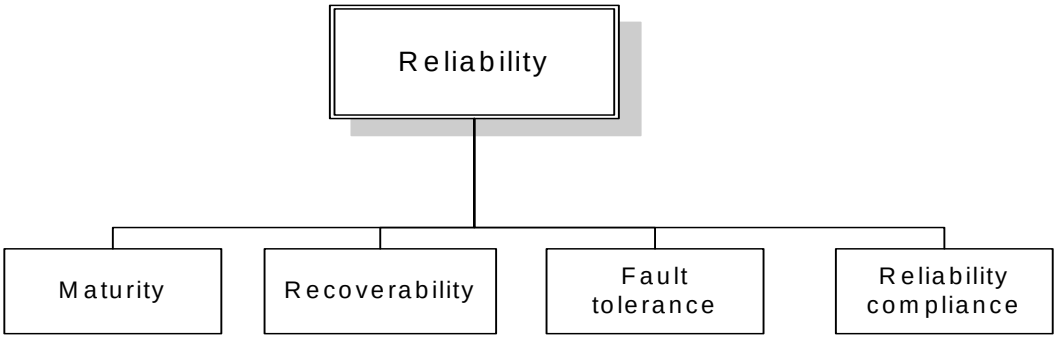
Bezporuchovost je vymezena jako schopnost IS či softwarového produktu zachovat specifikovanou úroveň výkonu při používání systému za stanovených podmínek [Vaníček 2004].

Mezi podcharakteristiky bezporuchovosti patří [ISO/IEC 9126]:

- *Zralost* {maturity} - Zralost je vymezena jako schopnost systému vyvarovat se poruchám (selháním) v důsledku závad systému nebo důsledky takovýchto selhání minimalizovat.
- *Odolnost vůči vadám* {fault tolerance} - Odolnost vůči vadám je vymezena jako schopnost systému zachovat si při selhání systému nebo při nedodržení

požadovaného rozhraní ze strany uživatele určitou úroveň výkonu (poskytovaných služeb).

- *Schopnost zotavení* {recoverability} - Schopnost zotavení je vymezena jako vlastnost systému obnovit úroveň výkonu a zachovat data po odstranění poruchy.
- *Shoda v bezporuchovosti* {reliability compliance} – podobný význam jako u funkčnosti.



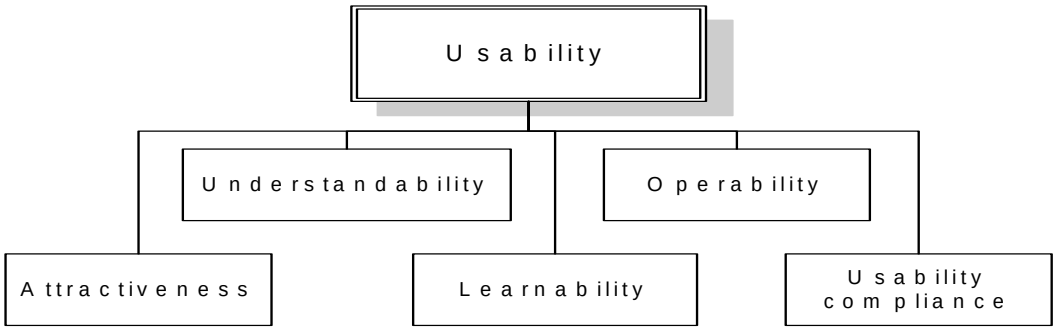
Obr. 3. Podcharakteristiky bezpečnosti [Struska, Vaníček 2005a]

2.2.1.3 Použitelnost {Usability}

Použitelnost je vymezena jako schopnost IS či softwarového produktu být srozumitelný, se snadno naučitelnou obsluhou, a atraktivní při používání za stanovených podmínek [Vaníček 2004].

Mezi podcharakteristiky použitelnosti patří [ISO/IEC 9126]:

- *Srozumitelnost* {understandability} - vlastnost systému, která umožňuje uživateli rozhodnout, zda se systém hodí pro řešení jeho problémů, jak je možné jej užít pro řešení jednotlivých úloh a za jakých podmínek.
- *Naučitelnost* {learnability} - je vymezena jako vlastnost systému, charakterizovaná mírou úsilí, které je třeba vynaložit pro rutinní využívání jeho možností.
- *Provozovatelnost* {operability} - je vymezena jako vlastnost systému usnadňující jeho obsluhu a řízení rutinní práce se systémem.
- *Atraktivnost* {attractiveness} - je vymezena jako schopnost systému umožnit příjemnou obsluhu a učinit užití systému přitažlivým.
- *Shoda v použitelnosti* {usability compliance} – podobný význam jako u funkčnosti.



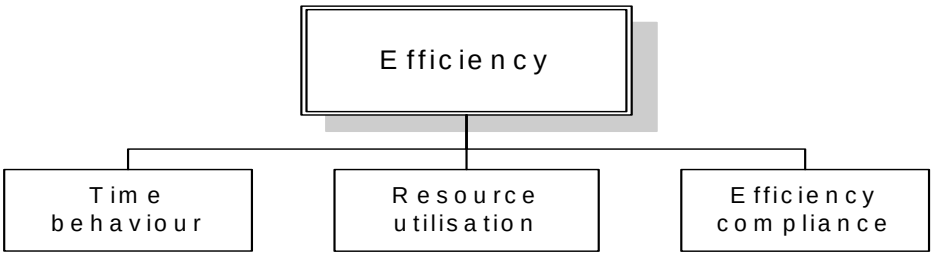
Obr. 4. Podcharakteristiky použitelnosti [Struska, Vaníček 2005a]

2.2.1.4 Účinnost {Efficiency}

Účinnost je vymezena jako schopnost IS či softwarového produktu poskytovat potřebný výkon vzhledem k množství použitých zdrojů při používání za stanovených podmínek [Vaníček 2004].

Mezi podcharakteristiky účinnosti patří [ISO/IEC 9126]:

- *Časové chování* {time behaviour} - je vymezeno jako schopnost systému zajistit požadovanou propustnost úloh za dané časové období, dobu výpočtu úlohy nebo odezvu systému při používání systému za daných podmínek.
- *Vymezení zdrojů* {resource utilization} - je vymezeno jako schopnost systému zajistit požadované funkce přiměřeným počtem typů a množstvím a rozsahem užitých zdrojů, které jsou potřeba k zabezpečení práce systému.
- *Shoda v účinnosti* {efficiency compliance} – podobný význam jako u funkčnosti.



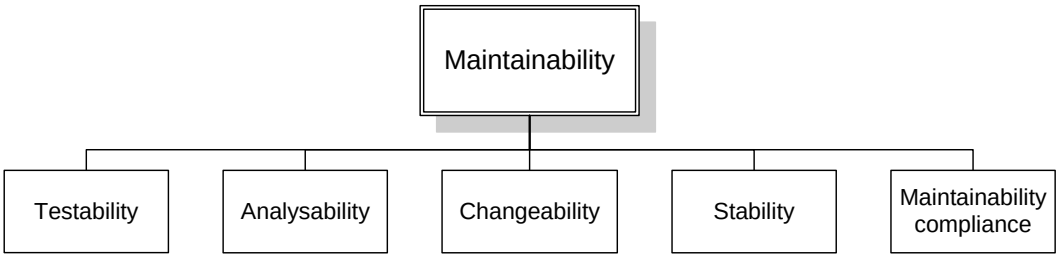
Obr. 5. Podcharakteristiky účinnosti [Struska, Vaníček 2005a]

2.2.1.5 Udržovatelnost {Maintainability}

Udržovatelnost je vymezena jako schopnost IS či softwarového produktu být modifikován. Modifikace zahrnují opravy nedostatků, vylepšování, adaptaci vzhledem ke změnám prostředí, změnám požadavků a změnám funkční specifikace [Vaníček 2004].

Mezi podcharakteristiky udržovatelnosti patří [ISO/IEC 9126]:

- *Analyzovatelnost* {analysability} – je vymezena jako schopnost systému usnadnit nalezení vady v případě výskytu poruch a schopnost určit, co má být změněno, aby byla vada odstraněna.
- *Měnitelnost* {changeability} - Měnitelnost je vymezena jako schopnost systému usnadňující provedení modifikace.
- *Stabilnost* {stability} - je vymezena jako schopnost systému zabránit nežádoucím důsledkům provedených modifikací.
- *Testovatelnost* {testability} - je vymezena jako schopnost systému zabezpečit snadnou validaci po provedení modifikace.
- *Shoda v udržovatelnosti* {maintainability compliance} – podobný význam jako u funkčnosti.



Obr. 6. Podcharakteristiky udržovatelnosti [Struska, Vaníček 2005a].

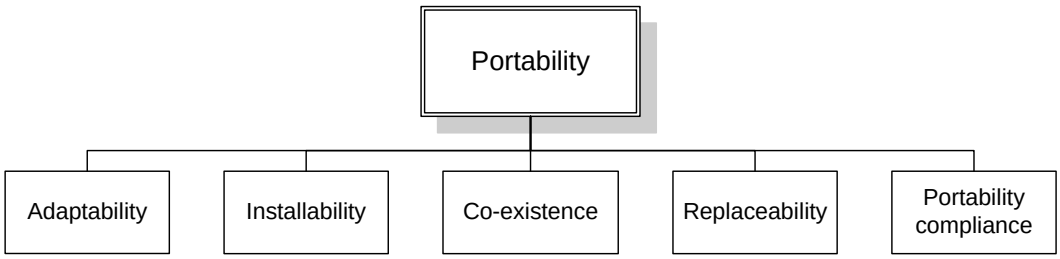
2.2.1.6 Přenositelnost {Portability}

Přenositelnost je vymezena jako schopnost IS softwarového produktu být přenesen z jednoho prostředí do jiného.

Mezi podcharakteristiky přenositelnosti patří [ISO/IEC 9126]:

- *Přizpůsobitelnost* {adaptability} - je vymezena jako schopnost systému být vlastními prostředky, které jsou jeho součástí, v průběhu používání přizpůsoben různým prostředím, v kterých má být využíván.
- *Instalovatelnost* {installability} - je vymezena jako vlastnost systému být zaveden tak, aby vyhovoval použití a práci v konkrétním prostředí.
- *Slučitelnost* {co-existence} - je vymezena jako schopnost systému pracovat společně s jinými systémy ve společném prostředí a využívat společné zdroje.
- *Nahraditelnost* {replaceability} - je vymezena jako schopnost systému nahradit funkci jiných systémů, určených pro stejný účel a pracujících ve shodném nebo podobném prostředí.

- *Shoda v přenositelnosti* {portability compliance} – podobný význam jako u funkčnosti.



Obr. 7. Podcharakteristiky přenositelnosti [Struska, Vaníček 2005a].

3 Techniky odhadu složitosti vývoje IS

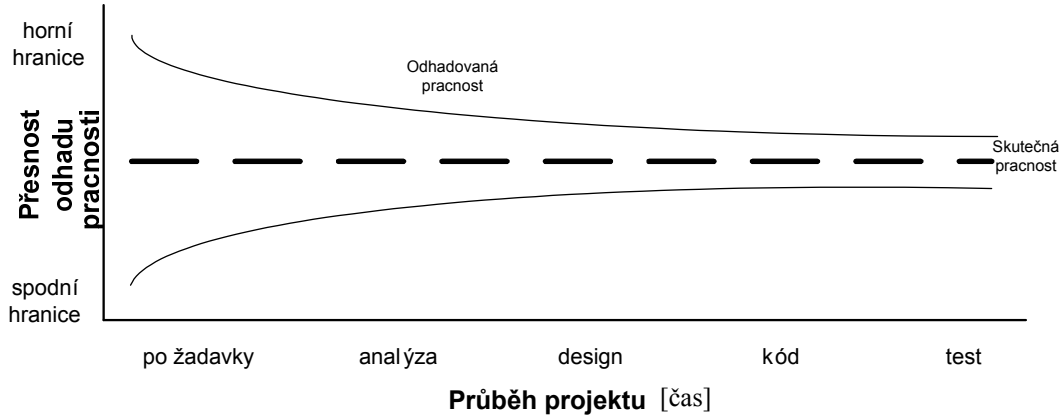
V počátečních fázích vývoje IS, kdy o požadovaném systému je k dispozici málo informací, je problém správného odhadu nejpalčivější. Právě tyto předběžné odhady jsou požadovány pro vypracování vhodné nabídky a uzavření smlouvy nebo při rozhodování o výhodnosti realizace projektu vlastními silami.

Je proto mimořádné důležité, aby po skončení etapy specifikace byl připraven alespoň hrubý odhad složitosti úkolu. To je důležité proto, aby bylo možno odhadnout, jak dlouho bude trvat etapa tvorby, implementace, zkoušení a uvedení do provozu. Dalším důvodem proč se do těchto odhadů pouštět je důležitost výstupů z pohledu rozhodnutí, zda se do tohoto projektu vůbec pustit. Někdy se totiž může stát, že náklady, které by musely být vynaloženy dalece převyšují možnosti realizátora. Je zřejmé, že „dříve znamená lépe“. Tzn. včasné odstoupení od neperspektivního projektu přinese úspory omezených prostředků, které mohou být investovány do jiných projektů.

Samozřejmě výsledky metod představených v této práci je možné v průběhu projektu zpřesňovat, takže se může stát, že se firma rozhodne od projektu odstoupit po dokončení jeho úvodních fází. I v těchto případech mohou úspory z toho rozhodnutí vysoce převyšovat konečně vynaložené prostředky.

Pro vypracování předběžné nabídky je dobré vědět, kdy bude daný produkt k dispozici. Dalším důležitým výstupem představených metod je časová náročnost realizace projektu, může se totiž ukázat, že okolnosti a možnosti dodavatelské firmy nedovolí dokončit požadovaný software včas, ale bude k dispozici v době, kdy již o požadovaný systém nebude potřebný zájem.

Z výše popsaných důvodů je zřejmé, že etapa specifikace je nejvýznamnější dobou pro rozhodnutí, zda se do vývoje softwaru pustit či nikoli.



Obr. 8. Zpřesňování odhadu pracnosti v průběhu projektu [Software metrics 2002].

Techniky odhadu pracnosti

Dominantní a klíčovou složkou nákladů na pořízení IS jsou náklady vyplývající z vývoje (náklady na mzdy vývojářů, nájemné kanceláří, sítě a komunikace, pojištění a daně). Ze všech ostatních nákladů, jako je např. cena hardware, licenční poplatky na SW, školení, se pracnost odhaduje nejhůře.

Podle Vahalíka [Vahalík 2004] existují tři kategorie technik přístupu k odhadu velikosti a ceny softwaru:

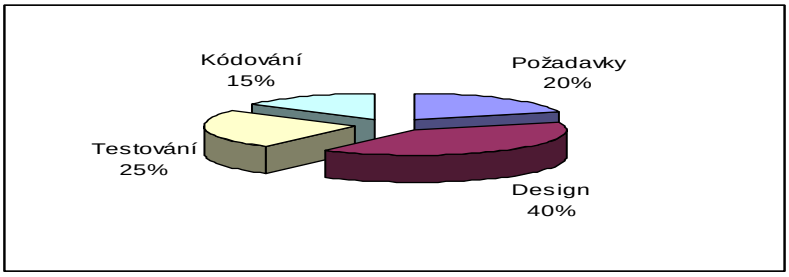
1. Expertní odhady.
2. Analogie.
3. Odhady založené na modelu.
    a. Analýza Function Points.
    b. Analýza Feature Points.
    c. Analýza UCP.
    d. COCOMO.

3.1 Expertní odhady

Technika vychází z odhadu nezávislého experta, který na základě svých předešlých zkušeností provede odhad. Expertní odhad je užitečný především v případech, kdy není dostatek měřitelných atributů, jež by mohly poskytnout data, která by byla vstupem pro jinou z metod.

Jorgensen a Sjoberg [Jorgensen, Sjoberg 2004] identifikovali jako nevýhodu metody to, že jde o subjektivní pohled experta na projekt. Z tohoto důvodu doporučují, aby odhadující expert neměl informace týkající se přesného očekávání zákazníka. Tento krok se doporučuje z toho důvodu, že znalost zákazníkových očekávání ovlivňuje expertův odhad i podvědomě. Dále platí, čím více expertů provádí odhad, tím jsou výsledky odhadu objektivnější.

Konzultanti společnosti Software metrics [Software metrics 2003] navrhli následující postup, který je založen na myšlence, že při expertním odhadu je pracnost fáze analýzy (požadavků) 20%, pak na základě znalosti požadavků a problémových oblastí může expert odhadnout pracnost analýzy. Jedna z možností rozložení pracnosti mezi jednotlivé etapy je zobrazena na obrázku 9. Další přístupem k odhadu pracnosti pomocí expertního odhadu je vynásobením pracnosti analýzy požadavků pěti (počet fází životního cyklu projektu IS), (5·20% = 100%) potom získá odhad celkové pracnosti.



Obr. 9. Fáze projektu s procentním rozložením pracnosti mezi jednotlivé fáze [Software metrics 2003].

3.2 Analogie

Podle konzultantů společnosti Software metrics [11] je tato technika založena na odhalení rozdílů mezi podobnými projekty a odhadu jejich dopadu na pracnost. Důležitým vstupem jsou zde data získaná z podobných projektů, na jejichž základě se odhad provede. Při použití této techniky musí být velmi pečlivě posuzována podobnost nejen vytvářeného IS, ale též podmínek pro jeho splnění.

3.3 Dekompozice

Podle elektronického zdroje London South Bank University [19] začíná dekompozice vytvářeného IS svůj odhad identifikací částí vhodných k rozkladu na nižší komponenty v rámci

produktu nebo specifikace projektových úkolů na nižších úrovních. Odhady pracnosti jsou získány dle požadovaného úsilí pro vytvoření/ dodání jednotlivých komponent nebo splnění úkolů na nižších úrovních.

Celkové odhady pracnosti jsou získány součtem odhadů jednotlivých komponent, které v praxi mohou záviset na průměrném úsilí, které může být upraveno na základě složitosti specifických komponent nebo složitosti specifických úkolů.

Mezi nejčastěji používaný typ dekompozice patří podle [19] tvorba systému zdola-nahoru {bottom - up method}. Metoda funguje následujícím způsobem, celkový IS je rozdělen do vzájemně souvisejících celků, které jsou vyvinuty samostatně a poté spojeny ve výsledný produkt. Metoda je pro techniku dekompozice vhodná z toho důvodu, že může vycházet z rozkladu IS, který byl proveden pro odhad pracnosti a postupně ho skládat ve společný celek.

3.4 Odhady založené na modelu

3.4.1 Metoda Function Points

3.4.1.1 IBM Function Points

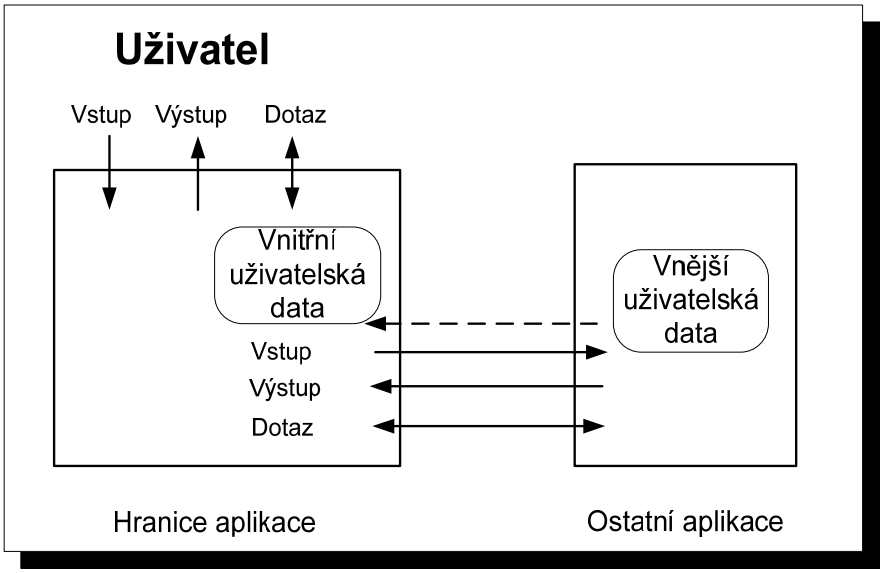
První metodu toho typu vyvinul A. Albrecht v laboratořích IBM v polovině 70. let minulého století za účelem odhadu složitosti projektů IS. Oficiálně byla publikována až v roce 1983 [Albrecht, Gaffney 1983]. Metoda IBM Function Points se stala základem pro vznik dalších níže uvedených modifikací metody Function Points.

Obecně je známo, že se jedná o poněkud problematický a neúplný podklad pro odhad složitosti, která jistě závisí na mnoha dalších faktorech, než pouze na rozsahu funkcí. Tyto ostatní faktory se mohou pro funkční jednice uplatnit pouze nepřímo.

Aby bylo možné provádět odhady pomocí metody Function Points je nutné provést nejprve odhad požadavků na vyvíjený IS. Dále se určí **charakteristiky** systému (viz tabulka 7) a následně odhad **funkcí** v systému. Funkce jsou součástí odhadu systémových charakteristik,

které se skládají z pěti hlavních komponent, jenž používají ty metody Function Points, které jsou přímým následovníkem metody IBM Function Points.

Odhad systémových charakteristik se skládá z objemu funkcí vstupujících do/ z aplikace, funkce lze rozdělit do dvou skupin, první jsou označovány jako transakční funkce a druhé jako datové funkce. Definice transakčních a datových funkcí obsahují následující odstavce.

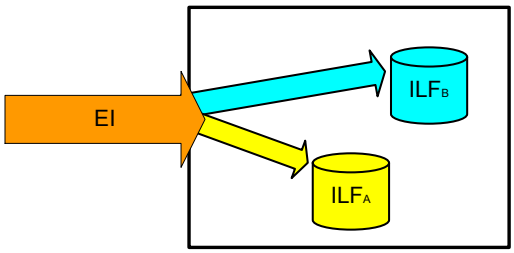


Obr. 10. Grafické znázornění užití metody IBM Function Points [SPR 2002]

Definice pěti hlavních komponent metody podle Longstreeta ze společnosti SoftwareMetrics [Longstreet 2004]:

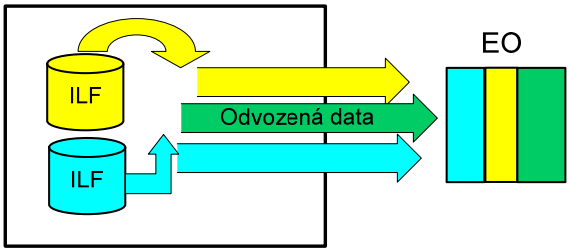
Typy transakčních funkcí

- Vstupy do aplikace {EI – External Inputs} – jsou základní procesy, ve kterých data procházejí hranice z venku dovnitř systému. Data mohou pocházet z datové vstupní obrazovky nebo z jiné aplikace. Data mohou být využita k údržbě jednoho nebo více vnitřních logických souborů. Dále data mohou nést buď kontrolní nebo obchodní informace. Jestliže jde o kontrolní informace, pak nemusí docházet k aktualizaci vnitřních logických souborů. Obrázek reprezentuje jednoduchý EI, který aktualizuje 2 ILF's (FTR's – File Types Referenced).



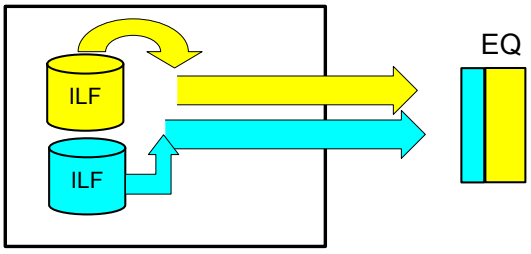
Obr. 11. Grafické znázornění External Inputs

- Výstupy z aplikace {EO – External Outputs} – základní procesy, ve kterých získaná data přecházejí hranice systému z vnitřku ven. Dodatečně EO může aktualizovat ILF. Data vytvářející reporty nebo výstupní soubory posílané dalším aplikacím. Jeden nebo více výstupních souborů jsou vytvořeny z jednoho nebo více vnitřních a vnějších logických souborů. Následující obrázek ukazuje příklad EO s 2 FTR's, obrázek obsahuje také odvozenou informaci (zeleně), vznikla odvozením ze dvou vnitřních logických souborů (ILF's).



Obr. 12. Grafické znázornění External Outputs

- Uživatelské dotazy {EQ – External inQuiry} – základní proces se vstupními i výstupními komponentami, který výsledná data získává z jednoho nebo více vnitřních a vnějších logických souborů. Vstupní proces neaktualizuje žádné vnitřní logické soubory a výstup z aplikace neobsahuje odvozená data. Níže umístěný obrázek zobrazuje EQ s dvěmi ILF's bez odvozených dat.



Obr. 13. Grafické znázornění External inQuiry

Typy datových funkcí

- Vnitřní logické soubory {ILF – Internal Logical Files} – jde o uživatelem identifikovatelnou skupinu logicky svázaných dat, která se sdílí pouze v rámci hranic aplikace a je udržována prostřednictvím vnějších vstupů.
- Vnější soubory rozhraní {EIF – External Interface Files} – jde o uživatelem identifikovatelnou skupinu logicky svázaných dat, které jsou použity pouze pro referenční účely. Data jsou umístěna zcela mimo aplikaci a jsou udržována jinou aplikací pomocí jejich společného rozhraní. Vnější logické soubory jsou považovány za vnitřní logické soubory spolupracující aplikace.

3.4.1.2 IFPUG Function Points

První metoda v seznamu představených modifikací metody Function Points je známa pod označením **IFPUG Function Points** (IFPUG = International Function Points Users Group). Podle skupiny autorů [IFPUG 2002], kteří IFPUG sestavili, je metoda založena na původním konceptu metody FP, která byla navržena v laboratořích IBM v sedmdesátých letech minulého století. Metoda funguje jako nástroj pro měření funkční hodnoty obchodních požadavků aplikačního softwaru viděného z pohledu uživatele softwaru.

Funkční hodnotu obchodních požadavků chápou autoři [IFPUG 2002] tak, že prostřednictvím uživatelského pohledu je vytvořen formální popis obchodních požadavků na vytvářený SW pomocí uživatelského jazyka. Aby mohly být uživatelské požadavky promítnuty do výsledného SW, musí je vývojoví pracovníci přeložit do programovacího jazyka.

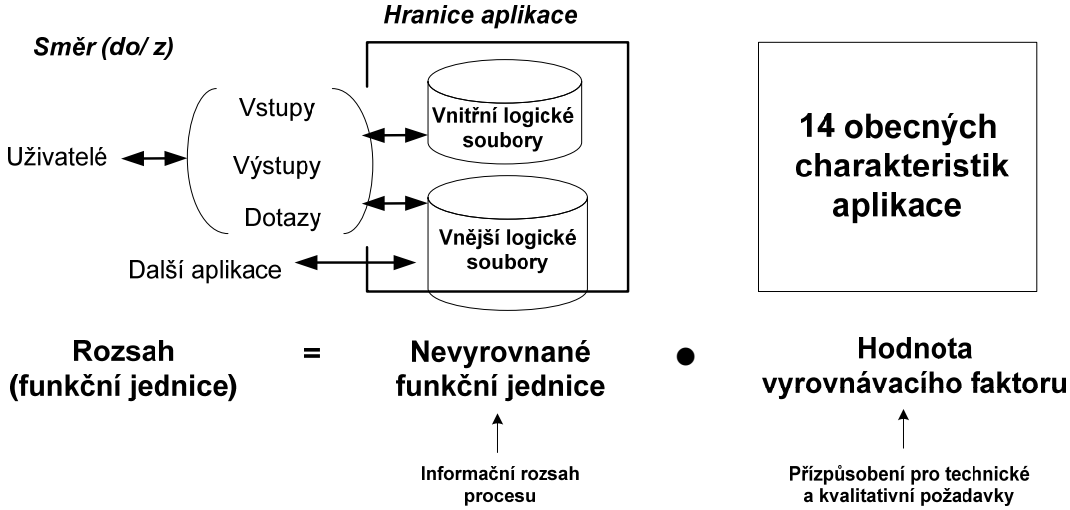
Z tohoto přístupu podle IFPUG [IFPUG 1999] vyplývá, že detailnější část funkčnosti, která musí být vyvinuta, zůstává uživatelsko-obchodním potřebám skryta. Jednoduchým příkladem je např.: pro budoucího uživatele není příliš podstatné, zda je požadovaná funkčnost zajištěna využitím jednoho procesoru či distribuována na několik vzájemně spolupracujících procesorech. Uživatel vidí pouze výslednou funkčnost a jen jednu hodnotu bez ohledu na další implementační úvahy. Z tohoto hlediska je funkčnost softwaru v jednotlivých vrstvách pod aplikační úrovní pro „normálního uživatele“ neviditelná a další potřeba ji měnit je ignorována.

3.4.1.2.1 Koncept metody IFPUG

Jak už bylo výše naznačeno, metoda IFPUG Function Points je přímým následovníkem původní metody IBM Function Points vyvinuté v polovině sedmdesátých let. Autoři IFPUG [IFPUG 2002] rozložili měřený software nebo jeho FUR (Functional User Requirements) do třech typů elementárních procesů (vstupy, výstupy a požadavky) a dvou typů souborů (vnitřní a vnější logické soubory) – definice viz výše. Každá komponentu poté klasifikovali podle ordinální stupnice:

jednoduchá – průměrná – složitá

a ohodnotili počtem funkčních bodů závisejících na její složitosti založené na počtu datových položek, složitosti formátu, lidských faktorech, atd.



Obr. 14. Schéma výpočtu metody IFPUG Function Points [Edwards 2007]

Složitost jednoduchého procesu závisí na počtu typů datového prvku na komponentu a na počtu typů souborů ovlivňujících jeho zpracování. Např. jednoduchý, průměrně složitý a složitý výstup má v případě nevyrovnané funkční jednice hodnocení 4, 5 a 7 nekorigovaných funkčních jednic (UFP) v tomto pořadí – viz tabulka 3. Hodnoty pro datové funkce.

Složitost souboru naproti tomu závisí nejen na typech datového prvku, ale také na počtu jeho záznamových typů. V případě, že je-li třeba zjistit hodnotu jednoduchého, středně složitého nebo složitého vnitřního logického souboru dostaneme hodnoty 7, 10 a 15 nekorigovaných funkčních jednic (unadjusted Function Points – UFP) – viz tabulka 5. Hodnoty pro transakční funkce.

Suma hodnot UFP všech jejích komponent dává nevyrovnanou hodnotu části softwaru. Tato hodnota je dále násobena s faktorem VAF, který je určen příspěvky 14 technických a kvalitativních faktorů poskytující korekční část celkové hodnoty funkčních jednicích.

3.4.1.2.2 Výpočet IFPUG Function Points

A. Odhad systémových charakteristik (odhad CHS)

Poté co byla každá komponenta zařazena mezi pět hlavních (EI's, EO's, EQ's, ILF's nebo EIF's) a ohodnocena jako jednoduchá, průměrná nebo složitá, přejde se k hodnocení transakcí (EI's, EO's, EQ's), které je založeno na počtu aktualizovaných nebo referenčních (File Types Referenced – FTR's) souborů a počtu prvků datových typů (Data Element Types – DET's). Pro soubory typu ILF's a EIF's je jejich hodnota stanovena na základě dvou matic typů záznamových (Record Element Types – RET's) a datových prvků (Data Element Types – DET's).

Typ záznamového prvku je uživatelsky rozpoznatelná podskupina datových prvků v rámci ILF nebo EIF. Typ datového prvku je jedinečné uživatelsky rozpoznatelné pole.

Každá z následujících tabulek obsahující číselné hodnocení participuje na klasifikaci souvisejícího procesu. Např. EI, které odkazuje nebo aktualizuje 2 typy referenčních souborů (FTR's) a má 7 datových prvků, dostane přiděleno ohodnocení průměrné, s výslednou hodnotou 4.

Tab. 1. Hodnocení EI [SPR 2002]

FTR's	DATOVÉ PRVKY		
	1 - 4	5 - 15	> 15
0 – 1	Nízký	Nízký	Průměrný
2	Nízký	Průměrný	Vysoký
3 nebo více	Průměrný	Vysoký	Vysoký

Tab. 2. Společné hodnocení EO a EQ [SPR 2002]

FTR's	DATOVÉ PRVKY		
	1 - 5	6 - 19	> 19
0 – 1	Nízký	Nízký	Průměrný
2 - 3	Nízký	Průměrný	Vysoký
> 3	Průměrný	Vysoký	Vysoký

Tab. 3. Hodnoty pro datové funkce [SPR 2002]

ODHAD	HODNOTY		
	EO	EQ	EI
Nízký	4	3	3
Průměrný	5	4	4
Vysoký	7	6	6

Postup výpočtu hodnot EQ's je stejný jako u komponent vnější výstupy a vnější vstupy, specifičnost komponenty EQ je v ohodnocení ordinální stupnicí (nízký, průměrný nebo vysoký), kde kopíruje hodnoty komponenty EO, ovšem přidělená hodnota odpovídá komponentě EI. Hodnocení je založeno na celkovém počtu unikátních (kombinující unikátní vstupní a výstupní strany) datových elementů (DET's) a typech referenčních souborů (FTR's) (kombinující unikátní vstupní a výstupní strany). Jestliže stejná FTR je užita na obou vstupních a výstupních stranách, pak dochází k počítání pouze v jeden časový okamžik. Jestliže stejná matice DET je užita na obou vstupních i výstupních stranách, potom dochází k počítání pouze v jeden časový okamžik.

Pro obě matice ILF a EIF platí, že počet typů záznamových prvků a počet typů datových elementů je užit pro ohodnocení v ordinální stupnici nízký, průměrný a vysoký. Typ záznamového prvku je definovaný jako uživatelsky rozpoznatelná podskupina datových prvků v rámci ILF nebo EIF. Naproti tomu typ datového prvku (DET) je určen jako jedinečné uživatelsky rozeznatelné pole na ILF nebo EIF.

Tab. 4. Společné hodnocení ILF a EIF [SPR 2002]

RET'S	DATOVÉ PRVKY		
	1 - 19	20 - 50	> 50
1	Nízký	Nízký	Průměrný
2 – 5	Nízký	Průměrný	Vysoký
> 5	Průměrný	Vysoký	Vysoký

Tab. 5. Hodnoty pro transakční funkce [SPR 2002]

ODHAD	HODNOTY	
	ILF	EIF
Nízký	7	5
Průměrný	10	7
Vysoký	15	10

Výsledek každé úrovně složitosti každého typu komponenty může být vložen do tabulky, tak jako následující. Každá vypočtená hodnota je násobena číselným hodnocením s pevně stanovenou hodnotou určité váhy. Vypočtené hodnoty jsou po řádkách sečteny, čímž je získána hodnota celého řádku každé komponenty. Sečtením tohoto sloupce je vypočten celkový počet nekorigovaných funkčních jednic (viz tabulka 6).

Tab. 6. Tabulka pro výpočet celkového počtu funkčních jednic [SPR 2002]

Typ komponenty	**Složitost komponent**			
	Nízký	Průměrný	Vysoký	Celkem
Vnější vstupy	__ • 3 = __	__ • 4 = __	__ • 6 = __	
Vnější výstupy	__ • 4 = __	__ • 5 = __	__ • 7 = __	
Vnější dotazy	__ • 3 = __	__ • 4 = __	__ • 6 = __	
Vnitřní logické soubory	__ • 7 = __	__ • 10 = __	__ • 15 = __	
Vnější soubory rozhraní	__ • 5 = __	__ • 7 = __	__ • 10 = __	
	Celkový počet nekorigovaných funkčních jednic (UFP)			
	Vynásobíme hodnotou korekčního faktoru (VAF)			
	Upravené funkční jednice celkem (FP)			

B. Odhad charakteristiky podmínek vývoje – Korekční faktor (odhad CHP)

Pro získání hodnoty korekčních funkčních jednic, která je zobrazena v předposledním řádku tabulky 6., je nutné:

- zhodnotit 14 obecných charakteristik systému vyjmenovaných v tabulce 7. Hodnocení probíhá pomocí stupnice od 0 do 5. (0 – faktor nemá žádný vliv, 5 – faktor má velmi významný vliv). Charakteristiky určují pracnost vývoje bez ohledu na podmínky, ve kterých bude systém vyvíjen,
- po ohodnocení charakteristik z tabulky 7. se sečtou hodnoty (0-5) všech 14 obecných charakteristik,
- a součet se vynásobí 0,01 a následně sečte s 0,65.

Těmito kroky byl proveden výpočet hodnoty korekčního faktoru (VAF), pro názornost celý vzorec výpočtu korekčního faktoru je následující:

$$VAF = 0{,}65 + (\text{suma obecných charakteristik systému} \cdot 0{,}01)$$

Tab. 7. Charakteristiky systému

Obecné charakteristiky systému		Krátký popis
1	Datová komunikace	Kolik komunikačních zařízení podporuje přenos nebo výměnu informací s aplikací nebo systémem?
2	Distribuované zpracování dat	Jak jsou řízena, distribuovaná data a zpracování funkcí?
3	Výkon	Byla časová odezva nebo výkon v souladu s požadavky uživatele?
4	Intenzita využití konfigurace	Jaká je intenzita využití současné HW platformy, na které budou aplikace vykonávány?
5	Transakční míra	Jak často jsou transakce zpracovávány – denně, týdně, měsíčně, atd.?
6	On-line vkládání dat	Jaké procento informací je vkládáno on-line?
7	Výkonnost koncového uživatele	Byla aplikace navržena, aby zlepšila pracovní výkon koncového uživatele?
8	On-line aktualizace	Kolik vnitřních logických souborů je aktualizováno on-line?
9	Složité zpracování	Disponuje aplikace rozsáhlým logickým a matematickým zpracováním?
10	Znovupoužitelnost	Byla aplikace vyvinuta, aby uspokojila jednu nebo více uživatelských potřeb?
11	Jednoduchost instalace	Jak složitá je úprava a instalace?
12	Provozní jednoduchost	Jak účinně a/ nebo automatizovaně je aplikace startována, zálohována a obnovena?
13	Multifunkční využití	Byla aplikace speciálně navržena, vyvinuta a podporována, aby mohla být instalována pro multifunkční využití v rámci organizací?
14	Ulehčení změn	Byla aplikace speciálně navržena, vyvinuta, aby podporovala lehké zavedení změn?

C. Odhad celkového počtu funkčních jednic

Výsledná hodnota IFPUG Function Points je získána tak, že dojde k vynásobení hodnot vyrovnané a nevyrovnané části FP (viz tabulka 6.):

$$FP\ (IFPUG) = VAF \cdot UFP.$$

3.4.1.3 COSMICS - FFP

Další modifikace původní metody IBM Function Points je z dílny Common Software Measurement International Consorcium [3] a nese označení COSMICS – FFP. Autoři metody ji navrhli tak, že představuje koncept založený na tzv. „pohledu měření“ (MV) a nabízí „měřiči“ možnost výběru. Za prvé může být použita pro měření funkčního rozměru měřícího hlediska „koncového uživatele“, stejně jako v případě měření metodou IFPUG Function Points, nebo může být použita k měření funkčního rozměru z pohledu vývojového pracovníka.

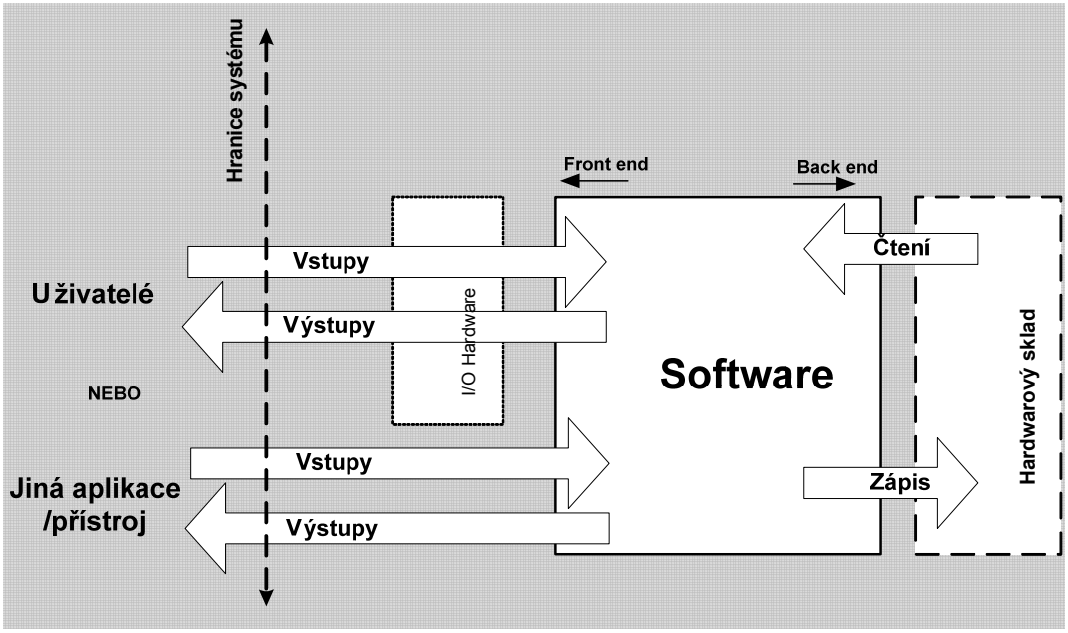
Jestliže je složitost měřena z měřícího pohledu „koncového uživatele“, může být podle autorů metody složitější vzájemné porovnání se složitostí ostatních projektů IS vytvářejících software odlišným způsobem s odlišnou technologií. Např. pokud software projektu *A* je vyvinut na jednoduché platformě, naproti tomu projekt *B* na mnohonásobné platformě různou technologií a projekt *C* vyžaduje práci na softwaru pod aplikační úrovní. Takové rozdíly jsou z měřícího pohledu koncových uživatelů neviditelné, a také neberou v úvahu míru velikosti těchto pracovních výstupů.

Metoda COSMIC – FPP tedy využívá spíše vývojářský pohled, který udává zvlášť hodnotu pro každou komponentu vícevrstvé (multi - level) architektury a pro každou jednotlivou vrstvu víceúrovňového softwaru. Takové hodnotící míry by mohly být využitelnější pro porozumění výkonu v průběhu vývoje a pro odhad cílů. Hodnoty různých měřících úhlů pohledu by neměly být porovnávány, ani vzájemně kombinovány.

Další přínos užití vývojářského pohledu v kombinaci s COSMIC – FPP je to, že všechny funkce alokované (rovnoměrně rozdělené) v rámci softwaru jsou zahrnuty do měřené hodnoty.

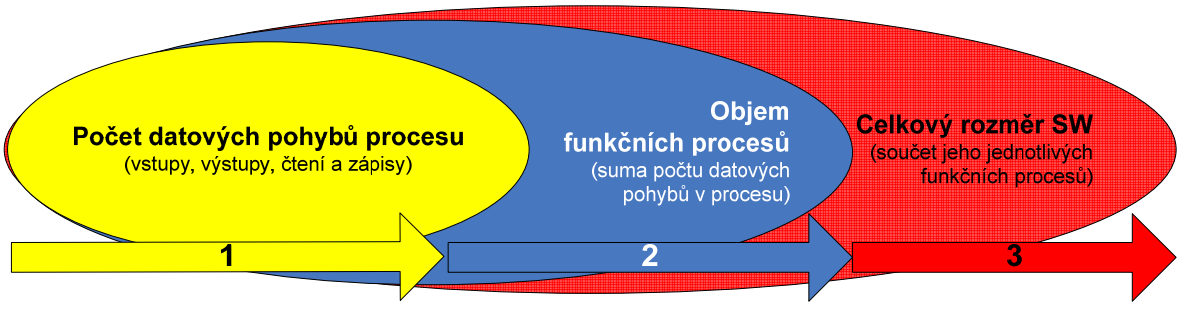
3.4.1.3.1 Koncept metody COSMIC – FPP

V metodě COSMIC – FPP [ISO/IEC 2003] je měřený software či jeho FUR (Function-User Requirements) rozdělen dle typu funkčního procesu, definovaného jako základní komponenta množiny funkčně-uživatelských požadavků zahrnujících jedinečnou soudržnou a nezávisle vytvořenou množinu typů datových pohybů (vstupy, výstupy, čtení, zápis) viz obrázek 15. Tato činnost je aktivována jedním nebo více typy spouštěcích příkazů buď přímo IS nebo nepřímo prostřednictvím aktora. Proces končí, pokud bylo provedeno vše, co je požadováno v typu spouštěcího případu.



Obr. 15. Schéma užití metody COSMIC – FPP [Edwards 2007]

Funkční typy procesů jsou rozloženy do dvou typových podprocesů (vstupy, výstupy) nebo dvou typů datových pohybů (čtení a zápisy), každý s předpokládaným propojeným objemem datové manipulace. Vstupy a výstupy jdou napříč rozhraním mezi měřeným softwarem a jeho uživateli (těmi mohou být osoby, další části softwaru nebo např. nějaký senzor, který komunikuje s měřeným softwarem). Čtení posunuje data z hardwarového skladu do prostoru měřeného softwaru, zatímco zápis posílá data opačným směrem, tzn. data zůstávají na protější straně běžícího funkčního procesu. Každý pohyb je oceněn hodnotou jednoho Cfsu (COSMICS Functional size unit).



Obr. 16. Postup výpočtu metodou COSMIC – FPP

Objem funkčního procesu je součet počtu jeho datových pohybů procesu a celkový rozměr softwaru je součet objemů všech jeho funkčních procesů. Nejjednodušší funkční proces má

objem 2 Cfsu (jeden vstup plus jeden výstup, nebo jeden zápis a jedno čtení), ale neexistuje žádná horní hranice velikosti funkčního procesu.

3.4.1.4 „Back-fire“ metoda Function Points

Metoda „Back - fire“ je podle konzultantů Software Productivity Research [SPR 2002] využitelná v případech zpětného hodnocení dříve realizovaných projektů a pro uživatele, kterým jsou bližší metriky vycházející z řádků zdrojového kódu. Metoda odhaduje funkční body na základě empirických vztahů již existujících mezi zdrojovým kódem a Function/ Feature Points ve všech známých jazycích. Hodnoty v tabulce 8. jsou založeny na průměrných hodnotách získané při měření již realizovaných projektů.

Tab. 8. Odhad poměru zdrojový kód/ FP pro jednotlivé programovací jazyky [Miller, Williams 2000]

Programovací jazyk	Zdrojový kód/ FP
Assembler	320
C	128
COBOL	107
Ada	70
DB Languages	40
Object Oriented	29
Query Languages	25
Generators	16

3.4.1.5 Softwarové nástroje pro metodu Function Points

Softwarový nástroj, který je možné použít pro výpočet hodnoty Function Points, se jmenuje FP RECORDER LITE V2.00. Byl vytvořen společností Chipsl (www.chipsl.com) a je k dispozici k stažení na stránkách www.softlookup.com (http://www.softlookup.com/display.asp?id=7109). Uživatel má na výběr způsob prezentace výsledků, kdy volí mezi vykreslením výsledků do grafů či tabulek dle definovaných kritérií.

Další softwarový produkt společnosti Charismatek má název Function Point WORKBENCH 6.0 (http://www.charismatek.com.au/_public1/html/fpw_demonstration.htm). Existuje ve dvou verzích profesionální a veřejná, výstupem jsou reporty generované v html formátu.

Tyto softwarové nástroje jsou vybrány z mnoha dalších, které podporují a zjednodušují výpočet Function Points. Jsou konstruovány tak, že uživatel pouze zadává vstupní hodnoty a software sám, aniž uživatel musí podrobně znát postup FP, vygeneruje výslednou hodnotu.

3.4.2 Metoda Feature Points

3.4.2.1 SPR Feature/ Function Points

3.4.2.1.1 Koncept metody SRP Feature Points

Metodu SPR Function Points vyvinul v roce 1986 Capers Jones a jeho organizace Software Productivity Research [13] při experimentální aplikaci metody Function Points na logické softwarové systémy, např. operační systémy, systémy pro přepojování telefonních linek, real time software, MIS software a další.

Původně Jones [Jones 2007] metodu SPR Function Points navrhl pro měření klasických manažerských IS. Při jejím testování zjistil její nevhodnost pro projekty v následujících oblastech:

- real time software – raketový obranný systém,
- systémový software – operační systémy,
- komunikační software – systémy pro přepojování telefonních linek,
- software řízení procesů – systémy pro řízení rafinérií,
- strojírenské aplikace – CAD, CIM, diskrétní simulace,
- matematický software a další.

Kvůli nedostatkům popsaných v kapitole 3.5 byla původní metoda SPR Function Points rozšířena tak, aby byla použitelná i pro systémy vyjmenované v předcházejícím odstavci. Vznikla tak metoda SPR Feature Points, která zahrnuje SPR Function Points a bere v úvahu také šestý parametr – algoritmus. Ten je rozšířením původních pěti parametrů metody IBM Function Points. Parametru algoritmus je přidělena váha 3. Metoda Feature Points tedy omezuje empirické váhy vnitřních logických souborů z původní průměrné hodnoty 10 na průměrnou hodnotu 7. To reflektuje poněkud omezený význam vnitřních logických souborů u systémových softwarů vzhledem k systémům informačním.

3.4.2.1.2 Výpočet SPR Feature Points

Při využití metody SPR Feature Points podle [SPR 2002] není nutné zjišťovat počet typů datových prvků, typů referenčních nebo záznamových souborů. Zároveň se nevyžaduje ohodnocení jednotlivých funkcí pomocí ordinální stupnice (jednoduchý-průměrný-složitý).

Na první pohled se tato metoda jeví jako daleko jednodušší a poskytuje rychlejší odhad složitosti budoucího IS.

A. Odhad hodnoty nekorigovaných Feature Points

Odhad hodnoty nekorigovaných Feature Points je založena na klasickém postupu sčítání transakčních a datových funkcí.

Identifikace funkcí systému

V prvním kroku se stejně jako u metody IFPUG Function Points zjistí počet datových a transakčních funkcí a navíc oproti metodě Function Points ještě algoritmů, ten se dále roznásobí vahami podle tabulky 9. Po jejich sečtení se získá celkový počet nekorigovaných Feature Points.

Tab. 9. Postup výpočtu nekorigovaných Feature Points [13]

Funkce	Váha	Výpočet	Celkem
EI	4	_ • 4 =	
EO	5	_ • 5 =	
EQ	4	_ • 4 =	
ILF	7	_ • 7 =	
EIF	7	_ • 7 =	
Algoritmus	3	_ • 3 =	
Celkový počet nekorigovaných Feature Points (FC)			

B. Odhad faktoru a multiplikátoru složitosti

V dalším kroku se doporučuje odpovědět na dvě skupiny otázek v tabulce 10., první skupina se dotýká složitosti problému a druhá datové složitosti.

Tab. 10. Složitost problému a datová složitost [SPR 2002]

	Složitost problému?
1	Jednoduché algoritmy a jednoduché výpočty?
2	Většina jednoduchých algoritmů a výpočtů?
3	Algoritmy a výpočty s průměrnou složitostí?
4	Některé obtížné nebo složité algoritmy?
5	Mnoho obtížných algoritmů a složitých výpočtů?

	Datová složitost?
1	Jednoduchá data s malým počtem proměnných?
2	Četnost proměnných, jednoduché vztahy mezi daty?
3	Multiplikační soubory, pole a datové průniky?
4	Složité struktury souborů a datové průniky?
5	Velmi složité struktury souborů a datové průniky?

Po vyhodnocení vyjmenovaných otázek se vyberou dvě, které nejlépe charakterizují vznikající IS a jejich pořadová hodnota se sečte. Dle její výše se vybere z tabulky 11 příslušný multiplikátor složitosti.

Tab. 11. Multiplikátor složitosti [SPR 2002]

Složitost problému a datová složitost	2	3	4	5	6	7	8	9	10
Multiplikátor složitosti (CoM)	0,6	0,7	0,8	0,9	1	1,1	1,2	1,3	1,4

C. Odhad celkového počtu Feature Points

Výsledný počet SPR Feature Points získáme roznásobením nekorigované části a multiplikátoru složitosti:

$$FP\ (SPR) = FC \cdot CoM.$$

3.4.2.2 Softwarové nástroje pro výpočet Feature Points

Software podporující výpočet metody Feature Points se jmenuje *SPR Knowledge Plan*, autorem je společnost SPR. Produkt je zabudovatelný do celopodnikového IS společnosti. Systém umožňuje provádět výpočet pracnosti projektu a zároveň obsahuje srovnávací data z 11 000 různých realizovaných softwarových projektů. Tato srovnávací data jsou pro tento software jedinečná tím, že uživatel zjistí pracnost svého projektu a následně ji může porovnat

s podobným již realizovaným projektem, aby zjistil, jak si jeho odhad ve srovnání s tím podobným stojí. Dá se říci, že tento softwarový nástroj kombinuje metodu Feature Points a metodu odhadu na základě analogie.

3.4.3 Metoda UCP

3.4.3.1 Koncept metody UCP

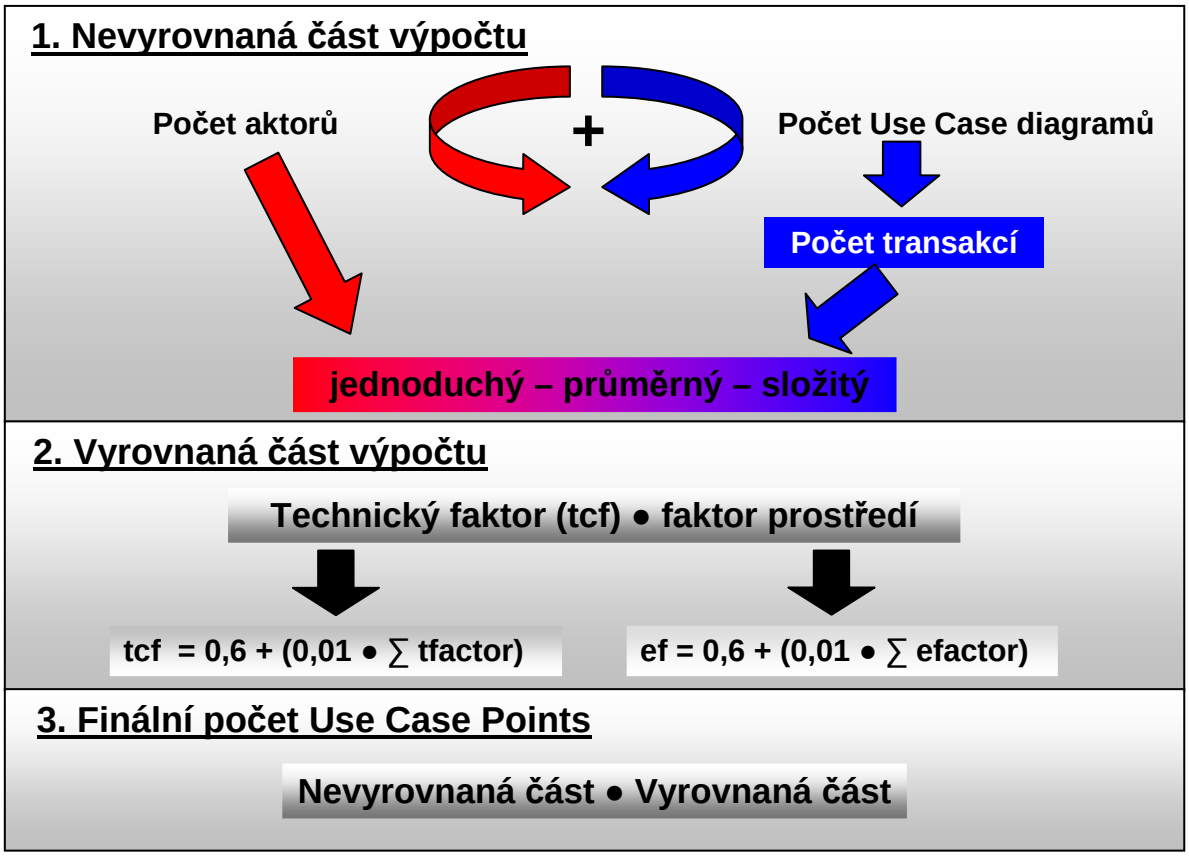
Metoda byla vyvinuta v roce 1993 Gustavem Karnerem [Karner 1993]. Autor ji vybudoval na myšlence vycházející z analýzy IBM Function Points (detaily v kapitole 3.4.1 Metoda Function Points), která vznikla v 70. letech ve výzkumných laboratořích IBM, za jejíhož autora je pokládán Allen Albrecht.

Podle Karnera [Karner 1993] mohou být modely Use Case jednoduše použity pro odhad složitosti vývoje softwaru a jeho testování. Pro použití metody UCP je velmi důležitá srozumitelnost jednotlivých Use Case, vztahů mezi nimi a rolí zapojených aktorů.

3.4.3.2 Výpočet složitosti metodou UCP

Výpočet UCP podle Karnera [Karner 1993] se skládá ze čtyřech částí:

- Nekorigovaná část UCP.
  - Počet aktorů
    - Aktor je v této práci chápán jako specifický název objektu, který určitým způsobem přijde do kontaktu s vyvíjeným IS.
  - Počet Use Case (případ užití).
    - Use Case diagram pro potřeby této práce zachycuje jednotlivé požadavky na systém tak, jak se bude jevit zvenčí.
- Technický faktor.
- Faktor prostředí.
- Faktor produktivnosti.



Obr. 17. Struktura užití metody UCP [Struska 2006a]

3.4.3.2.1 Nekorigovaná část výpočtu UCP

Výpočet této části se dále rozpadá do nižších úrovní – počet aktorů a počet Use Case.

A identifikace, klasifikace a přidělení vah aktorům {unadjusted actor weights – uaw}
Vychází se z předpokladu, že aktoři jsou externími objekty, které mají se systémem vzájemný vztah. Ve většině případů se může jednat o koncové uživatele, další programy, datové sklady, atd. Měly by být součástí projektové dokumentace k daným Use Case.

Zjištění aktoři se dále rozčlení do třech kategorií:

jednoduchý – průměrný – složitý.

Jednoduchý – jiný systém s rozhraním na měřený systém prostřednictvím různých programů (např. standardní aplikační program (nebo API)).

Průměrný – buď další systém nebo datový sklad, který je na měřený systém napojen prostřednictvím protokolu nebo textu založeném na rozhraní. Průměrný aktor spolupracuje se systémem přes protokol (např. HTTP, TCP/ IP, atd.).

Složitý – osoba spolupracující se systémem prostřednictvím grafického rozhraní (jde hlavně o koncové uživatele, kteří jsou klasifikováni v kategorii složitý).

Tab. 12. Kategorie – váhy aktorů

Typ aktora	Definice	Váha
Jednoduchý	Automatizované rozhraní	1
Průměrný	Interaktivní nebo protokolem řízené rozhraní	2
Složitý	Grafické rozhraní (lidský faktor)	3

Po rozdělení aktorů do třech kategorií, jsou jejich počty sečteny v rámci každé kategorie a roznásobeny přidělenými váhami. Jednoduchému aktorovi je přidělena váha 1, průměrnému váha 2 a složitému váha 3. Nevyrovnanou váhu aktora celkem získáme součtem vyvážených hodnot z jednotlivých kategorií.

Tab. 13. Výpočet nevyrovnané váhy aktorů.

Typ aktora	Váha aktora	Výpočet	Celkem
Jednoduchý	1	__ • 1 =	
Průměrný	2	__ • 2 =	
Složitý	3	__ • 3 =	
Nekorigovaná váha aktora celkem {Total unadjusted actor weight – uaw}			

B identifikace, klasifikace a přidělení vah Use Case {unadjusted Use Case weights – uucw}
Dalším krokem metody je určení počtu Use Case v systému. Každému Use Case by měly být přiděleny určité váhy závislé na počtu transakcí a/ nebo scénářů.

Transakce jsou definovány jako základní množina aktivit, které jsou buď vykonávány zcela nebo vůbec. Základem kategorizace je, kolik transakcí Use Case obsahuje.

V prvním kroku rozdělíme zjištěné Use Case do kategorií:

jednoduchý – průměrný – složitý.

Rozdělení se provede na základě počtu transakcí v Use Case diagramu, konkrétní počty transakcí, které jsou hraničními hodnotami mezi definovanými kategoriemi (viz tabulka 14).

Tab. 14. Souhrn typů Use Case a jejich popisů.

Typ Use Case	Popis	Váha
Jednoduchý	3 a méně transakcí nebo 5 a méně analyzovaných tříd	5
Průměrný	4 - 7 transakcí nebo 5 - 10 analyzovaných tříd	10
Složitý	7 a více transakcí nebo 10 a více analyzovaných tříd	15

Po rozdělení Use Case podle počtu transakcí mezi tři kategorie, přidělíme jednotlivým úrovním váhu a provedeme roznásobení, dále součty přes jednotlivé typy Use Case. Výsledkem tabulky je nekorigovaný počet Use Case (viz tabulka 15).

Tab. 15. Výpočet nevyrovnané váhy Use Case celkem.

Typ Use Case	Počet transakcí	Váha Use Case	Výpočet	Celkem
Jednoduchý	1 – 3	5	__ • 5 =	
Průměrný	4 – 7	10	__ • 10 =	
Složitý	8 >	15	__ • 15 =	
Nekorigovaná váha Use Case celkem (Total Unadjusted Use Case weight – uucw)				

C Výpočet nevyrovnané části UCP {Unadjusted Use Case Points – uucp}

Celkový počet nekorigovaných UCP získáme sečtením dvou výše vypočtených částí – nekorigovaná váha aktora (uaw) a nekorigovaná váha Use Case (uucw).

unadjusted UCP (uucp) = uaw + uucw

3.4.3.2.2 Technický faktor {Technical Factor}

Pro dokončení výpočtu celkové složitosti projektu je nutné určit tzv. technický faktor, který bude ovlivněn i úrovní znalostí zapojených zaměstnanců. Jejich úkolem bude ohodnocení vybraných 13 faktorů, které souvisejí s technickou stránkou vytvářeného IS. Hodnotící stupnice zahrnuje hodnoty od 0 do 5. Pokud bude faktoru přidělena hodnota 0, pak je jeho vliv na IS bezvýznamný, v případě 5 je velmi významný.

Tab. 16. Výpočet hodnoty technického faktoru UCP

Pořadí	Popis	Váha	Výpočet	Celkem	Komentář
t1	distribuovaný systém	2	__ • 2 =		
t2	doba reakce nebo požadovaná rychlost zpracování/ výkon	1	__ • 1 =		
t3	efektivnost koncového uživatele	1	__ • 1 =		
t4	složitost vnitřního zpracování	1	__ • 1 =		
t5	znovupoužitelnost kódu	1	__ • 1 =		
t6	jednoduchost instalace	0,5	__ • 0,5 =		
t7	jednoduchost užití	0,5	__ • 0,5 =		
t8	přenositelnost	2	__ • 2 =		
t9	snadnost změny	1	__ • 1 =		
t10	paralelní vývoj	1	__ • 1 =		
t11	speciální požadavky na bezpečnost	1	__ • 1 =		
t12	přímé zapojení třetí strany	1	__ • 1 =		
t13	požadavek speciálního tréninku	1	__ • 1 =		
Technický faktor celkem (Total technical factor – tFactor)					X

Všechny vyjmenované hodnoty technického faktoru v tabulce nemusí být ohodnoceny pokaždé na všech projektech, ale je vhodné vybrat pro konkrétní projekt ty nejvhodnější a těm přidělit nejvyšší hodnotu (tj. hodnota 5). Vysvětlení jednotlivých pojmů z tabulky je následující:

- distribuovaný systém – složitost architektury systému (zda systém běží na jednom či více strojích),
- doba reakce nebo požadovaná rychlost zpracování/ výkon,
- efektivnost koncového uživatele – schopnost uživatele řešit složité úkoly s pomocí IS jednodušeji a rychleji,
- složitost vnitřního zpracování – závisí na složitosti vnitřní struktury IS,
- znovupoužitelnost kódu – znamená schopnost celého nebo alespoň části kódu být znovu použit v tvorbě dalších částí,
- jednoduchost instalace – stupeň náročnosti postupných kroků při instalaci systému,
- jednoduchost užití - jde o rychlou naučitelnost a srozumitelnost pro uživatele při práci se systémem,
- přenositelnost – viz kapitola 2.2.1.6 této práce,
- snadnost změny – úroveň složitosti v případě úprav celého systému nebo jeho jednotlivých částí,
- paralelní vývoj – předpoklady pro tvorbu několika částí systému najednou,

- speciální požadavky na bezpečnost – v případech, kdy jsou od systému vyžadovány specifické bezpečnostní požadavky (cíle),
- přímé zapojení třetí strany – úroveň spolupráce se třetí stranou, za kterou je možno považovat např. nezávislého testéra nebo poradenskou firmu. Jejich úkolem je zajistit nezávislý pohled,
- požadavek speciálního tréninku – určených pro IT zaměstnance budoucích zákazníků.

Hodnoty (0–5) přidělené každému faktoru se roznásobí přidělenou váhou, dále se provede jejich součet. Takto získáme tzv. technický faktor (tFactor), který dále dosadíme do vzorce pro výpočet technického faktoru složitosti (technical complexity factor – tcf).

$$tcf = 0{,}6 + (0{,}01 \cdot tFactor)$$

3.4.3.2.3 Faktor prostředí {Environment Factor}

Vliv na odhad doby realizace projektu má také skutečnost související s dovednostmi a znalostmi zaměstnanců. Tyto vlivy jsou zastřešeny tzv. faktorem prostředí. Postup jeho hodnocení je stejný jako u technického faktoru, tzn. nejprve ohodnotíme 8 faktorů týkajících se prostředí, ve které IS vzniká. Máme opět k dispozici stupnici od 0 do 5 (0 – bezvýznamný, 5 – velmi významný dopad).

Tab. 17. Seznam faktorů prostředí UCP

Pořadí	Popis	Váha	Výpočet	Celkem	Komentář
e1	obeznámení s užitým projektovým modelem (např. RUP)	1,5	__ • 1,5 =		
e2	zkušenosti s aplikacemi	0,5	__ • 0,5 =		
e3	zkušenosti s objektovou orientací	1	__ • 1 =		
e4	kapacita vedoucího analytika	0,5	__ • 0,5 =		
e5	Motivace	1	__ • 1 =		
e6	vyváženost požadavků	2	__ • 2 =		
e7	zaměstnanci na částečný úvazek	1	__ • 1 =		
e8	složitost programovacího jazyka	1	__ • 1 =		
Faktor prostředí celkem (Total environment factor - eFactor)					X

Všechny vyjmenované hodnoty technického faktoru v tabulce nemusí být ohodnoceny pokaždé na všech projektech, ale je vhodné vybrat pro konkrétní projekt ty nejvhodnější a těm přidělit nejvyšší hodnotu (tj. hodnota 5). Vysvětlení jednotlivých pojmů z tabulky je následující:

- obeznámení s užitým projektovým modelem – úroveň znalostí metodik jednotlivých zaměstnanců pracujících na daném projektů IS,
- zkušenosti s aplikacemi – úroveň znalostí jednotlivých zaměstnanců u aplikací používaných v projektu (aplikace pro vývoj, návrh, realizace, atd.)
- zkušenosti s objektovou orientací – schopnost zaměstnanců pracovat v objektovém prostředí
- kapacita vedoucího analytika – objem doby, kterou je schopen vedoucí analytik věnovat odhadovanému projektu,
- motivace – úroveň motivace pro dokončení daného projektu včas a ve stanovených nákladech,
- vyváženost požadavků – schopnost zákazníka definovat své požadavky a četnost jejich pravděpodobných změn,
- zaměstnanci na částečný úvazek – počet zaměstnanců na částečný úvazek zapojených v projektu,
- složitost programovacího jazyka.

Po ohodnocení faktorů se jim přidělené hodnoty roznásobí váhami a výsledky se sečtou. Touto operací získáme celkovou hodnotu faktoru prostředí {total environment factor – eFactor}, kterou dosadíme do vzorce pro výpočet složitosti faktoru prostředí {Environment complexity factor – ecf}, z něhož získáme výslednou hodnotu faktoru prostředí.

$$ecf = 1{,}4 + (-0{,}03 \cdot eFactor)$$

3.4.3.2.4 Faktor produktivnosti {Productivity factor}

Stanovený počet člověkohodin na jednici UCP se liší v závislosti na různých vlivech (zkušenosti programátorského týmu, velikost vyvíjeného IS, atd.). Ve světové literatuře zabývající se metodou UCP obecně se doporučuje hodnota navržená Karnerem [Karner 1993], jde o 20 člověkohodin na jednu *aucp*. Pokud členové týmu včetně vedení společnosti nebudou spokojeni s výsledkem získaným přepočtem 20 člověkohodin, mohou použít jednoduché pravidlo, které jim umožní stanovit si svůj vlastní koeficient v intervalu doporučeném pro projekt.

S určitou mírou nejistoty Karner [Karner 1993] říká, že by se hodnota faktoru produktivnosti měla pohybovat mezi 15 a 36 člověkohodinami na jednu *aucp*. Vychází z jednoduchého předpokladu určení hodnoty faktoru produktivnosti z hodnot přiděleným jednotlivým faktorům

prostředí pomocí jednoduchého výpočtu, kterým projektový tým získá počet hodin zaměstnance na jednu Use Case jednici.

Pokud hodnoty přidělené faktorům e1 až e6 jsou všechny nižší než 3, jsou dále sečteny dohromady a k tomuto součtu jsou přidány přidělené hodnoty faktorů *e7* a *e8* v případě, že jsou větší než 3. Potom pokud je součet jejich výsledné hodnoty 2 nebo méně, doporučuje se použít 20 člověkohodin.

Tab. 18. Výpočet faktoru produktivnosti na základě přidělených hodnot faktoru prostředí

<table>
<tr><th>Pořadí</th><th>Popis</th><th>Podmínka</th><th>Výpočet</th><th>Výpočet</th></tr>
<tr><td>e1</td><td>obeznámení s užitým projektovým modelem (např. RUP)</td><td>přidělená hodnota < 3</td><td rowspan="6">Pokud podmínka platí, budou sečteny přidělené váhy – suma I.</td><td rowspan="8">(suma I. + suma II.) <= 2, pak se doporučuje hodnota 20 člověhodin na jednu UCP</td></tr>
<tr><td>e2</td><td>zkušenosti s aplikacemi</td><td>přidělená hodnota < 3</td></tr>
<tr><td>e3</td><td>zkušenosti s objektovou orientací</td><td>přidělená hodnota < 3</td></tr>
<tr><td>e4</td><td>kapacita vedoucího analytika</td><td>přidělená hodnota < 3</td></tr>
<tr><td>e5</td><td>motivace</td><td>přidělená hodnota < 3</td></tr>
<tr><td>e6</td><td>vyváženost požadavků</td><td>přidělená hodnota < 3</td></tr>
<tr><td>e7</td><td>zaměstnanci na částečný úvazek</td><td>přidělená hodnota > 3</td><td rowspan="2">Pokud podmínka platí, budou sečteny přidělené váhy – suma II.</td></tr>
<tr><td>e8</td><td>složitost programovacího jazyka</td><td>přidělená hodnota > 3</td></tr>
</table>

Pokud je výsledkem výpočtu hodnota v intervalu 3 – 4, pak doporučuje Karner [Karner 1993] 28 člověkohodin na *ucp*. V případě, že výsledek převyšuje hodnotu 5, pak doporučuje, aby byla znovu přehodnocena přijatelnost projektu. Projekt spadá do kategorie rizikových. Když se po přehodnocení dojde k podobnému výsledku, doporučuje se pracovat s 36 člověkohodinami na *ucp*.

3.4.3.2.5 Celkový počet UCP {Total UCP}

Pro celkový počet UCP dosadíme výše vypočtené části do vzorce UCP a roznásobením nevyrovnané části UCP s technickým faktorem a faktorem prostředí, získáme korekční počet UCP {adjusted UCP – aucp}.

$$aucp = uucp \cdot tcf \cdot ecf$$

V postupu výpočtu představeném v kapitole 3.4.3 Metoda UCP jsme se dopracovali k odhadu pracnosti vyvíjeného IS. Složitost je zde zatím odhadnuta bezrozměrným číslem, které vzejde ze vzorce pro korekční počet UCP. Pro překlenutí od složitosti ke skutečné pracnosti, je nutné roznásobit výslednou hodnotu stanoveným faktorem produktivnosti neboli počtem člověkohodin na jednici.

$$Pracnost = aucp \cdot pf$$

3.4.3.3 Softwarové nástroje pro metodu UCP

Odhad složitosti pomocí metody UCP umožňuje software Enterprise Architekt (http://www.sparxsystems.com.au/UCMetrics.htm), jehož součástí je modul Odhad projektu. Modul umožňuje provádět odhad složitosti automaticky v průběhu návrhu IS.

Dále existuje velké množství různých nástrojů podporujících metodiku UML a nabízejících uživateli možnosti provádění odhadu složitosti vyvíjeného IS.

3.4.4 Metoda COCOMO

3.4.4.1 Koncept metody COCOMO

Důležitým vstupním údajem pro jakékoliv plánování a řízení prací na projektech IS a softwarových produktů, jak už bylo v kapitolách výše zmíněno, je odhad nákladů, které bude třeba na vývoj vynaložit. Tento odhad lze provést nejdříve v okamžiku, kdy se ví, co se bude vyvíjet. Tedy poté, kdy je k dispozici specifikace problému. Nejčastějším prostředkem v této rané etapě je odhad objemu pomocí metody Function Points (viz kapitola 3.4.1 Metoda Function Points).

Podle Vaníčka [Vaníček 2004] je potřeba odhad v průběhu vývoje stále korigovat a zpřesňovat v závislosti na tom, jaká vstupní data máme k dispozici. V průběhu realizace projektu lze užít některý z odhadů složitosti vycházející z grafu řízení či hierarchického rozkladu. Pro odhad mezimodulové složitosti pak odhady soudržnost modulů, spřaženost modulů, respektive odhady složitosti objektového řešení.

Po dokončení implementace doporučuje Vaníček [Vaníček 2004] užít odhady založené na zdrojovém kódu softwaru, které je ovšem vhodné korigovat s odhady podle předchozích odstavců, jenž hodnotí nejen pouhý „objem“ kódu, ale i složitost toku řízení, respektive rozkladu, který kód vyjadřuje. Tyto údaje jsou ovšem k dispozici příliš pozdě na to, aby byly použitelné pro rozhodování o tom, že se za danou cenu raději projekt řešit nebude, a obvykle i příliš pozdě na to, abych bylo řešení opraveno. Podle Vaníčka [Vaníček 2004] však mohou být velmi užitečné pro hodnocení řešitelského kolektivu a jako zkušenost „pro příště“. Lze doporučit pro případ, kdy firma předpokládá, že podobný software bude v budoucnu opět zpracovávat (viz kapitola 3.2 Analogie).

V každém případě se dá konstatovat, že zkoumání řešitelských podkladů, tedy specifikace, dokumentace různých fází projektů a posléze kódu, vede k odhadu složitosti produktu. Dále si Vaníček [Vaníček 2004] klade otázku, jaký je vztah mezi složitostí a náklady, respektive složitostí, počtem řešitelů a časem, který je na realizaci projektu potřeba. Pro jednoduchost předpokládá, že složitost je dána počtem řádků zdrojového kódu. Ve skutečnosti situace samozřejmě tak jednoduchá není.

Dále Vaníček [Vaníček 2004] předpokládá, že jiné faktory, které složitost ovlivňují (např. složitost řídících konstrukcí v kódu či úroveň programovacího jazyka nebo nástroje, ve kterém se implementace provádí), se již podařilo zohlednit tím, že byl rozsah kódu znásoben vhodnou konstantou a že v modifikovaném počtu řádků zdrojového kódu, který bude východiskem pro další úvahy, jsou již tyto korekce obsaženy.

Ukazuje se, že vztah mezi rozsahem softwaru a náklady na jeho vývoj není zdaleka lineární. U softwaru vyvíjeného pro prvé počítače v padesátých a šedesátých letech, kdy neexistovala žádná obecně přijatá pravidla a metodiky pro vývoj softwaru, se zdálo dokonce, že tento vztah je kvadratický. Vzrostl-li rozsah na dvojnásobek, vzrostla pracnost a tím i náklady čtyrnásobně [Vaníček 2004].

Jedním v současnosti nejznámějším odhadem růstu složitosti v závislosti na objemu je tak zvaný odhad COCOMO [COnstructive COst MOdel]. Prostřednictvím Function Points je vypočítán objem vytvářeného softwaru a následnou úlohou metody COCOMO je pomocí vstupních údajů o objemu navrhovaného IS predikovat pracnost.

Další důležitou otázkou je podle Vaníčka [Vaníček 2004] minimální doba, za kterou lze projekt realizovat (opět do etapy předání ověřeného systému uživateli či prvým uživatelům). Ta samozřejmě závisí na jeho celkové pracnosti a na počtu pracovníků, kteří se mu budou věnovat. Ani zde však neplatí jednoduchá přímá úměra, která se nabízí. U softwaru je nepřípustnost takovýchto zjednodušení ještě patrnější. Každá „dělba práce" totiž vyžaduje stanovit přesná rozhraní komponent, které vzniknou u různých autorů paralelně, a následnou kompletaci těchto komponent.

To bývá nesnadná práce, někdy dokonce obtížnější než vlastní realizace jednotlivé komponenty, a představuje jakousi „práci navíc". Bez dalších podrobností se spokojíme s předběžným odhadem Vaníčka [Vaníček 2004], který tvrdí, že práci na softwaru nelze „drobit do nekonečna". Existuje určitá mez, kdy je další dělení již velmi neekonomické a vede nejen k neúměrnému nárůstu nákladů, ale počínaje určitou úrovní dokonce k nárůstu doby potřebné pro realizaci projektu.

3.4.4.2 Výpočet složitosti metodou COCOMO

COCOMO je nejznámější a nejpopulárnější model pro odhad nákladů. Metoda COCOMO byla vyvinuta na přelomu sedmdesátých a osmdesátých let minulého století Barry Boehmem [Boehm 1981]. Tento první model se skládal z hierarchie tří modelů s rostoucím významem detailu – základní model COCOMO, střední model COCOMO a pokročilý model COCOMO. Tyto modely byly vyvinuty pro odhad projektů zákaznického softwaru a softwaru stavěného na míru. Rovnice pro výpočet základního a středního modelu jsou k dispozici v tabulce 20 a 21.

Naproti tomu se COCOMO 2.0 objevilo teprve nedávno kvůli neschopnosti původního modelu (COCOMO 1.1) přesně odhadovat objektově-orientovaný software, spirálovitě vytvořený software {created via spiral}, evoluční modely {evolutionary models} a software vyvinutý pro komerční účely na sklad {commercial-off-the-shelf software}.

Jaké jsou podle McGibbona základní rozdíly mezi modely COCOMO 1.1 a 2.0 [McGibbon 1997]:

- Zatímco COCOMO 1.1 musí znát buď rozsah softwaru v KSLOC {thousand source lines of code} nebo počet Function Points jako vstup, COCOMO 2.0 poskytuje rozdílný model odhadu pracnosti založený na vývojových etapách projektu. Cílem verze 2.0 je získat jako vstupy do modelu jen informace, které jsou k dispozici uživatelům v daný

čas. Jako výsledek, v průběhu nejranější návrhové etapy projektu označované jako Application Composition (Sestavení aplikace), model užívá Object Points (objektové body) odhadující složitost. Object Points jsou měřítkem softwarového rozsahu systému, odpovídající Function Points. V průběhu etapy Early Design (prvotní design), kdy máme minimální znalosti o projektovém rozsahu a projektovém týmu, jsou počty nevyrovnaných (unadjusted) Function Points užity jako vstup do modelu. Po výběru architektury, designu a způsobu vývoje, kdy začíná vlastní vývoj, se dostáváme do etapy, která je označována jako Post Architecture. V tomto bodě jsou vstupem modelu zdrojové řádky kódu.

- Zatímco COCOMO 1.1 poskytovalo bodové odhady pracnosti a časový plán, COCOMO 2.0 poskytuje pravděpodobný rozsah odhadů reprezentující jednu standardní odchylku blížící se nejpravděpodobnějšímu odhadu.
- COCOMO 2.0 je nastaveno pro znovupoužití softwaru a reinženýring, jsou zde užity automatické nástroje pro překlad existujícího softwaru. COCOMO 1.1 poskytovalo malou přizpůsobitelnost pro tyto faktory.
- COCOMO 2.0 také počítá s požadavky proměnlivosti ve svých odhadech.
- Exponent rozsahu ve vzorcích pracnosti v COCOMO 1.1 se mění s vývojovým módem. COCOMO 2.0 používá faktorů v pěti stupních pro generalizaci a nahrazení efektů vyvíjeného módu.

Tab. 19. Souhrnný přehled základních rozdílů metod COCOMO 1.1 a COCOMO 2.0

	COCOMO 1.1	**COCOMO 2.0**
1.	Znalost rozsahu SW v KSLOC {thousand source lines of codes} jako vstup.	Poskytuje rozdílný model odhadu pracnosti založený na vývojových etapách projektu.
2.	Poskytuje pouze bodové odhady pracnosti a časový plán.	Poskytuje pravděpodobný rozsah odhadů reprezentující jednu standardní odchylku blížící se k nejpravděpodobnějšímu odhadu.
3.	Malá přizpůsobitelnost v případě reinženýring a znovupoužitelnost.	Je nastavena pro znovupoužití SW a reinženýring.
4	Nebere v úvahu požadavky proměnlivosti v odhadech.	Počítá s požadavky proměnlivosti v odhadech.

Tab. 20. Přehled vzorců pro výpočet jednotlivých typů metody COCOMO převzato z [McGibbon 1997]

Model Name	**Effort Equation**	**Parameters**
Basic COCOMO	$Effort = a(KSLOC)^b$	Organic: a=2,4, b=1,05 Semidetached: a=3,0, b=1,12 Embedded: a=3,6, b=1,20
Intermediate COCOMO	$Effort = EAF \times a(KSLOC)^b$	EAF is the product of 15 cost driver attributes Organic: a=3,2, b=1,05 Semidetached: a=3,0, b=1,12 Embedded: a=2,8, b=1,20
COCOMO 2.0 Application Composition Model	$Effort = \frac{NOP}{PROD}$ $NOP = OP\frac{(100 - \%reuse]}{100}$	*OP* is Object Points *NOP* is New Object Points *PROD* is Productivity Rate *PROD* = 4 – very low; 7 – low; 13 – normal; 25 – high; 50 – very high
COCOMO 2.0 Early Design and Post Architecture Model	$Effort = \frac{ASLOC(\frac{AT}{100})}{ATPROD} + a[size(1 + \frac{BRAK}{100})]^b \pi EM_i$ $b = 1{,}01 + \frac{1}{100}\sum SF_j$ $Size = KSLOC + KASLOC(\frac{100 - AT}{100})AAM$	a = 2,5 SF_j = scale factor EM_i = effort multiplier *BRAK* = percentagecode discarded due to requirements volatility *ASLOC* = size of adapted components *AT* = percent of components

Model Name	Effort Equation	Parameters
		adapted *ATPROD* = Automatic Translation Productivity *AAM* = Adaptation Adjustment Multiplier

Tab. 21. Vzorce pro výpočet časového plánu metody COCOMO převzato z [McGibbon 1997]

Model Name	Schedule Equation	Parameters
Basic and Intermediate COCOMO	$Schedule = 2{,}5Effort^{c}$	Organic: c = 0,38 Semidetached c = 0,35 Embedded c = 0,32
COCOMO 2.0	$Schedule = c[Effort^{0{,}33+0{,}2(b-1{,}01)}]\frac{SCED\%}{100}$ $b = 1{,}01 + \frac{1}{100}\sum SF_j$	c = 3, 0 *SFj* = scale factor *SCED%* = schedule compression/ expansion parameter

3.4.4.2.1 Metoda COCOMO 1.1

Původní verzi odhadu COCOMO pro pracnost vývoje softwaru publikoval Boehm [Boehm 1981] na počátku osmdesátých let minulého století nejprve jako cíl, ke kterému se měl vývoj softwaru blížit při dodržení zásad softwarového inženýrství a využívání moderních metodik tvorby programů v imperativním prostředí. Zkušenosti z dobře řízených projektů, kde byly tyto zásady dodrženy, ukazují, že s výjimkou velmi specifických systémů pracujících v reálném čase, u kterých je překročení časového limitu kritické (například zbraňové systémy), se dnes podařilo těmto odhadům přiblížit nebo jich dosáhnout. Některé vysoce specifické problémy se však těmto odhadům stále vymykají.

Proměnné metody COCOMO 1.1 se obvykle označují:

- *E*……celková pracnost projektu uváděná v člověkoměsících za předpokladu, že se řešitel vlastní práci na projektu věnuje plně asi 150 hodin měsíčně .
- *T*……minimální „rozumná“ doba realizace projektu v měsících práce. Tím se rozumí doba, za kterou lze systém realizovat, aniž by spěch velmi významně zvýšil náklady na vývoj.
- *V*……modifikovaný počet tisíců zdrojových řádků kódu tvořícího softwarové řešení. Modifikace by měla brát ohled na složitost toku řízení uvnitř modulů a složitost komunikace mezi moduly, tedy na hodnoty měr složitosti programování v malém i ve velkém.

Potom platí odhady

$$E = a \cdot V^{b},$$

$$T = c \cdot E^{d},$$

kde *a, b, c, d* jsou empiricky zjišťované konstanty.

Oba základní vzorce metody COCOMO 1.1 lze kombinovat dosazováním proměnných z prvního vzorce do vzorce druhého a naopak. Tato možnost je demonstrována na proměnné *E*, kterou dosadíme do druhého vzorce. Ze vzorce prvého dostaneme vzorec pro přímý výpočet *T* na základě *V*. Dostáváme

$$T = p \cdot V^{q},$$

kde hodnoty konstant *p* a *q* závisejí na zvolených hodnotách *a, b, c* a *d* vztahy

$$p = c \cdot a^{d}, q = b \cdot d$$

Základní model COCOMO 1.1 [Boehm 1981] rozlišuje tři úrovně projektů zvané „módy“, které jsou charakterizovány takto:

- Tak zvaný „organický mód“, při kterém relativně malý softwarový tým pracuje na známé aplikaci, ve které se užívají běžné algoritmy. Pracuje v „domácím prostředí“ na stabilním hardwaru. Pro tento mód doporučuje Boehm tuto volbu konstant:

a = 3,2; *b* =1,05; *c* = 2,5; *d* = 0;38, tedy *p* = 3,89; *q* = 0,40.

- Tak zvaný „přechodný mód“, který je přechodem mezi snadným – „organickým“ módem a obtížným „vázaným módem“ a týká se například projektů se zvýšenými nároky na komunikaci. Pro něj se doporučuje volit konstanty takto:

  $a = 3{,}0$; $b = 1{,}12$; $c = 2{,}5$; $d = 0{,}35$, tedy $p = 3{,}69$; $q = 0{,}39$.

- Tak zvaný „vázaný mód“ pro velké projekty, řešené za obtížných podmínek, při častých změnách požadavků v průběhu řešení, na nestabilním hardwaru, pod nestabilním operačním systémem, s vysokými nároky na interakce a s požadavky na časovou odezvu. Pro něj jsou doporučené konstanty:

  $a = 2{,}8$; $b = 1{,}20$; $c = 2{,}5$; $d = 0{,}32$ tedy $p = 3{,}48$; $q = 0{,}38$.

Boehmův návrh konstant je na prvý pohled poněkud zvláštní a patrně diskutabilní. U krátkých programů (například délky tisíce řádků) dává totiž pro program v obtížném „vázaném“ módu nižší pracnost, než pro program stejného objemu, realizovaný v snazším „organickém“ módu [Vaníček 2004].

Tento paradox lze podle Vaníčka [Vaníček 2004] odstranit drobnou opravou Boehmova doporučení tak, že konstantu a budeme volit vždy stejnou, například rovnou 3, bez rozdílu módu. Pak u „krátkých“ programů na módu nezáleží, pro „rozsáhlé“ programy je růst pracnosti v závislosti na objemu u složitějších módů strmější. I to, že u složitějších softwarových produktů vychází minimální rozumný čas pro jejich realizaci vždy kratší než u produktů v jednodušším módu, je více než podivné.

Vaníček [Vaníček 2004] dochází při svých zkoumání původně navržených koeficientů k odlišným závěrům, z tohoto důvodu nelze rozdělení projektů do módu chápat dogmaticky. Alternativa, kterou Vaníček představil, pracuje ve vzorci COCOMO s následujícími konstantami. Je otázkou diskuse, zda je možné ji považovat za „konkurenční“ model k Boehmovým třem módům. Domnívám se, že do jisté míry ano.

Vaníček [Vaníček 2004] doporučuje zvolit konstantně $a = 3$. To odpovídá produktivitě práce cca 17 řádků zdrojového kódu „malého“ programu za den, ovšem včetně návrhu, dokumentace,

zkoušení atd. Dále volit vždy $c = 2.5$. Hodnotu konstant b a d pak volit v závislosti na tak zvané třídě programových systémů například takto:

- Třída 1: Ekonomické úlohy a zpracování hromadných dat s jednoduchými algoritmy zpracování:

$b \in \langle 1{,}05; 1{,}10\rangle$, $d \in \langle 0{,}3; 0{,}35\rangle$, tedy $p \in \langle 3{,}48; 3{,}67\rangle$, $q \in \langle 0.31; 039\rangle$.

- Třída 2: Ekonomické a vědeckotechnické úlohy se složitými algoritmy zpracování, avšak bez interakce, a úlohy pracující s rozsáhlými nedistribuovanými bázemi dat, bez paralelního přístupu:

$b \in \langle 1{,}08; 1{,}13\rangle$, $d \in \langle 0{,}34; 0{,}36\rangle$, tedy $p \in \langle 3{,}63; 3{,}71\rangle$,$q \in \langle 0{,}37; 0{,}41\rangle$.

- Třída 3: Informační systémy budované nad stabilním operačním systémem, pracující s distribuovanými bázemi dat a s možností interakčního paralelního přístupu více uživatelů paralelně a s požadavky na paralelní změny dat z více míst, avšak bez kritických požadavků na odezvy v reálném čase:

$b \in \langle 1{,}12; 1{,}17\rangle$, $d \in \langle 0{,}35; 0{,}38\rangle$, tedy $p \in \langle 3{,}67; 3{,}80\rangle$, $q \in \langle 0{,}39; 0{,}45\rangle$.

- Třída 4: Informační systémy s požadavky pro třídu 3 a navíc s požadovanou „měkkou“ dobou odezvy (požaduje se například zpravidla odezva do jedné sekundy, překročení časového limitu však není fatální chybou, pokud k němu dochází pouze zřídka). Systémy obsluhující paralelní procesy a virtuální zdroje. Příkladem může být jádro a řídící části operačního systému:

$b \in \langle 1{,}16; 1{,}22\rangle$, $d \in \langle 0{,}36; 0{,}40\rangle$, tedy $p \in \langle 3{,}71; 3{,}88\rangle$, $q \in \langle 0{,}41; 0{,}49\rangle$.

- Třída 5: Komplikované výpočty s paralelními procesy a s krátkou „tvrdou“ - absolutní dobou odezvy. S velkými soubory dat, kde každé překročení doby odezvy znamená nepřípustnou havárii a nároky na bezporuchovost jsou enormní. Vyžaduje se matematický důkaz korektnosti algoritmů, nezávislé dvojí programové

zpracování a obdobné mimořádné prostředky k zabezpečení spolehlivosti (řídicí systémy technologií, řízení kosmických letů, zbraňové systémy apod.)

Tento způsob odhadu nelze užít, jediné možné odhady jsou z pracnosti analogických systémů (viz kapitola 3.2 Analogie).

Při výše uvedeném nastavení koeficientů již podle výpočtů Vaníčka [Vaníček 2004] k paradoxům nedochází. Složitější projekty jsou pracnější a trvají déle. Potřebná pracnost i minimální rozumný čas na jejich realizaci roste s objemem potřebného softwaru strměji než u úloh jednoduchých.

Je nutné podotknout, že Vaníčkovi odhady vycházejí z jeho zkušeností s vývojem rozsáhlých softwarových systémů v sedmdesátých a osmdesátých letech minulého století a ze skutečných dat o těchto systémech a pracnosti jejich realizace. V současnosti jsou pro návrh systémů k dispozici pokročilejší metody, nástroje a hardware, pro které je moderní software navrhován, se od tehdejšího liší výrazně. Proto nelze v současnosti přisuzovat jeho alternativě koeficientů COCOMO jiný význam než metodický. Ten spočívá v tom, že vyladění koeficientů modifikujících jakýkoliv parametrizovatelný odhad musí být takové, aby nevedlo k paradoxním výsledkům pro žádný typ softwaru.

Přitom vymezení jednotlivých tříd je třeba považovat pouze za vymezení příkladem. Vaníček [Vaníček 2004] se odkazuje pouze na typy systému, u kterých měl možnost se řízení prací účastnit a pracnost sledovat nebo aspoň o nich získat potřebné informace. Pro systémy, jejichž charakteristika nebyla do předchozího výčtu zahrnuta, nebo které je obtížné zařadit, je třeba nalézt srovnatelně obtížný systém, který je zde uveden a zvolit hodnotu podle něj.

Výběr konkrétní hodnoty z daného oboru by se měl řídit zkušenostmi řešitelů v dané oblasti a podmínkami pro práci řešitelského týmu. Systémy třídy 5 se odhadu vymykají. Je nutné je hodnotit individuálně. Pracnost zde může být až desetinásobně vyšší.

Otázky vzbuzuje stále volba konstanty *d*. V původním Boehmově [Boehm 1981] návrhu její negativní závislost na módu či třídě systémů kompenzuje minimální čas potřebný na řešení tak, že u náročnějších systémů jej zkracuje. Tím do jisté míry vyrovnává zvýšení celkové pracnosti způsobené vyšší hodnotou parametru *b*, který ovlivní velikost pracnosti *E* směrem nahoru. Důvody mohou být ty, že složité úkoly je nutné svěřit erudovaným a vzdělaným řešitelům. Ti však bývají zpravidla i pilnější a výkonnější. Složitější úkol řešený špičkovými řešiteli pak

může být vyřešen dříve než snazší, stejného rozsahu, řešený průměrnými. Z toho patrně vychází experimentální ověření Boehmových vzorců.

V praxi se stává, že pro náročné úkoly zpravidla není možné získat dostatek kvalifikovaných a erudovaných řešitelů. Nutnost do práce zapojit i méně vhodné pracovníky vede zpravidla k tomu, že jejich výkonnost je nižší, produkt má více chyb, které je nutno opravovat, a projekt se tak zpožďuje. Erudice a píle špičkových členů týmu se pak do značné míry vyčerpá na podporu nováčků. To zpochybňuje oprávněnost Boehmovy korekce minimální doby řešení a podporuje spíše koeficienty navržené Vaníčkem a potvrzené jeho zkušenostmi.

Kompromisem může být podle Vaníčka [Vaníček 2004] volba konstanty *d* pevně, bez ohledu na mód či třídu řešených systémů, a to na hodnotě 0,38, navržené Boehmem pro organický mód. Zdá se, že takto získané hodnoty minimální doby řešení se více přibližují realitě a výpočet je jednodušší. Zbývá tedy jediná proměnná hodnota exponentu *b*, jejíž hodnota závisí na třídě obtížnosti softwaru a je vybírána z intervalu <1,05; 1,22>. Ostatní parametry vzorců COCOMO 1.1 lze považovat za konstanty: $a = 3$; $c = 2{,}5$; $d = 0{,}38$.

Korekce odhadů metody COCOMO 1.1

V odhadech COCOMO 1.1 nejsou zahrnuty některé faktory, které mohou podstatně ovlivnit celkovou pracnost *E* a následně i minimální rozumnou dobu řešení *T*. Jde především o faktory charakterizující nároky na produkt a jeho jakost, faktory výpočetní techniky, která je pro vývoj k dispozici, faktory popisující kvalifikaci předpokládaných řešitelů a faktory popisující řízení projektu a softwarové nástroje, které mají řešitelé k dispozici. Proto byla navržena metoda, jak odhad korigovat. Metoda spočívá (obdobně jako u druhého faktoru ve vzorcích pro odhad pomocí funkčních jednic) v tom, že klasifikujeme každý faktor v ordinální stupnici:

velmi nízký – nízký – normální – vysoký – velmi vysoký

a v závislosti na této klasifikaci odhad COCOMO 1.1 násobíme hodnotou blízkou k jedné. Přitom, je-li faktor hodnocen stupněm normální, násobíme vždy číslem jedna (hodnota odhadu se nemění). Při nízkém hodnocení bude činitel, kterým odhad násobíme, menší než 1, při vysokém větší než 1.

Před tím, než uvedeme jeden z mnoha možných návrhů faktorů, které lze brát podle Vaníčka [Vaníček 2004] v úvahu, a jejich možné hodnocení, je účelné upozornit, že obdobné faktory se

vyskytují již v metodě funkčních jednic. Pokud byl odhad *V* objemu produktu stanoven touto metodou a na faktor byl již zde vzat ohled, nelze jej zahrnout znova. Tím by se jeho vliv započítal nesprávně „dvakrát“ (přesněji, uplatnila by se jeho druhá mocnina).

Právě tak, pokud se vycházelo ze skutečného zdrojového kódu, mohl být tento nebo analogický faktor uplatněn již při přechodu od skutečného počtu řádků k modifikovanému počtu řádků a opět mohlo dojít k násobnému započtení. V obou těchto případech je nutné při korekci odhadu COCOMO tento faktor již znova nehodnotit a příslušný multiplikativní koeficient stanovit rovný jedné. Korekce odhadu COCOMO 1.1 tak skýtá pouze možnost uplatnit ty faktory, které se nám zatím v hodnocení nikde „udat“ nepodařilo.

Následující faktory a jejich hodnocení je třeba považovat spíše za ukázku možné volby, než za definitivní návod. Lze uvažovat například faktory:

- RELY – Požadovaná spolehlivost (bezporuchovost a udržovatelnost) produktu.
- DATA – Rozsah obsluhovaných dat (rozměr databáze a složitost vztahů v ní).
- CPLX – Složitost systému (ty faktory, které se dosud v hodnocení nepodařilo uplatnit).
- TIME – Požadavky na účinnost v oblasti doby potřebné pro provedení výpočtu.
- STOR – Požadavky na účinnost v oblasti úspory paměti počítače.
- VIRT – Stabilita výpočetního prostředí (hardware), na kterém probíhá vývoj.
- TURN – Rychlost odezvy počítače, na kterém probíhá vývoj.
- ACAP – Znalosti a zkušenosti vedoucího analytika projektu.
- AEXP – Úroveň znalostí řešitelů o aplikační oblasti, které se produkt týká.
- PCAP – Profesionální kvalifikace a zkušenosti výkonných programátorů v týmu.
- VEXP – Znalosti virtuálního počítače (operačního systému a softwarového prostředí).
- LEXP – Znalosti programového jazyka, v kterém je systém implementován.
- MODP – Užití moderních programovacích metod (strukturované, modulární, objektové).
- TOOL – Užití nástrojů pro podporu vývoje (CASE TOOLS).
- SCED – Zda je vývoj řádně plánován a termíny nejsou napjaté.

Možné hodnocení může být opět předmětem diskuse. Vzhledem k tomu, že východiskem je pouze ordinální stupnice ocenění vlivu jednotlivých faktorů na pracnost, je jakékoliv jejich užití jako multiplikativních faktorů ve výpočtech z hlediska teorie měření sporné. Nicméně v konkrétních případech nám může umožnit se přiblížit realitě. Jako vždy u podobných „dodatečných korekcí“ samozřejmě hrozí možnost účelového zneužití směrem k našim

apriorním představám, které mohou být pouhým předsudkem. Sporné je i to, zda lze váhu jednotlivých faktorů stanovit odpovědně.

Ve skutečnosti mohou být požadavky na jeden vybraný faktor (například bezporuchovost) tak enormní, že jediný faktor ovlivní pracnost až na několikanásobek. Na druhé straně, kdybychom interval kolem hodnoty 1 rozšířili tak, aby bylo možné tuto skutečnost postihnout, připustili bychom tak vysokou kumulaci několika korekcí, která by původní odhad, založený na datech, která jsou méně subjektivní než hodnocení faktorů, znehodnotili úplně. Jedna z možných (i když také problematických) voleb multiplikativních koeficientů je navržena v následující tabulce 22 [Vaníček 2004].

Tab. 22. Cost drivers pro korekci odhadů metody COCOMO

Faktor f_i	**Velmi nízký**	**Nízký**	**Normální**	**Vysoký**	**Velmi vysoký**
RELY	0,65	0,85	1	1,1	1,4
DATA	0,8	0,94	1	1,08	1,16
CPLX	0,7	0,85	1	1,15	1,3
TIME	1	1	1	1,1	1,3
STORE	1	1	1	1,06	1,25
VIRT	1,3	1,15	1	0,9	0,9
TURN	1,2	1,1	1	0,9	0,9
ACAP	1,5	1,2	1	0,95	0,9
AEXP	1,3	1,12	1	0,85	0,7
PCAP	1,4	1,2	1	0,85	0,7
VEXP	1,2	1,1	1	0,9	0,9
LEXP	1,2	1,1	1	0,9	0,9
MODP *)	1,45	1,2	1	0,85	0,7
TOOL	1,3	1,1	1	0,85	0,7
SCED	1,2	1,05	1	1,05	1,1

*) Navržené rozsahy vlivu faktoru se týkají „velkých“ projektů (od cca 1/2 miliónu řádek zdrojového kódu), pro „menší“ projekty bude vliv faktoru patrně nižší.

Modifikovaný vzorec pracnosti *E* v člověkoměsících vypočtené metodou COCOMO z objemu *V* má pak tvar

$$E = 3 \times V^b \times \prod_{j=1}^{15} f_i \ ,$$

kde f_j je hodnota *j*-tého faktoru.

3.4.4.2.2 Metoda COCOMO 2.0

COCOMO 2.0 Application Composition model (viz tabulka 20, 3. řádek) užívá podle McGibonna [McGibbon 1997] pro provedení odhadu tzv. Object Points. Model dále podporuje využití integrovaných CASE nástrojů. Objekty zahrnují obrazovky, reporty a moduly v programovacích jazycích třetí generace. Není pravidlem, že Object Points jsou nutně navázány na objekty v objektově orientovaném programování.

Pomocí metody je odhadován počet nezpracovaných objektů, stejně tak i složitost každého objektu. Na základě odhadu těchto hodnot je dále vypočítán vážený součet objektových bodů {Object-Point count}. Následně jsou odhadnuty procenta užití a předpokládaná produktivnost. Na základě těchto výstupů, jsme schopni spočítat i odhad pracnosti zkoumaného projektu (viz tabulka 20. řádek 3.).

Function Points v životním cyklu jsou pravděpodobně jednodušeji odhadnutelné než prvotní KSLOC. Jednoduchým důvodem pro toto tvrzení je, že Function Points utvářejí rozsah vstupů pro model COCOMO 2.0 Early Design Model. Nevyrovnané Function Points jsou převedeny na ekvivalent počtu zdrojových řádků (SLOC) použitím tabulky 23.

Tab. 23. Konverze UFP (nevyrovnaný počet Function Points) na SLOC [McGibbon 1997]

Programovací jazyk	**SLOC/ UFP**
Ada	71
AI Shell	49
APL	32
Assembley	320
Assembley (Macro)	213
ANSI/ Quick/ Turbo Basic	64
Basic – Compiled	91
Basic – Interpreted	128
C	128
C++	29
ANSI COBOL 85	91
Fortran 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91

COCOMO 2.0 podle McGibbona bere ve svém odhadu pracnosti v úvahu i efekty plynoucí z reinženýringu. Pokud totiž projekt zahrnuje automatické transakce, pak nejprve musíme odhadovat:

- Produktivitu automatického přeložení {ATPROD – automatic translation produktivity}, odhadnutá z předchozích odhadů vývoje,
- Rozsah, v tisících řádcích zdrojového kódu, nepřeloženého kódu a kódu pro překlad (KASLOC) projektu,
- Procento komponent vyvinutých z reinženýringu softwaru (ADAPT),

- Procento komponent, které jsou automaticky překládány (AT).

Zatímco v COCOMO 1.1, rovnice pracnosti je vyrovnávána 15 cost drivers atributů (viz tabulka 22), COCOMO 2.0 definuje sedm cost drivers pro odhad prostřednictvím modelu Early Design:

- Lidské kapacity,
- Produktová spolehlivost a složitost,
- Požadovaná znuvupoužitelnost,
- Složitost platformy,
- Zkušenosti týmu,
- Vybavenost,
- Omezení časovým plánem.

Všechny z těchto multiplikátorů pracnosti nemusí být nutně zahrnuty mezi několik multiplikátorů modelů COCOMO 2.0 Early Design and Post Architecture.

Softwarové modely COCOMO 1.1 se navrhují jako zobrazení klesajících výnosů z rozsahu {decreasing returns to scale}. Klesající výnosy {decreasing returns} jsou v rovnicích pracnosti zachyceny prostřednictvím exponentu SLOC, který nabývá hodnoty větší nebo rovny jedna. Tento exponent se mění v rámci třech COCOMO 1.1 vývojových módů (organický, vázaný a přechodný mód). COCOMO 2.0 nedělí projekty jednoznačně podle vyvinutých módů, ale pracuje s tzv. pěti faktorovou stupnicí [McGibbon 1997]:

- Konfliktnost {Precedentedness},
- Flexibilita vývoje {Development flexibility},
- Rozhodnutí o architektuře/ riziku {Architecture/ risk resolution},
- Soudržnost týmu {Team cohesion},
- Zralost organizačního procesu {Organization process maturity}.

Jakmile je nastavena architektura, může být použit model COCOMO 2.0 Post-Architecture. Použití softwaru pro reinženýringové a automatické překlady se projeví v rovnicích Early Design (ASLOC, AT, ATPROD a AAM). Podstatná změna {BRAK – breakage} nebo procento kódu vyhozeného kvůli změnovým požadavkům je v 2.0 vzato v úvahu. Znovupoužití softwaru {RUF – reused SW} se projeví v rovnici pracnosti při nastavení rozsahu u přizpůsobení vyrovnávacího multiplikátoru {AAM – adaptation adjustment multiplier}.

Tento multiplikátor je vypočítán z procentních odhadů upraveného návrhu {DM – design modified}, procenta upraveného kódu {CM – code modified}, integrace úpravy pracnosti {IM – integration effort modification}, porozumění softwaru {SU – software understanding} a ohodnocení a přizpůsobení {AA – assesment and assimilation}. Sedmnáct multiplikátorů pracnosti je definováno pro model Early Design and Post-Architecture shrnutý do čtyřech kategorií:

- Produktové faktory,
- Faktory platformy,
- Osobní faktory,
- Projektové faktory.

Tyto čtyři kategorie se vzájemně podobají čtyřem kategoriím modelu COCOMO 1.1 – atributy produktu, výpočetní atributy, lidské atributy a projektové atributy. Většina ze sedmnácti faktorů verze 2.0 jsou podobné patnácti faktorům verze 1.1. Nové faktory představené ve verzi 2.0 zahrnují požadovanou znovupoužitelnost, zkušenost s platformami, zkušenosti s jazykem a nástroji, spojitost s lidským faktorem a obratem. Některé jako např. užití moderních programovacích technik nebo zkušenosti s virtuálním strojem byly multiplikátorem pracnosti v COCOMO 1.1 a jsou převzaty i do COCOMO 2.0.

Jednoduchý časový plán vývoje pracnosti je definován pro všechny tři modely COCOMO 2.0 v tabulce 21.

3.4.4.3 Softwarové nástroje pro metodu COCOMO 2.0.

Pro výpočet složitosti se u metody COCOMO 2.0 používá např.softwarový nástroj COSTAR 7.0 [Costar 2005], který umožní jednoduchý a rychlý odhad složitosti projektu bez hlubší znalosti dané metody.

Podobně i CALICO 7.0 [Calcio 2005] umožňuje pracovat až se 4 000 řádky kódu definující COCOMO model, dále umožňuje přidávat nejdůležitější náklady, upravovat konstanty definující úsilí, měnit porovnávající základnu, atd.

Další možností je USC COCOMO 2.01999.0 Software, který je k dispozici na internetových stránkách zabývajících se metodou COCOMO 2.0.

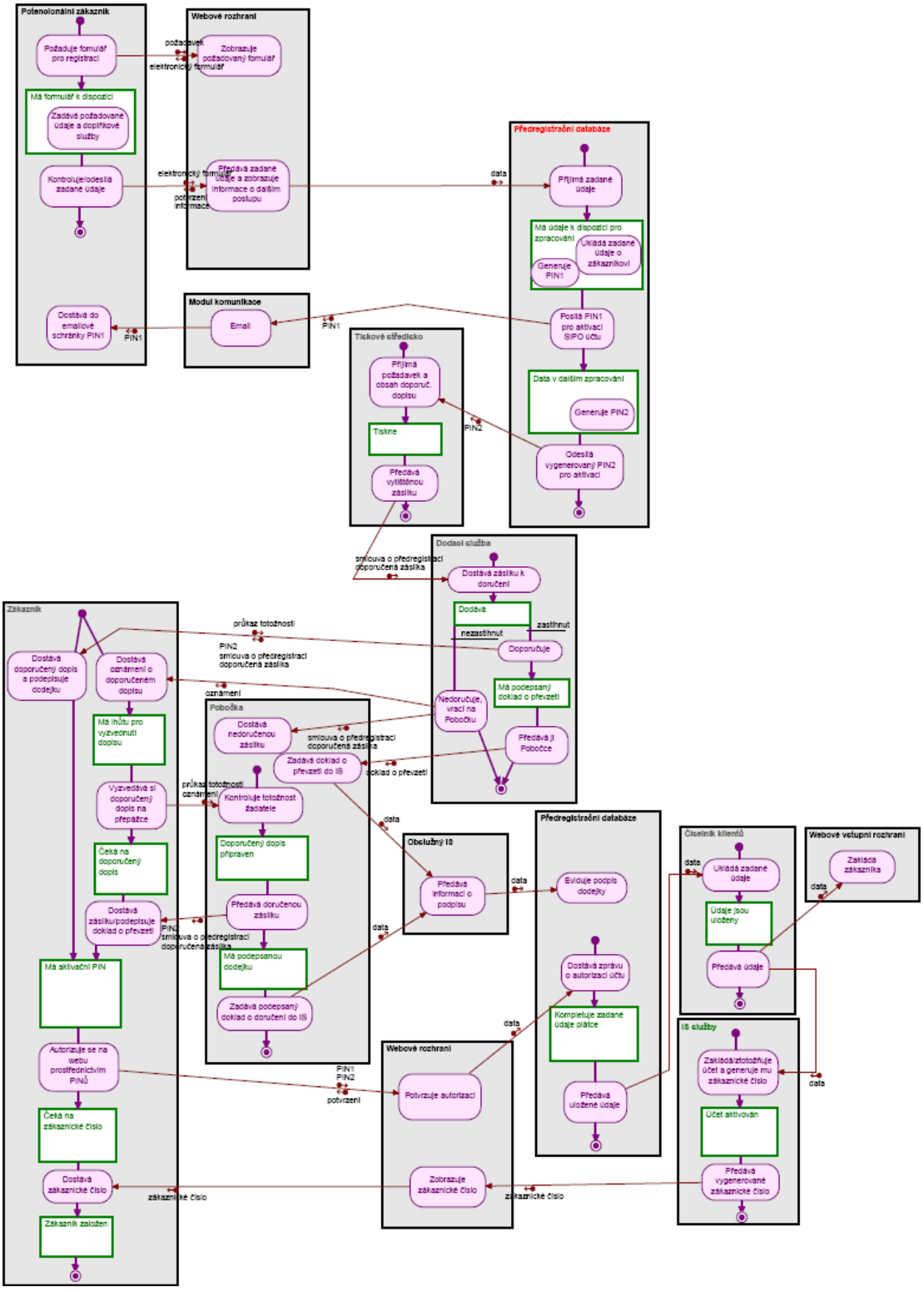
U uživatelů, kteří nechtějí používat konkrétní softwarové nástroje existuje možnost si pro metodu COCOMO 2.0, stejně jako pro všechny předcházejících metody vytvořit vlastní nástroj pro výpočet složitosti např. v Excelu.

3.4.5 Úvod do metody BORM

Metodu BORM (Business and Object Relation Modelling) lze pokládat za původní českou metodu určenou pro analýzu a návrh IS.

Metoda BORM je vyvíjena postupně od roku 1993. Od počátku byla orientována na podporu tvorby objektově orientovaných softwarových systémů založených na čistých objektově orientovaných programovacích jazycích a vývojových prostředích, jakými jsou například prostředí Smalltalku a nerelační objektové databáze. BORM je možné využít nejen ve tvorbě softwaru, ale i k analýze požadavků na projektovaný systém a na modelování podnikových procesů.

V současné době je vývoj metody podporován firmou Deloitte&Touche Czech Republic and Central Europe, kde je tato metoda také prakticky používána.



Obr. 18. Ukázka BORM procesního diagramu z programu CraftCase 1.5.9

Metoda BORM je detailněji popsána v příloze 7.1 této práce. V případě zájmu o hlubší porozumění metodě BORM lze doporučit publikaci [Carda, Merunka, Polák 2003]. Kniha obsahuje podrobný popis metody BORM a ukázky jejího praktického uplatnění.

3.5 Zhodnocení technik odhadu složitosti vývoje IS

Autor této práce v průběhu studia představených metod došel následujícím hodnocením, které obsahují přednosti či nedostatky metod včetně jeho pohledu na vhodnost či nevhodnost použití vybrané metody na projekt tvorby IS.

Autor práce by chtěl upozornit na předcházející závěrečné podkapitoly představující softwarové nástroje pro praktickou aplikaci popsaných metod. Tyto nástroje mají hlavní přínos v tom, že umožňují aplikaci metod odhadu složitosti i uživatelům bez hlubší znalosti výpočetního postupu metody. Odhadce pouze zadá vstupní parametry projektu, výstupem je celková složitost projektu. S ní se může buď spokojit nebo s ní dále pracovat tak, že upraví vstupní parametry a výpočet provede znovu.

3.5.1 Zhodnocení metody Function Points

Za společný rys metod funkčních jednic je považována vlastnost, na jejímž základě Function Points odhadují složitost jako součin dvou faktorů. Prvý faktor je založen na funkcích, které jsou od produktu vyžadovány, druhý na podmínkách, které pro vytvoření produktu objektivně existují.

Vědeckou veřejností je původní metoda Function Points doporučována pro výpočet složitosti projektů, které jsou více výpočetně složité, tzn. obsahují větší množství logických datových souborů než algoritmů. V případě, že by IS obsahoval přibližně stejné množství algoritmů a logických datových souborů, lze aplikovat metodu Function Points i Feature Points.

V průběhu doby vzniklo několik verzí metody Function Points, které se pokouší zdokonalovat původní přístup metody IBM Function Points. Za nejjednodušeji použitelnou modifikací IBM Function Points považuje autor této práce metodu Back-fire Function Points (kapitola 3.4.1.4), která umožňuje jednoduchý a rychlý přepočet řádků zdrojového kódu na jednu funkční jednici. I přes tuto jednoduchost by si měl odhadce uvědomit, že tyto hodnoty je nutné brát jako

průměrné a stejně tak s nimi nakládat. Je totiž zřejmé, že byly zjišťovány na různých projektech, jenž byly realizovány za odlišných podmínek.

IFPUG Function Points je další modifikací původní metody. IFPUG se snaží prostřednictvím uživatelského pohledu vytvořit formální popis obchodních požadavků, které vývojoví pracovníci musí přeložit do programovacího jazyka.

To, že metoda IFPUG brala v úvahu při vývoji IS pouze uživatelský pohled, vedlo autory metody COSMICS Function Points k tomu, aby vyvinuli metodu novou. Právě metoda COSMICS – FPP se snaží postavit odhad složitosti na tzv. „pohledu měření“, který nabízí odhadci možnost výběru pohledu měření. Buď použije stejně jako u IFPUG měření z pohledu uživatele nebo nový tzv. vývojový pohled COSMICS – FPP (detaily viz 3.4.1.3.1 Koncept metody COSMIC – FPP).

Cílem vývojového pohledu bylo postihnout i projekty s odlišnou technologií, jejichž podstatná funkcionalita je uživatelskému pohledu skryta. Přínosem užití vývojářského pohledu v kombinaci s COSMIC – FPP je to, že všechny funkce alokované (rovnoměrně rozdělené) v rámci softwaru jsou zahrnuty do měřené hodnoty.

3.5.2 Zhodnocení metody SPR Function/ Feature Points

Metoda SPR nejprve vznikla jako další varianta metody IBM Function Points, při její aplikaci na IS vyjmenované v kapitole 3.4.2.1.1 Koncept metody SRP Feature Points se zjistilo, že její výstupy představují velmi diskutabilní výsledky. Diskutabilní jsou proto, že se jedná o vysoce algoritmicky složité IS, které jsou zároveň rozptýleny do vstupů a výstupů. Metoda Function Points nedokáže tuto algoritmickou složitost dostatečně přesně pracovat (viz kapitola 3.5.1 Zhodnocení metody Function Points).

Z výše uvedených důvodů byla vyvinuta metoda SPR Feature Points. Její výpočet je doplněn o tzv. šestý parametr – algoritmus, který je rozšířením původních pěti parametrů metody IBM Function Points. Výpočet metody SPR Feature Points je jednodušší a poskytuje rychlejší odhad složitosti budoucího IS v porovnání s Function Points. Metoda je doporučena pro odhad složitosti algoritmicky složitějších IS.

3.5.3 Zhodnocení metody UCP

Ve své době představovala metoda UCP převratný způsob výpočtu složitosti vývoje IS. Zcela nepochybně lze konstatovat, že v současné době se stále jedná o široce použitelnou metodu, která umožňuje provádět odhady složitosti různých druhů IS.

Poslední dobou se začínají v návrzích IS používat metody, které nemodelují IS jako „uzavřený celek“, ale jako komplexní přehled funkcí, scénářů a diagramů obsahující všechny procesy, které nově vznikající IS ovlivňuje. V tomto směru se autor této práce domnívá, že metoda UCP zaostává.

Nevýhodou UCP vůči těmto metodám je, že přílišná míra zkreslení modelovaného Use Case může způsobit následující situaci. Use Case diagram nedokáže do dostatečného detailu postihnout všechny procesy, které bude vznikající IS používat včetně zachycení aktivit a stavů, do kterých se při práci s IS dostanou zapojení aktoři.

Příkladem může být situace, kdy se až v průběhu vývoje IS přijde na okolnosti, které z důvodu lajdáctví při tvorbě Use Case modelů byly opomenuty a přitom významným důvodem mohou ovlivnit celkovou pracnost IS. Toto ovšem tvrzení neplatí pouze pro tuto metodu. Naproti tomu lze konstatovat, že přístup metody může vést k úspoře nákladů na úvodní analýzu z toho důvodu, že není nutné provádět detailnější analýzu procesů.

Výsledky hodnocení metody UCP z této kapitoly vzal autor práce v úvahu při návrhu nové metody BORMp, která vychází z logiky metody UCP.

3.5.4 Zhodnocení metody COCOMO

Odhady COCOMO jsou založeny na snaze určit minimální dobu, za kterou lze ještě realizaci projektu považovat za rentabilní. Pokud by se vývojářská firma snažila o rychlejší realizaci, narážela by již na hranici nereálného očekávání.

Jde-li o odhad pracnosti vývoje softwaru, metoda COCOMO je nastavena jako cíl, ke kterému by se měl vývoj softwaru blížit při dodržení zásad softwarového inženýrství a využívání moderních metodik tvorby programů. Původní metoda COCOMO 1.1 se skládá z hierarchie tří modelů s rostoucím významem detailu – základní model COCOMO, střední model COCOMO a pokročilý model COCOMO. Tyto modely byly vyvinuty pro odhad projektů zákaznického

softwaru a softwaru stavěného na míru. Její podstatnou nevýhodou pro aplikaci v počáteční fázi vývoje softwaru je, že požaduje jako vstupní parametr odhad počtu zdrojových řádků vyvíjeného systému.

V reakci na tyto zjištění byla vyvinuta nová verze, tzv. COCOMO 2.0, která se snaží reagovat na vývoj v oblasti tvorby IS. Metoda COCOMO 2.0 je rozdělena na 3 modely – Application Composition, Early Design a Post Architectural Model a snaží se odstranit hlavní nevýhody starší verze. Především metoda COCOMO Application Composition zvyšuje využitelnost modelu COCOMO 2.0 i v počátečních fázích vývoje softwaru.

Metodu COCOMO považuje autor práce za nejznámější a nejpopulárnější model pro odhad nákladů vývoje IS.

3.5.5 Předpoklady pro vznik metody BORMp

Ddůležitou součástí odhadu složitosti v metodě BORM je procesní diagram, v němž je nutné nadefinovat kromě funkcí a scénářů též participanty a datové toky. Každý participant v rámci svého zapojení do procesu provádí aktivity a dostává se do stavů. Procesní diagramy jsou považovány za jeden z hlavních výstupů procesní analýzy IS. Procesní diagram tedy pro odhad složitosti poskytuje důležité vstupní údaje pro metodu BORMp, která je využívá pro svůj odhad pracnosti.

Důležitou částí procesního diagramu pro odhad složitosti je také participant. Participant reprezentuje v pojetí BORMu konkrétní jednotku modelované reality. Mezi participanty nepatří jen živé bytosti, ale i stroje, IS a další prvky přímo se účastnící procesu.

Přechody mezi stavy a aktivitami, které se vzájemně střídají, jsou doplněny komunikacemi spojující aktivity mezi zúčastněnými participanty. Pro zpřesnění modelu lze doplnit procesní diagram datovými toky, které zachycují objekty, jenž si participanti při vzájemných iteracích vyměňují. Může jít například o informační, finanční nebo materiálové toky (dokument, formulář, potvrzení, atd.). Přesné vysvětlení použitých pojmů obsahuje příloha 7.1 Charakteristika metody BORM.

K předpokladům popsaným v předcházejících odstavcích bylo přihlédnuto v průběhu vyvoje metody BORMp s tím, že postup výpočtu je postaven na konceptu metody UCP, jejíž identifikované nevýhody se snaží odstranit.

3.5.6 Závěrečné zhodnocení metod

Přestože výsledky v práci představených a zhodnocených metod jsou diskutabilní, dá se z jejich používání profitovat. Autor práce se domnívá, že techniky založené na modelu vyjmenované v kapitole 3.4 Odhady založené na modelu podporují široce uznávané postupy a jsou podle názoru autora této práce lépe obhajitelné před netechnicky zaměřenými manažery, kteří mají o realizaci projektu rozhodovat.

4 Návrh metody BORMp

4.1 Postup nastavení vah a koeficientů metody BORMp

Na základě diskuzí s kolegy na Katedře informačního inženýrství a Katedře operační a systémové analýzy byl jako nejvhodnější přístup pro stanovení koeficientů a vah metody BORMp vybrána kombinace expertního odhadu a metody analytického hierarchického rozkladu (AHP). Kombinace těchto dvou přístupů byla použita pro nastavení koeficientů a vah faktorů technického, prostředí a zákaznického pro navrhovanou metodu BORMp.

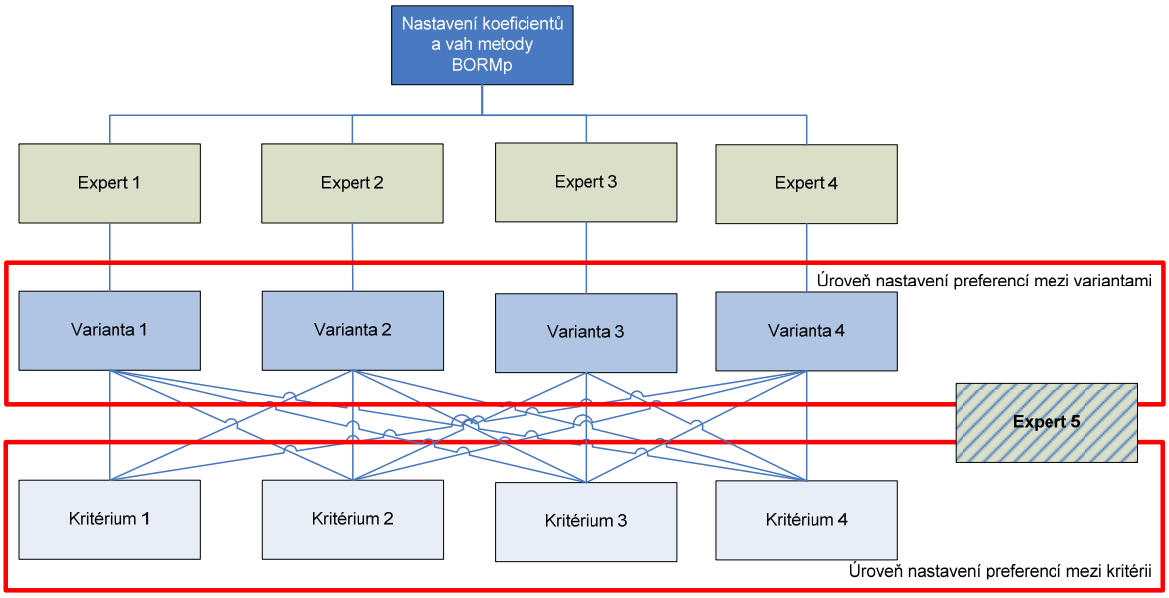
4.1.1 Praktická aplikace metody AHP pro nastavení vah a koeficientů metody BORMp

Nastavení koeficientů a vah probíhalo následujícím postupem. Nejprve došlo k oslovení čtyř vybraných IT expertů, jejichž úkolem bylo navrhnout koeficienty a váhy pro technický faktor, faktor prostředí a zákaznický faktor.

Vybraný expert musel splňovat minimálně tyto požadavky:

- znalost oblasti odhadu složitosti,
- znalost metody BORM,
- zkušenosti z praktické aplikace metody BORM.

V případě, že expert splnil stanovené podmínky, byl požádán o odhad. Odhady jednotlivých expertů byly dále shromážděny a pro výběr nejvhodnější varianty byla použita metoda AHP. Před vlastním výpočtem byly využity poznatky tzv. pátého experta, jehož úkolem bylo nastavení preferencí mezi kritérii a variantami získanými od vybraných expertů. Průběh vyhodnocení probíhal podle následujícího schématu na obrázku 19.



Obr. 19. Schéma vyhodnocení získaných expertních odhadů použitím metody AHP

4.1.1.1 Expertní hodnoty pro váhy faktorů metody BORMp

V prvním kroku je nutné nastavit váhy jednotlivých faktorů, ty budou v metodě BORMp sloužit při hodnocení podmínek, ve kterých navržený IS bude vznikat. Váhy je nutné nastavit pro:

- 13 technických faktorů,
- 7 faktorů prostředí a
- 7 zákaznických faktorů.

Po vysvětlení a zasvěcení jednotlivých expertů do situace, získal autor jejich odhady vah jednotlivých faktorů, které jsou shrnuty v následující tabulce 24 a rozděleny podle jednotlivých expertů, variant a faktorů.

Tab. 24. Expertní odhady vah

		Expert 1	Expert 2	Expert 3	Expert 4
I. Technický faktor					
t1	distribuovaný systém	1,5	2,2	2,0	2,0
t2	doba reakce nebo požadovaná rychlost zpracování/ výkon	1,5	2,2	0,8	1,0
t3	efektivnost koncového uživatele	1,8	0,9	1,0	1,0
t4	složitost vnitřního zpracování	1,2	0,7	0,9	1,0
t5	znovupoužitelnost kódu	0,9	1,6	1,3	1,0
t6	jednoduchost instalace	1,6	1,4	1,6	0,5
t7	jednoduchost užití	1,8	1,1	1,9	0,5
t8	přenositelnost	1,6	2,8	1,8	2,0
t9	snadnost změny	1,2	1,6	1,8	1,0
t10	paralelní vývoj	1,8	0,4	1,6	1,0
t11	speciální požadavky na bezpečnost	1,5	2,0	1,8	1,0
t12	přímé zapojení třetí strany	1,1	1,4	1,1	1,0
t13	požadavek speciálního tréninku	2,2	1,3	1,6	1,0
II. Faktor prostředí					
e1	obeznámení s užitým projektovým modelem (např. RUP)	1,3	0,9	1,1	1,5
e2	zkušenosti s aplikacemi	1,6	1,7	1,7	0,5
e3	zkušenosti s objektovou orientací	1,6	0,9	1,3	1,0
e4	kapacita vedoucího analytika	1,2	2,3	1,3	0,5
e5	motivace	0,7	1,3	1,9	1,0
e6	zaměstnanci na částečný úvazek	1,5	2,0	1,8	1,0
e7	složitost programovacího jazyka	1,8	2,2	1,8	1,0
III. Zákaznický faktor					
c1	znalost problematiky IS	1,5	0,5	1,6	1,2
c2	kapacita projektového manažera u zákazníka	1,1	2,0	1,2	1,1
c3	kapacita pracovníků na projektu	2,1	1,5	1,6	0,8
c4	obeznámení s organizací projektu	2,1	0,5	0,8	1,2
c5	návaznost na existující IS	2,1	2,0	2,2	1,6
c6	složitost nahrazovaných IS	2,2	1,5	0,8	1,6
c7	vyváženost požadavků	1,7	1,0	1,8	1,8
		Varianta 1	Varianta 2	Varianta 3	Varianta 4

4.1.1.2 Aplikace metody AHP na úlohu nastavení vah

V souladu s přístupem AHP byla nejprve sestavena hierarchická struktura problému (viz obrázek 19), dále byly stanoveny váhy pro hodnocená kritéria. Tyto váhy byly nastaveny

pátým expertem na základě jeho expertních znalostí. Ohodnocení vztahů mezi jednotlivými variantami bylo provedeno pátým expertem, ten odhadl vztahy mezi odhady jednotlivých expertů, pomocí kterých byly dopočítány požadované hodnoty.

Následující kapitoly práce představují postup nastavení vztahů mezi variantami a vyhodnocení nejvhodnějších variant.

I. Váhy technického faktoru

a. Stanovení vah kritérií

Vztahy mezi kritérii byly nastaveny pátým expertem, který na základě svých zkušeností mezi nimi nastavil preference. Pro ověření správnosti vah mezi kritérii je nutné sledovat koeficient konzistence (viz kapitola 7.2).

Tab. 25. Váhy stanovené Saatyho metodou.

	kritérium 1	kritérium 2	kritérium 3	Kritérium 4	b_i	v_i
kritérium 1	1,00	3,00	2,00	7,00	2,546	0,503
kritérium 2	0,33	1,00	2,00	3,00	1, 189	0,235
kritérium 3	0,50	0,50	1,00	4,00	1,000	0,197
kritérium 4	0,14	0,33	0,25	1,00	0,330	0,065
Celkem					5,065	

Koeficient konzistence pro váhy technického faktoru je **0,047.**

b. Saatyho matice pro kritérium 1

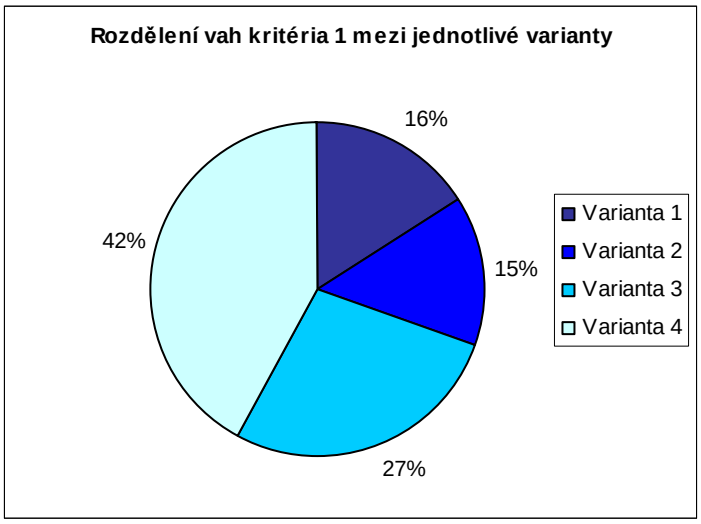
Vztahy mezi kritérii byly kvantifikovány v kroku a. Nyní zbývá provést porovnání na poslední úrovni hierarchie – mezi variantami. K tomu je potřeba vyplnit Saatyho matice, ve kterých se porovnávají varianty z hlediska jednotlivých kritérií. Vztahy v Saatyho matici byly sestaveny opět pátým expertem, který je nastavil na základě svých zkušeností.

Tab. 26. Saatyho matice pro kritérium 1.

0,503	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	0,503•v_i
Varianta 1	1,00	1,00	0,50	0,50	0,707	0,161	0,081
Varianta 2	1,00	1,00	0,50	0,33	0,639	0,145	0,073
Varianta 3	2,00	2,00	1,00	0,50	1,189	0,270	0,136
Varianta 4	2,00	3,00	2,00	1,00	1,861	0,423	0,213
Celkem					4,396		

Koeficient konzistence pro kritérium 1 technického faktoru je **0,015.**

Matice je doplněna koláčovým grafem (viz obrázek 20), který ukazuje, jak si varianty rozdělily váhu daného kritéria (uvedeno tučně v levém horním rohu tabulek).



Obr. 20. Rozdělení váhy kritéria 1 pro jednotlivé varianty technického faktoru

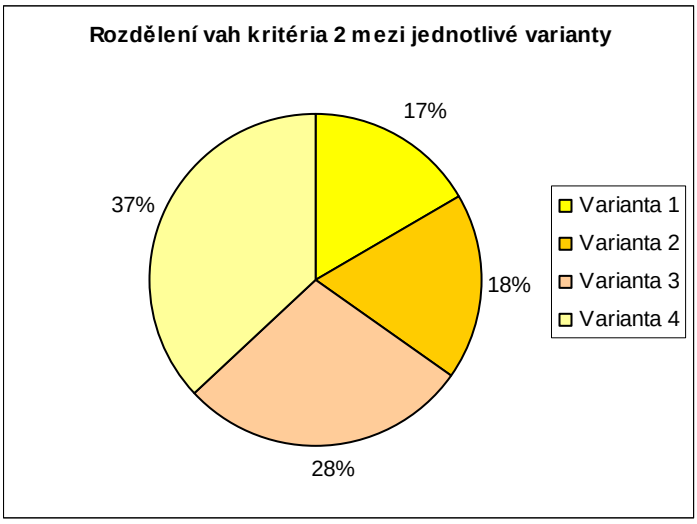
c. Saatyho matice pro kritérium 2

Obdobně jako u kritéria 1 bylo postupováno u následujících kritérií. Opět se vychází z kritériální matice z tabulky 25.

Tab. 27. Saatyho matice pro kritérium 2.

0,235	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,235 \cdot v_i$
Varianta 1	1,00	0,50	1,00	0,50	0,707	0,168	0,039
Varianta 2	2,00	1,00	0,50	0,33	0,760	0,180	0,042
Varianta 3	1,00	2,00	1,00	1,00	1,189	0,282	0,066
Varianta 4	2,00	3,00	1,00	1,00	1,565	0,371	0,087
Celkem					4,221		

Koeficient konzistence pro kritérium 2 technického faktoru je **0,081.**



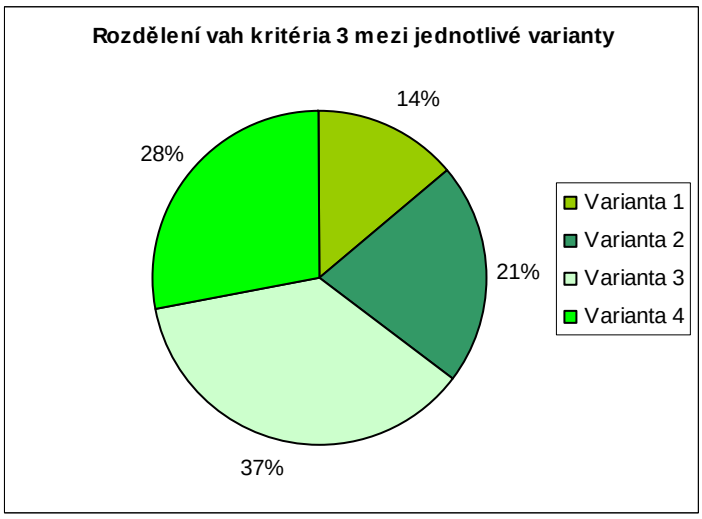
Obr. 21. Rozdělení váhy kritéria 2 pro jednotlivé varianty technického faktoru

d. Saatyho matice pro kritérium 3

Tab. 28. Saatyho matice pro kritérium 3.

0,197	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,197 \cdot v_i$
Varianta 1	1,00	0,50	0,50	0,50	0,595	0,140	0,028
Varianta 2	2,00	1,00	0,33	1,00	0,904	0,212	0,042
Varianta 3	2,00	3,00	1,00	1,00	1,565	0,368	0,073
Varianta 4	2,00	1,00	1,00	1,00	1,189	0,280	0,055
Celkem					4,252		

Koeficient konzistence pro kritérium 3 technického faktoru je **0,051.**



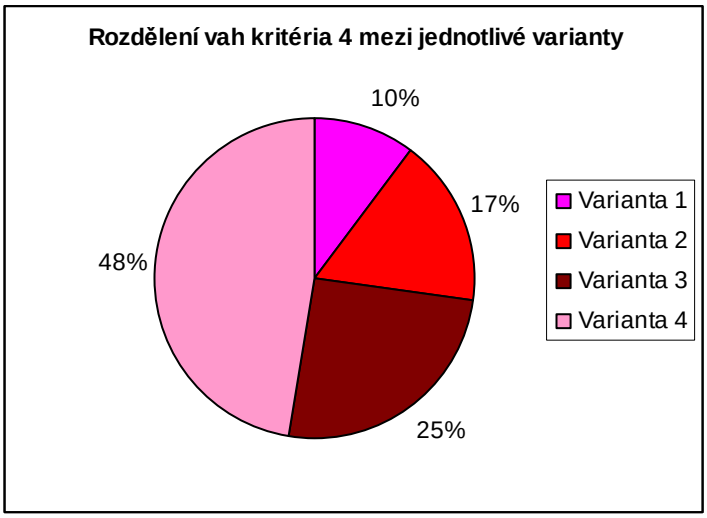
Obr. 22. Rozdělení váhy kritéria 3 pro jednotlivé varianty technického faktoru

e. Saatyho matice pro kritérium 4

Tab. 29. Saatyho matice pro kritérium 4.

0,065	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,065 \cdot v_i$
Varianta 1	1,00	0,33	0,50	0,33	0,485	0,104	0,007
Varianta 2	3,00	1,00	0,50	0,25	0,783	0,168	0,011
Varianta 3	2,00	2,00	1,00	0,50	1,189	0,255	0,017
Varianta 4	3,00	4,00	2,00	1,00	2,213	0,474	0,031
Celkem					4,671		

Koeficient konzistence pro kritérium 4 technického faktoru je **0,069.**



Obr. 23. Rozdělení váhy kritéria 4 pro jednotlivé varianty technického faktoru

f. Vyhodnocení pořadí vybraných variant

Tab. 30. Vyhodnocení variant.

	Součet	Pořadí
Varianta 1	0,155	**4**
Varianta 2	0,168	**3**
Varianta 3	0,291	**2**
Varianta 4	**0,386**	**1**

Stejným způsobem byl aplikován i u faktoru prostředí a zákaznického faktoru.

II. Váhy faktoru prostředí

a. Stanovení vah kritérií

Tab. 31. Váhy stanovené Saatyho metodou.

	kritérium 1	kritérium 2	kritérium 3	kritérium 4	b_i	v_i
kritérium 1	1,00	2,00	1,00	3,00	1,565	0,365
kritérium 2	0,50	1,00	2,00	2,00	1,189	0,277
kritérium 3	1,00	0,50	1,00	2,00	1,000	0,233
kritérium 4	0,33	0,50	0,50	1,00	0,537	0,125
Celkem					4,292	

Koeficient konzistence pro váhy faktoru prostředí je **0,057.**

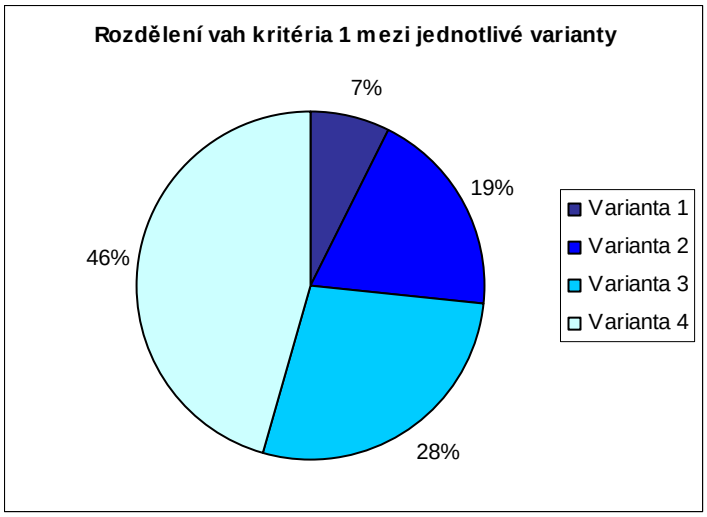
b. Saatyho matice pro kritérium 1

Obdobně jako u kritéria 1 technického faktoru bylo postupováno u následujících kritérií. Nyní se vychází z kritériální matice tabulky 31.

Tab. 32. Saatyho matice pro kritérium 1.

0,365	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,365 \cdot v_i$
Varianta 1	1,00	0,33	0,20	0,25	0,359	0,074	0,027
Varianta 2	3,00	1,00	0,50	0,50	0,931	0,192	0,070
Varianta 3	5,00	2,00	1,00	0,33	1,351	0,278	0,102
Varianta 4	4,00	2,00	3,00	1,00	2,213	0,456	0,166
Celkem					4,854		

Koeficient konzistence pro kritérium 1 technického faktoru je **0,067.**



Obr. 24. Rozdělení váhy kritéria 1 pro jednotlivé varianty faktoru prostředí

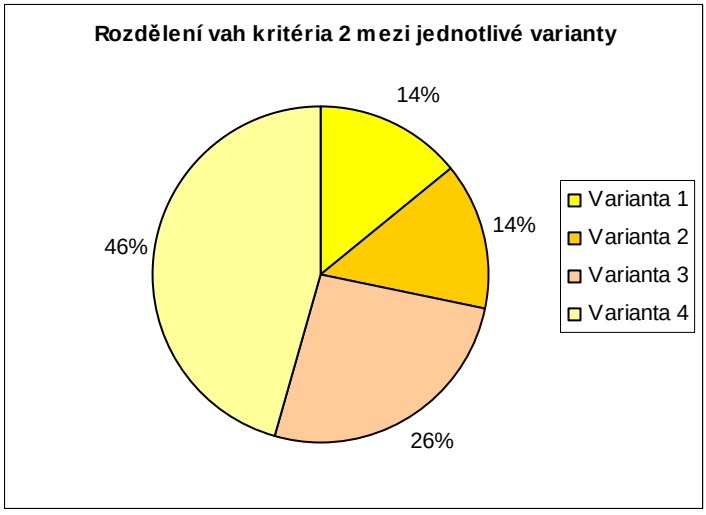
c. Saatyho matice pro kritérium 2

Obdobně jako u kritéria 1 bylo postupováno u následujících kritérií. Opět se vychází z kritériální matice z tabulky 31.

Tab. 33. Saatyho matice pro kritérium 2.

0,277	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,277 \cdot v_i$
Varianta 1	1,00	1,00	0,33	0,50	0,639	0,141	0,039
Varianta 2	1,00	1,00	0,50	0,33	0,639	0,141	0,039
Varianta 3	3,00	2,00	1,00	0,33	1,189	0,263	0,073
Varianta 4	2,00	3,00	3,00	1,00	2,060	0,455	0,126
Celkem					4,527		

Koeficient konzistence pro kritérium 2 technického faktoru je **0,066.**



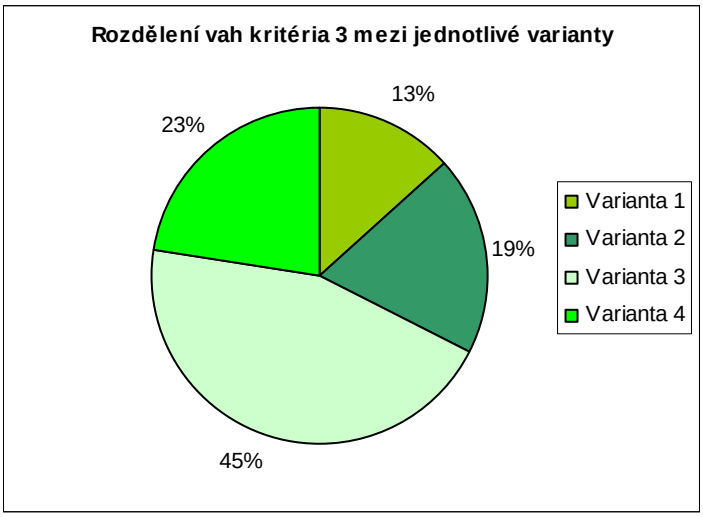
Obr. 25. Rozdělení váhy kritéria 2 pro jednotlivé varianty faktoru prostředí

d. Saatyho matice pro kritérium 3

Tab. 34. Saatyho matice pro kritérium 3.

0,233	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,233 \cdot v_i$
Varianta 1	1,00	0,50	0,25	1,00	0,595	0,134	0,031
Varianta 2	2,00	1,00	0,50	0,50	0,841	0,190	0,044
Varianta 3	4,00	2,00	1,00	2,00	2,000	0,451	0,105
Varianta 4	1,00	2,00	0,50	1,00	1,000	0,225	0,053
Celkem					4,435		

Koeficient konzistence pro kritérium 3 technického faktoru je **0,062.**



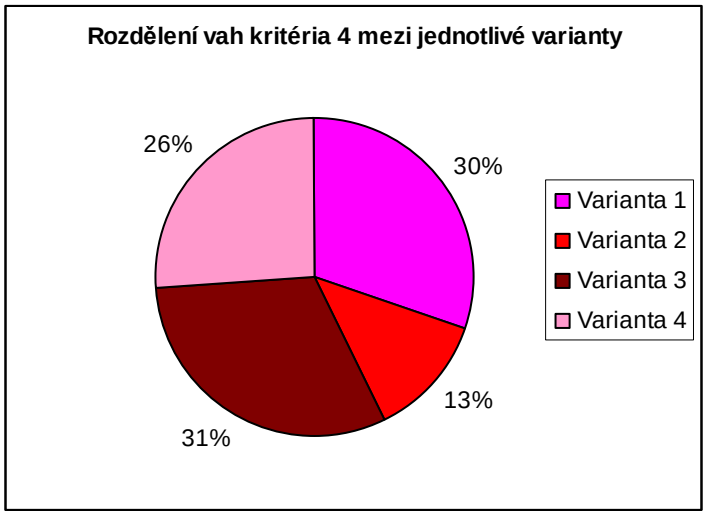
Obr. 26. Rozdělení váhy kritéria 3 pro jednotlivé varianty faktoru prostředí

e. Saatyho matice pro kritérium 4

Tab. 35. Saatyho matice pro kritérium 4.

0,125	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,125 \cdot v_i$
Varianta 1	1,00	2,00	0,33	4,00	1,278	0,302	0,038
Varianta 2	0,50	1,00	0,33	0,50	0,537	0,127	0,016
Varianta 3	3,00	3,00	1,00	0,33	1,316	0,311	0,039
Varianta 4	0,25	2,00	3,00	1,00	1,107	0,261	0,033
Celkem					4,238		

Koeficient konzistence pro kritérium 4 technického faktoru je -**0,008.**



Obr. 27. Rozdělení váhy kritéria 4 pro jednotlivé varianty faktoru prostředí

f. Vyhodnocení pořadí vybraných variant

Tab. 36. Vyhodnocení variant.

	Součet	Pořadí
Varianta 1	0,135	**4**
Varianta 2	0,169	**3**
Varianta 3	0,318	**2**
Varianta 4	**0,378**	**1**

III. Váhy zákaznického faktoru

a. Stanovení vah kritérií

Tab. 37. Váhy stanovené Saatyho metodou.

	kritérium 1	kritérium 2	kritérium 3	kritérium 4	b_i	v_i
kritérium 1	1,00	0,50	2,00	3,00	1,316	0,306
kritérium 2	2,00	1,00	0,33	3,00	1,186	0,275
kritérium 3	0,50	3,03	1,00	2,00	1,319	0,306
kritérium 4	0,33	0,33	0,50	1,00	0,485	0,113
Celkem					4,307	

Koeficient konzistence pro váhy faktoru prostředí je **-0,034.**

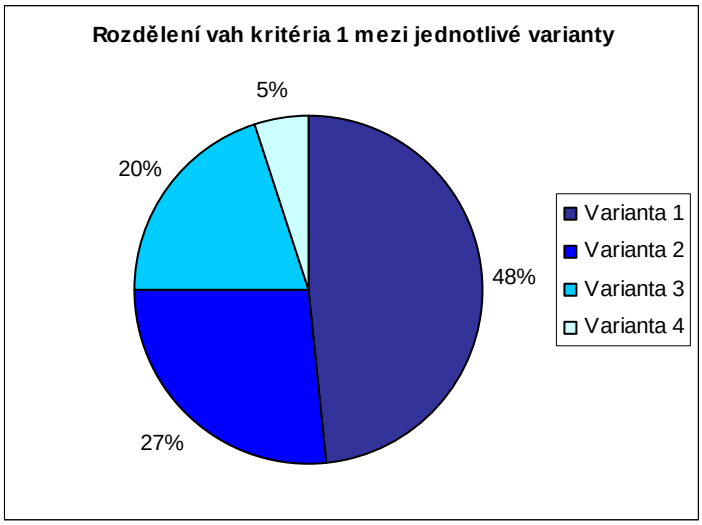
b. Saatyho matice pro kritérium 1

Obdobně jako u kritéria 1 technického faktoru a faktoru prostředí bylo postupováno u následujících kritérií zákaznického faktoru. Nyní se vychází z kritériální matice tabulky 37.

Tab. 38. Saatyho matice pro kritérium 1.

0,306	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,306 \bullet v_i$
Varianta 1	1,00	3,00	2,00	7,00	2,546	0,482	0,176
Varianta 2	0,33	1,00	2,00	6,00	1,414	0,268	0,098
Varianta 3	0,50	0,50	1,00	5,00	1,057	0,200	0,073
Varianta 4	0,14	0,17	0,20	1,00	0,263	0,050	0,018
Celkem					5,280		

Koeficient konzistence pro kritérium 1 technického faktoru je **0,052.**



Obr. 28. Rozdělení váhy kritéria 1 pro jednotlivé varianty zákaznického faktoru

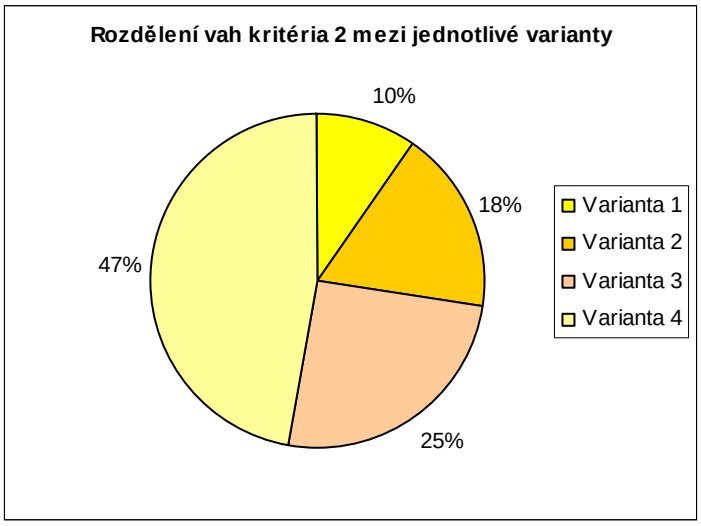
c. Saatyho matice pro kritérium 2

Obdobně jako u kritéria 1 bylo postupováno u následujících kritérií. Opět se vychází z kritériální matice z tabulky 37.

Tab. 39. Saatyho matice pro kritérium 2.

0,275	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,275 \cdot v_i$
Varianta 1	1,00	0,50	0,33	0,25	0,452	0,096	0,027
Varianta 2	2,00	1,00	0,50	0,50	0,841	0,179	0,049
Varianta 3	3,00	2,00	1,00	0,33	1,189	0,253	0,070
Varianta 4	4,00	2,00	3,00	1,00	2,213	0,471	0,130
Celkem					4,695		

Koeficient konzistence pro kritérium 2 technického faktoru je **0,041.**



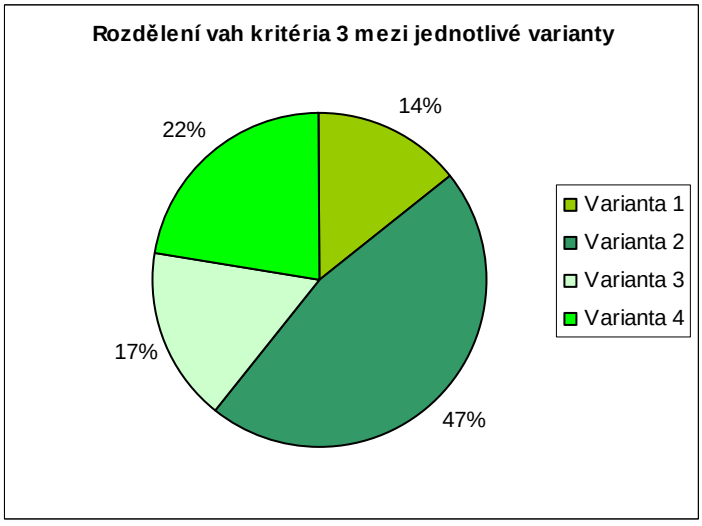
Obr. 29. Rozdělení váhy kritéria 2 pro jednotlivé varianty zákaznického faktoru

d. Saatyho matice pro kritérium 3

Tab. 40. Saatyho matice pro kritérium 3.

0,306	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,306 \cdot v_i$
Varianta 1	1,00	0,33	0,50	1,00	0,639	0,143	0,044
Varianta 2	3,00	1,00	3,00	2,00	2,060	0,462	0,142
Varianta 3	2,00	0,33	1,00	0,50	0,760	0,170	0,052
Varianta 4	1,00	0,50	2,00	1,00	1,000	0,224	0,069
Celkem					4,459		

Koeficient konzistence pro kritérium 3 technického faktoru je **0,056.**



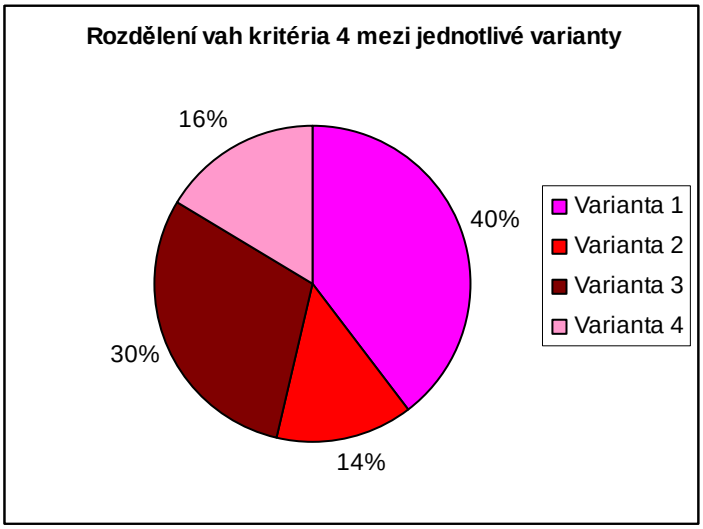
Obr. 30. Rozdělení váhy kritéria 3 pro jednotlivé varianty zákaznického faktoru

e. Saatyho matice pro kritérium 4

Tab. 41. Saatyho matice pro kritérium 4.

0,113	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,113 \cdot v_i$
Varianta 1	1,00	2,00	1,00	3,00	1,565	0,398	0,045
Varianta 2	0,50	1,00	0,33	0,50	0,537	0,137	0,015
Varianta 3	1,00	0,50	1,00	4,00	1,189	0,303	0,034
Varianta 4	0,33	2,00	0,25	1,00	0,639	0,163	0,018
Celkem					3,931		

Koeficient konzistence pro kritérium 4 technického faktoru je -**0,010.**



Obr. 31. Rozdělení váhy kritéria 4 pro jednotlivé varianty zákaznického faktoru

f. Vyhodnocení pořadí vybraných variant

Tab. 42. Vyhodnocení variant.

	Součet	Pořadí
Varianta 1	0,291	**2**
Varianta 2	**0,304**	**1**
Varianta 3	0,229	**4**
Varianta 4	0,235	**3**

IV. Vybrané váhy jednotlivých faktorů

Z experty navržených možností byly na základě výše provedených výpočtů metody AHP vybrány nejvhodnější varianty vah jednotlivým faktorům. Vybrané varianty jsou zvýrazněny červenou barvou v tabulce 43.

Tab. 43. Vyhodnocení variant

		Expert 1	Expert 2	Expert 3	Expert 4
I. Technický faktor					
t1	distribuovaný systém	1,5	2,2	2,0	**2,0**
t2	doba reakce nebo požadovaná rychlost zpracování/ výkon	1,5	2,2	0,8	**1,0**
t3	efektivnost koncového uživatele	1,8	0,9	1,0	**1,0**
t4	složitost vnitřního zpracování	1,2	0,7	0,9	**1,0**
t5	znovupoužitelnost kódu	0,9	1,6	1,3	**1,0**
t6	jednoduchost instalace	1,6	1,4	1,6	**0,5**
t7	jednoduchost užití	1,8	1,1	1,9	**0,5**
t8	přenositelnost	1,6	2,8	1,8	**2,0**
t9	snadnost změny	1,2	1,6	1,8	**1,0**
t10	paralelní vývoj	1,8	0,4	1,6	**1,0**
t11	speciální požadavky na bezpečnost	1,5	2,0	1,8	**1,0**
t12	přímé zapojení třetí strany	1,1	1,4	1,1	**1,0**
t13	požadavek speciálního tréninku	2,2	1,3	1,6	**1,0**
II. Faktor prostředí					
e1	obeznámení s užitým projektovým modelem (např. RUP)	1,3	0,9	1,1	**1,5**
e2	zkušenosti s aplikacemi	1,6	1,7	1,7	**0,5**
e3	zkušenosti s objektovou orientací	1,6	0,9	1,3	**1,0**
e4	kapacita vedoucího analytika	1,2	2,3	1,3	**0,5**
e5	motivace	0,7	1,3	1,9	**1,0**
e6	zaměstnanci na částečný úvazek	1,5	2,0	1,8	**1,0**
e7	složitost programovacího jazyka	1,8	2,2	1,8	**1,0**
III. Zákaznický faktor					
c1	znalost problematiky IS	1,5	**0,5**	1,6	1,2
c2	kapacita projektového manažera u zákazníka	1,1	**2,0**	1,2	1,1
c3	kapacita pracovníků na projektu	2,1	**1,5**	1,6	0,8
c4	obeznámení s organizací projektu	2,1	**0,5**	0,8	1,2
c5	návaznost na existující IS	2,1	**2,0**	2,2	1,6
c6	složitost nahrazovaných IS	2,2	**1,5**	0,8	1,6
c7	vyváženost požadavků	1,7	**1,0**	1,8	1,8
		Varianta 1	Varianta 2	Varianta 3	Varianta 4

4.1.1.3 Expertní hodnoty pro koeficienty faktorů metody BORMp

Pro finalizaci postupu celé metody BORMp musí být v druhém kroku nastaveny koeficienty výpočtových vzorců pro jednotlivé faktory. Pro řešení této úlohy byl použit stejný postup jako u nastavení vah jednotlivých faktorů. Experti poskytli varianty koeficientů zobrazených v tabulce 44. Poskytnuté odhady jsou rozděleny podle jednotlivých expertů.

Tab. 44. Expertní odhady koeficientů

	Expert 1	Expert 2	Expert 3	Expert 4
Technický faktor				
Technický faktor složitosti (tcf)	0,3 + (0,025 • tFactor)	0,6 + (0,009 • tFactor)	0,8 + (0,003 • tFactor)	0,4 + (0,03 • tFactor)
Faktor prostředí				
Složitost faktoru prostředí (ecf)	1,1 + (-0,02 • eFactor)	1,7 + (-0,015 • eFactor)	1,8 + (-0,025 • eFactor)	1,7 + (-0,015 • eFactor)
Zákaznický faktor				
Zákaznický faktor složitosti (ccf)	0,5 + (0,01 • cFactor)	0,6 + (-0,005 • cFactor)	0,9 + (-0,01 • cFactor)	0,3 + (0,02 • cFactor)
	Varianta 1	Varianta 2	Varianta 3	Varianta 4

4.1.1.4 Aplikace metody AHP na úlohu nastavení koeficientů

Při výpočtu byla použita stejná hierarchická struktura problému (viz obrázek 19). Váhy kritérií byly nastaveny pátým expertem na základě jeho expertních znalostí. Ohodnocení vztahů mezi jednotlivými variantami bylo též provedeno pátým expertem, ten odhadl preference mezi odhady jednotlivých expertů.

Pro výběr nejvhodnější varianty byla opět použita metoda AHP.

I. Koeficienty technického faktoru

a. Stanovení vah kritérií

Vztahy mezi kritérii byly nastaveny pátým expertem, který na základě svých zkušeností nastavil mezi nimi preference. Pro ověření správnosti vah mezi kritérii je nutné sledovat koeficient konzistence (viz kapitola 7.2 Úvod do metody AHP).

Tab. 45. Váhy stanovené Saatyho metodou.

	Kritérium 1	kritérium 2	kritérium 3	kritérium 4	b_i	v_i
Kritérium 1	1,00	2,00	3,00	3,00	2,060	0,453
Kritérium 2	0,33	1,00	2,00	4,00	1,278	0,281
Kritérium 3	0,50	0,25	1,00	3,00	0,783	0,172
Kritérium 4	0,20	0,33	0,50	1,00	0,427	0,094
Celkem					4,547	

Koeficient konzistence pro koeficienty technického faktoru je **0,027.**

b. Saatyho matice pro kritérium 1

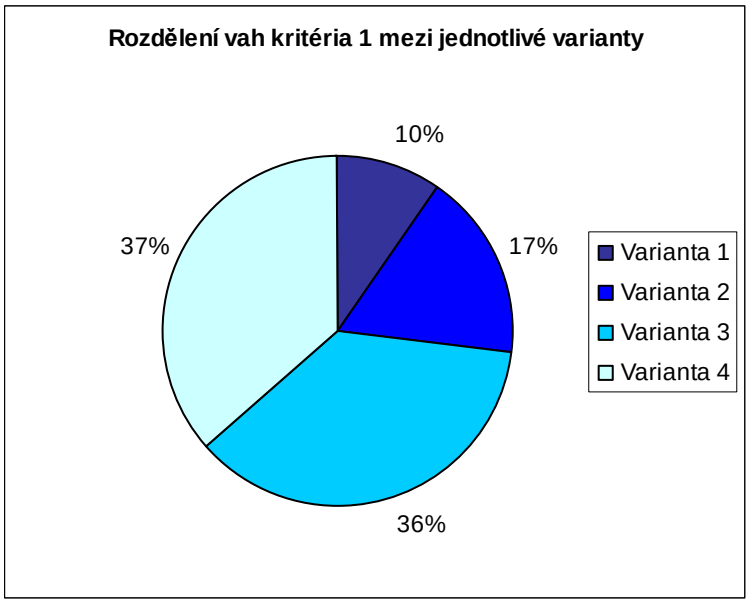
Na základě nastavených vztahů mezi kritérii se nyní provede porovnání na úrovni hierarchie – mezi variantami. K tomu je potřeba vyplnit Saatyho matice, ve kterých se porovnávají varianty z hlediska jednotlivých kritérií. Vztahy v Saatyho matici byly sestaveny opět pátým expertem, který je nastavil na základě svých zkušeností.

Tab. 46. Saatyho matice pro kritérium 1.

0,453	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,453 \cdot v_i$
Varianta 1	1,00	0,33	0,25	0,50	0,452	0,098	0,045
Varianta 2	3,00	1,00	0,50	0,25	0,783	0,170	0,077
Varianta 3	4,00	2,00	1,00	1,00	1,682	0,366	0,166
Varianta 4	2,00	4,00	1,00	1,00	1,682	0,366	0,166
Celkem					4,598		

Koeficient konzistence pro kritérium 1 technického faktoru je **0,053.**

Matice je doplněna koláčovým grafem (viz obrázek 32), který ukazuje, jak si varianty rozdělily váhu daného kritéria (uvedeno tučně v levém horním rohu tabulek).



Obr. 32. Rozdělení váhy kritéria 1 pro jednotlivé varianty technického faktoru

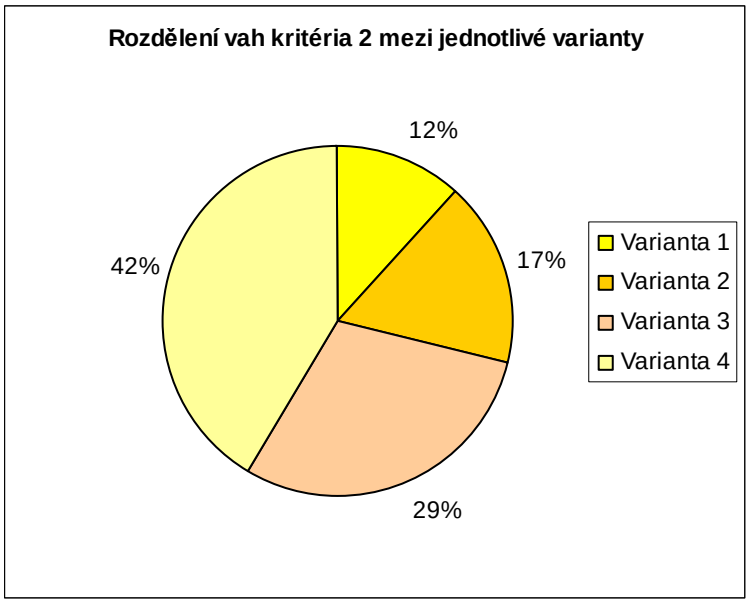
c. Saatyho matice pro kritérium 2

Obdobně jako u kritéria 1 bylo postupováno u následujících kritérií. Opět se vychází z kritériální matice z tabulky 45.

Tab. 47. Saatyho matice pro kritérium 2.

0,281	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,281 \cdot v_i$
Varianta 1	1,00	0,50	0,33	0,50	0,537	0,120	0,054
Varianta 2	2,00	1,00	0,50	0,33	0,760	0,170	0,077
Varianta 3	3,00	2,00	1,00	0,50	1,316	0,294	0,133
Varianta 4	2,00	3,00	2,00	1,00	1,861	0,416	0,188
Celkem					4,474		

Koeficient konzistence pro kritérium 2 technického faktoru je **0,054.**



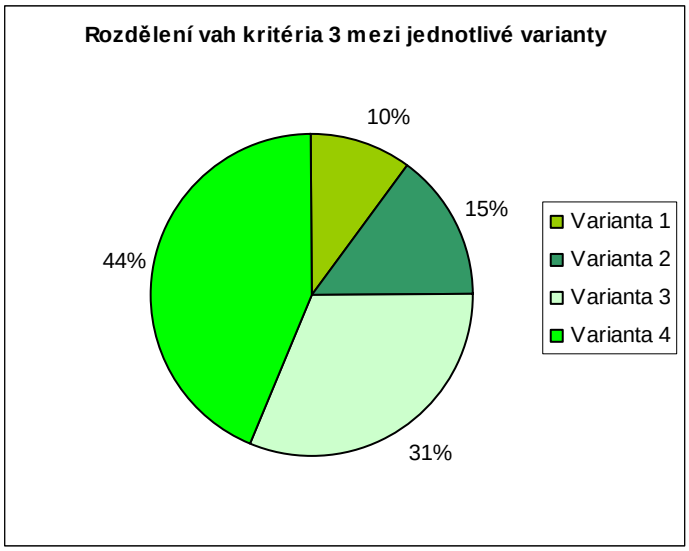
Obr. 33. Rozdělení váhy kritéria 2 pro jednotlivé varianty technického faktoru

d. Saatyho matice pro kritérium 3

Tab. 48. Saatyho matice pro kritérium 3.

0,172	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,172 \cdot v_i$
Varianta 1	1,00	0,50	0,33	0,33	0,485	0,104	0,047
Varianta 2	2,00	1,00	0,33	0,33	0,687	0,146	0,066
Varianta 3	3,00	3,00	1,00	0,50	1,456	0,311	0,141
Varianta 4	3,00	3,00	2,00	1,00	2,060	0,439	0,199
Celkem					4,688		

Koeficient konzistence pro kritérium 3 technického faktoru je **0,040.**



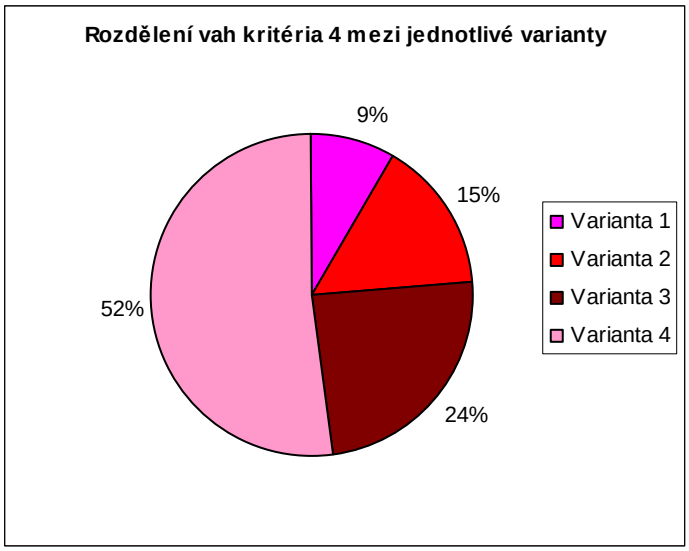
Obr. 34. Rozdělení váhy kritéria 3 pro jednotlivé varianty technického faktoru

e. Saatyho matice pro kritérium 4

Tab. 49. Saatyho matice pro kritérium 4.

0,094	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,094 \cdot v_i$
Varianta 1	1,00	0,33	0,50	0,20	0,427	0,086	0,039
Varianta 2	3,00	1,00	0,33	0,33	0,760	0,153	0,069
Varianta 3	2,00	3,00	1,00	0,33	1,189	0,239	0,108
Varianta 4	5,00	3,00	3,00	1,00	2,590	0,522	0,236
Celkem					4,966		

Koeficient konzistence pro kritérium 4 technického faktoru je **0,082.**



Obr. 35. Rozdělení váhy kritéria 4 pro jednotlivé varianty technického faktoru

f. Vyhodnocení pořadí vybraných variant

Tab. 50. Vyhodnocení variant.

	Součet	Pořadí
Varianta 1	0,185	**4**
Varianta 2	0,290	**3**
Varianta 3	0,548	**2**
Varianta 4	**0,789**	**1**

Stejným způsobem byl aplikován i u faktoru prostředí a zákaznického faktoru.

II. Koeficienty faktoru prostředí

a. Stanovení vah kritérií

Tab. 51. Váhy stanovené Saatyho metodou

	Kritérium 1	kritérium 2	kritérium 3	kritérium 4	b_i	v_i
Kritérium 1	1,00	2,00	3,00	4,00	2,213	0,450
Kritérium 2	0,50	1,00	2,00	4,00	1,414	0,288
Kritérium 3	0,33	0,50	1,00	5,00	0,955	0,194
Kritérium 4	0,25	0,25	0,20	1,00	0,334	0,068
Celkem					4,917	

Koeficient konzistence pro váhy faktoru prostředí je **0,067.**

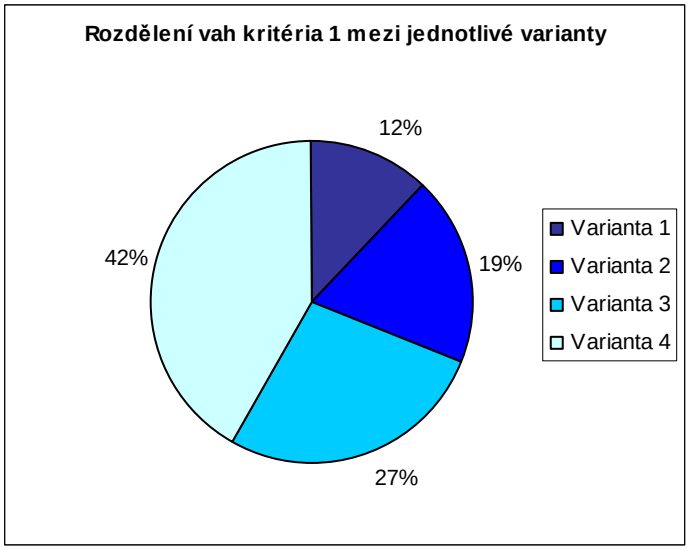
b. Saatyho matice pro kritérium 1

Obdobně jako u kritéria 1 technického faktoru bylo postupováno u následujících kritérií. Nyní se vychází z kritériální matice z tabulky 51.

Tab. 52. Saatyho matice pro kritérium 1.

0,450	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,45 \cdot v_i$
Varianta 1	1,00	0,50	0,33	0,50	0,537	0,121	0,055
Varianta 2	2,00	1,00	0,50	0,50	0,841	0,190	0,085
Varianta 3	3,00	2,00	1,00	0,33	1,189	0,269	0,121
Varianta 4	2,00	2,00	3,00	1,00	1,861	0,420	0,189
Celkem					4,429		

Koeficient konzistence pro kritérium 1 technického faktoru je **0,086.**



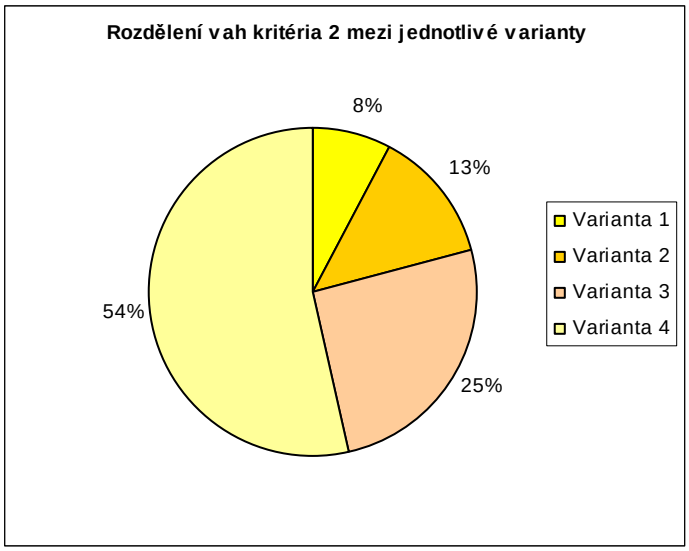
Obr. 36. Rozdělení váhy kritéria 1 pro jednotlivé varianty faktoru prostředí

c. Saatyho matice pro kritérium 2

Tab. 53. Saatyho matice pro kritérium 2.

0,288	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,288 \cdot v_i$
Varianta 1	1,00	0,50	0,25	0,20	0,398	0,077	0,022
Varianta 2	2,00	1,00	0,33	0,33	0,687	0,132	0,038
Varianta 3	4,00	3,00	1,00	0,25	1,316	0,254	0,073
Varianta 4	5,00	3,00	4,00	1,00	2,783	0,537	0,154
Celkem					5,183		

Koeficient konzistence pro kritérium 2 technického faktoru je **0,074.**



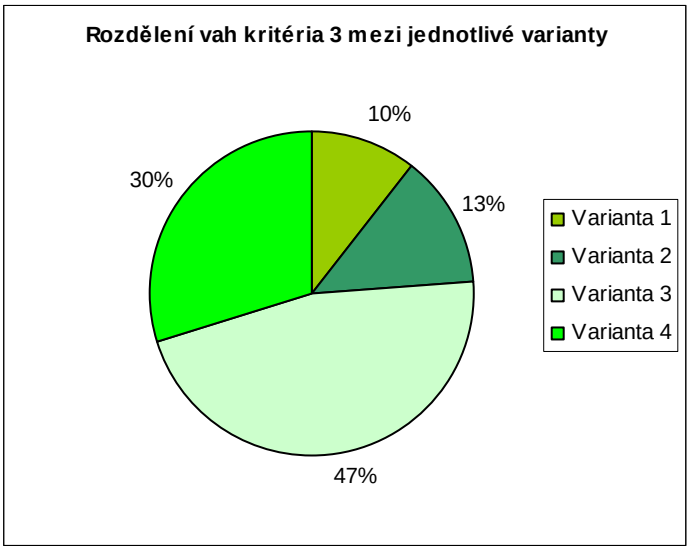
Obr. 37. Rozdělení váhy kritéria 2 pro jednotlivé varianty faktoru prostředí

d. Saatyho matice pro kritérium 3

Tab. 54. Saatyho matice pro kritérium 3.

0,194	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,194 \cdot v_i$
Varianta 1	1,00	0,50	0,25	0,50	0,500	0,105	0,020
Varianta 2	2,00	1,00	0,33	0,25	0,639	0,134	0,026
Varianta 3	4,00	3,00	1,00	2,00	2,213	0,464	0,090
Varianta 4	2,00	4,00	0,50	1,00	1,414	0,297	0,058
Celkem					4,767		

Koeficient konzistence pro kritérium 3 technického faktoru je **0,065.**



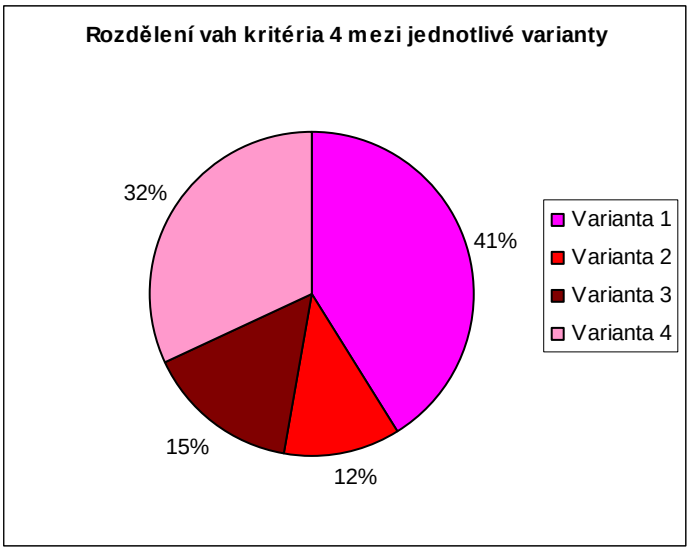
Obr. 38. Rozdělení váhy kritéria 3 pro jednotlivé varianty faktoru prostředí

e. Saatyho matice pro kritérium 4

Tab. 55. Saatyho matice pro kritérium 4.

0,068	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,068 \cdot v_i$
Varianta 1	1,00	2,00	3,00	2,00	1,861	0,410	0,028
Varianta 2	0,50	1,00	0,50	0,33	0,537	0,118	0,008
Varianta 3	0,33	2,00	1,00	0,33	0,687	0,151	0,010
Varianta 4	0,50	3,00	3,00	1,00	1,456	0,321	0,022
Celkem					4,542		

Koeficient konzistence pro kritérium 4 technického faktoru je **0,071.**



Obr. 39. Rozdělení váhy kritéria 4 pro jednotlivé varianty faktoru prostředí

f. Vyhodnocení pořadí vybraných variant

Tab. 56. Vyhodnocení variant

	Součet	Pořadí
Varianta 1	0,125	**4**
Varianta 2	0,158	**3**
Varianta 3	0,294	**1**
Varianta 4	**0,423**	**2**

III. Váhy zákaznického faktoru

a. Stanovení vah kritérií

Tab. 57. Váhy stanovené Saatyho metodou

	kritérium 1	kritérium 2	kritérium 3	kritérium 4	b_i	v_i
kritérium 1	1,00	0,50	0,25	3,00	0,783	0,158
kritérium 2	2,00	1,00	2,00	5,00	2,115	0,428
kritérium 3	4,00	0,50	1,00	4,00	1,682	0,341
kritérium 4	0,33	0,20	0,25	1,00	0,359	0,073
Celkem					4,938	

Koeficient konzistence pro váhy faktoru prostředí je **0,072.**

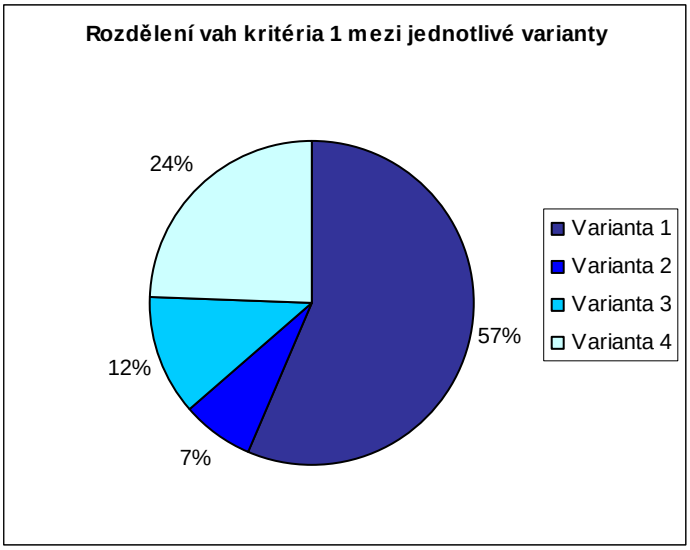
b. Saatyho matice pro kritérium 1

Obdobně jako u kritéria 1 technického faktoru a faktoru prostředí bylo postupováno u následujících kritérií zákaznického faktoru. Nyní se vychází z kritériální matice z tabulky 57.

Tab. 58. Saatyho matice pro kritérium 1.

0,158	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,158 \cdot v_i$
Varianta 1	1,00	7,00	3,00	4,00	3,027	0,563	0,089
Varianta 2	0,14	1,00	0,50	0,33	0,393	0,073	0,012
Varianta 3	0,33	2,00	1,00	0,25	0,639	0,119	0,019
Varianta 4	0,25	3,00	4,00	1,00	1,316	0,245	0,039
Celkem					5,375		

Koeficient konzistence pro kritérium 1 technického faktoru je **0,088.**



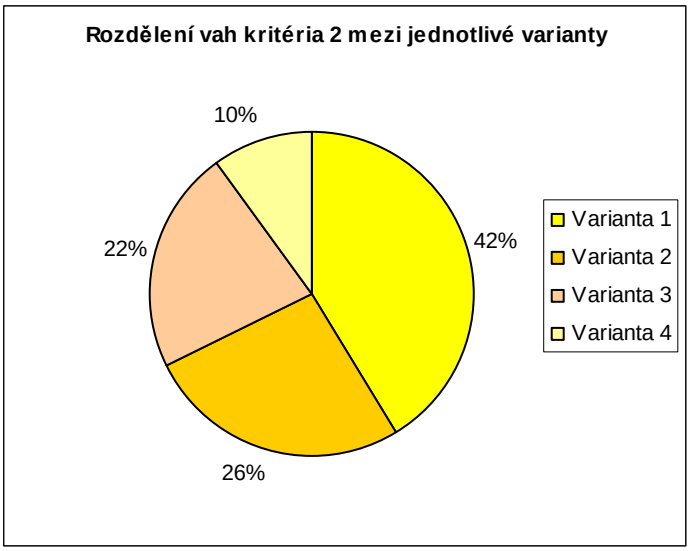
Obr. 40. Rozdělení váhy kritéria 1 pro jednotlivé varianty zákaznického faktoru

c. Saatyho matice pro kritérium 2

Tab. 59. Saatyho matice pro kritérium 2.

0,428	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,428 \cdot v_i$
Varianta 1	1,00	1,00	3,00	4,00	1,861	0,413	0,177
Varianta 2	1,00	1,00	1,00	2,00	1,189	0,264	0,113
Varianta 3	0,33	1,00	1,00	3,00	1,000	0,222	0,095
Varianta 4	0,25	0,50	0,33	1,00	0,452	0,100	0,043
Celkem					4,502		

Koeficient konzistence pro kritérium 2 technického faktoru je **0,053.**



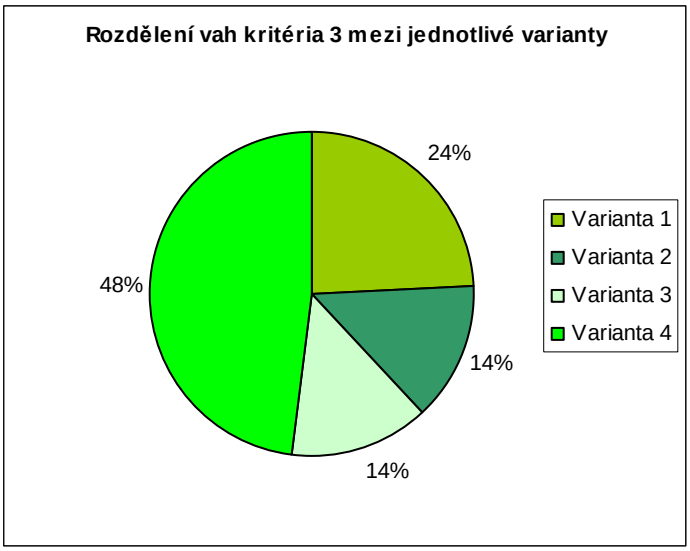
Obr. 41. Rozdělení váhy kritéria 2 pro jednotlivé varianty zákaznického faktoru

d. Saatyho matice pro kritérium 3

Tab. 60. Saatyho matice pro kritérium 3.

0,341	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,341 \cdot v_i$
Varianta 1	1,00	3,00	2,00	0,25	1,107	0,241	0,082
Varianta 2	0,33	1,00	1,00	0,50	0,639	0,139	0,047
Varianta 3	0,50	1,00	1,00	0,33	0,639	0,139	0,047
Varianta 4	4,00	2,00	3,00	1,00	2,213	0,481	0,164
Celkem					4,598		

Koeficient konzistence pro kritérium 3 technického faktoru je **0,098.**



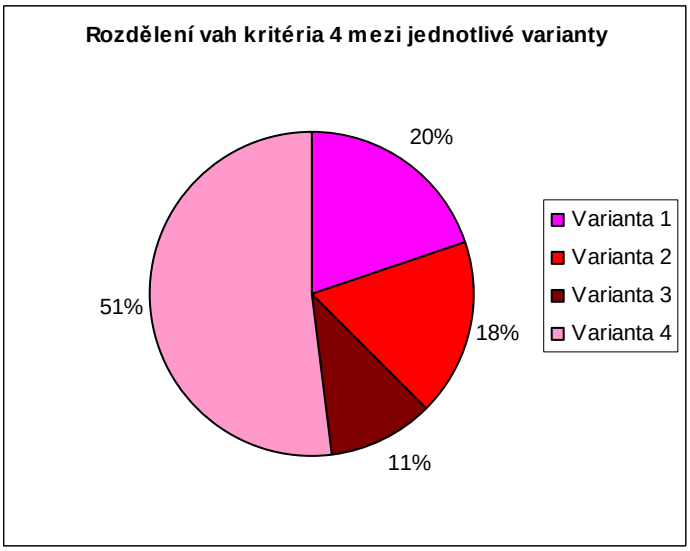
Obr. 42. Rozdělení váhy kritéria 3 pro jednotlivé varianty zákaznického faktoru

e. Saatyho matice pro kritérium 4

Tab. 61. Saatyho matice pro kritérium 4.

0,073	Varianta 1	Varianta 2	Varianta 3	Varianta 4	R_i	v_i	$0,073 \cdot v_i$
Varianta 1	1,00	0,33	3,00	0,50	0,841	0,197	0,014
Varianta 2	3,00	1,00	0,33	0,33	0,760	0,178	0,013
Varianta 3	0,33	0,50	1,00	0,25	0,452	0,106	0,008
Varianta 4	2,00	3,00	4,00	1,00	2,213	0,519	0,038
Celkem					4,266		

Koeficient konzistence pro kritérium 4 technického faktoru je **0,052.**



Obr. 43. Rozdělení váhy kritéria 4 pro jednotlivé varianty zákaznického faktoru

f. Vyhodnocení pořadí vybraných variant

Tab. 62. Vyhodnocení variant.

	Součet	Pořadí
Varianta 1	**0,363**	**4**
Varianta 2	0,185	**3**
Varianta 3	0,169	**1**
Varianta 4	0,283	**2**

IV. Vybrané váhy jednotlivých faktorů

Z experty navržených možností byly na základě výše provedených výpočtů metody AHP vybrány nejvhodnější varianty koeficientů jednotlivým faktorům. Vybrané varianty jsou zvýrazněny červenou barvou v tabulce 63 a použity v následujících kapitolách.

Tab. 63. Vyhodnocení variant

	Expert 1	**Expert 2**	**Expert 3**	**Expert 4**
Technický faktor				
Technický faktor složitosti (tcf)	0,3 + (0,025 • tFactor)	0,6 + (0,009 • tFactor)	0,8 + (0,003 • tFactor)	0,4 + (0,03 • tFactor)
Faktor prostředí				
Složitost faktoru prostředí (ecf)	1,1 + (-0,02 • eFactor)	1,7 + (-0,015 • eFactor)	1,8 + (-0,025 • eFactor)	1,7 + (-0,015 • eFactor)
Zákaznický faktor				
Zákaznický faktor složitosti (ccf)	0,5 + (0,01 • cFactor)	0,6 + (-0,005 • cFactor)	0,9 + (-0,01 • cFactor)	0,3 + (0,02 • cFactor)
	Varianta 1	Varianta 2	Varianta 3	Varianta 4

4.2 Návrh výpočtu složitosti metodou BORMp

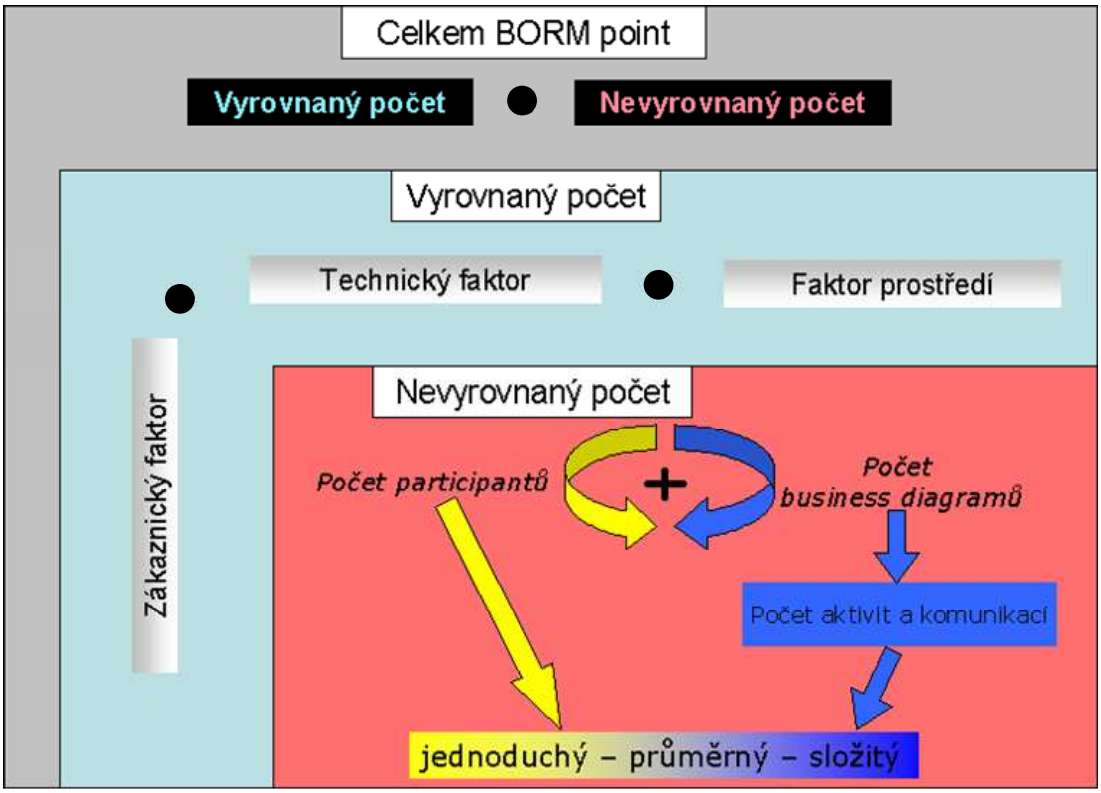
Návrh postupu dále představené metody vychází z výpočtu tzv. metody UCP (viz kapitola 3.4.3 Metoda UCP). Jde o světově uznávanou metodu určenou pro odhad složitosti IS. Navrhovaná metoda BORMp přebírá některé části metody UCP, které dále upravuje pro konkrétní aplikaci na metodu BORM.

Nové části metody BORMp jsou navrženy tak, aby odstranily některé známé nedostatky metody UCP. Za jeden z nich považuje autor této práce malý důraz na postavení zákazníka v projektu, protože nezkrocené požadavky právě z jejich strany mohou velice výrazně ovlivnit pracnost.

Metoda BORMp se pokouší odhadovat pracnost na základě vybraných komponent metody BORM. Výpočet je podobně jako u Use Case rozdělen do dvou nezávislých kroků. V prvním kroku se odhad opírá o počet participantů, kteří budou s vyvíjeným IS pracovat, v druhém se pracuje s tzv. procesními diagramy obsahujícími zapojené participanty a jejich aktivity a stavy.

Dále lze z procesních diagramů získat informace o přechodech mezi aktivitami a stavy v rámci jednotlivých participantů a o komunikacích, které si mezi sebou posílají zapojení participanti.

Navrhovaný výpočet odhadu pracnosti pomocí metody BORMp se rozpadá do dvou kroků. Je to z důvodu oddělení výpočtu založeném na modelování IS a faktorů, které zprostředkovaně hodnotí prostředí, ve kterém je IS vytvářen.



Obr. 44. Struktura výpočtu metody BORMp.

V prvním kroku se provede výpočet nevyrovnané části hodnoty. Ve druhém se určí jednotlivé faktory – technický faktor, faktor prostředí a zákaznický faktor.

- Nekorigovaná část výpočtu,
  - Počet participantů
    - participant je v této práci chápán jako objekt představující účastníka modelovaného procesu [Carda, Merunka, Polák 2003].
  - Počet procesních (business) diagramů
    - procesní diagram je v práci chápán jako obraz všech možných průběhů procesu pomocí současného zobrazení dvou dimenzí problému. První dimenzí jsou role (zn. průběh aktivit jednoho objektu v procesu) zúčastněných objektů jako automaty se stavy a přechody. Druhou dimenzí je sled komunikací mezi objekty, který představuje řídící a datové toky mezi objekty v procesu [Carda, Merunka, Polák 2003].
- Technický faktor.
- Faktor prostředí.
- Zákaznický faktor.

- Faktor produktivnosti.

4.2.1 Nekorigovaná část výpočtu BORMp

V první části výpočtu se metoda BORMp, jak již bylo výše naznačeno, rozděluje dále do dvou mezikroků. Toto rozdělení je zvoleno z důvodu rozlišení složitosti jednotlivých participantů a složitosti procesů uvnitř procesních diagramů, které přímo souvisejí se vznikajícím IS.

A Identifikace, klasifikace a přidělení vah participantům {unadjusted participant weights – upw}
Předpokládá se, že participanti jsou externími objekty, které mají se systémem nějaký vzájemný vztah, jde především o objekty – např. koncoví uživatelé, další programy, datové sklady, atd. Měly by být součástí projektové dokumentace. Kvůli širokému využití metody BORM pro modelování různých procesů musí být do výpočtu zahrnuti pouze participanti, kteří mají přímý vztah k vytvářenému IS.

V dalším kroku jsou participanti rozděleni dle úrovně složitosti na:

jednoduchý – průměrný – složitý.

Jednoduchý – jiný systém s rozhraním na měřený systém prostřednictvím různých automatizovaných programů (např. standardní aplikační program (nebo API)).

Průměrný – buď další systém nebo datový sklad, který je na měřený systém napojen prostřednictvím protokolu nebo textu založeném na uživatelském rozhraní. Průměrný participant spolupracuje se systémem přes protokol (např. HTTP, TCP/ IP, atd.).

Složitý – osoba spolupracující se systémem prostřednictvím grafického rozhraní (jde hlavně o koncové uživatele, kteří jsou klasifikováni v kategorii složitý).

Tab. 64. Návrh hodnot vah pro jednotlivé kategorie participantů

Typ participanta	Definice	Váha
Jednoduchý	Automatizované rozhraní	1
Průměrný	Interaktivní nebo protokolem řízené rozhraní	2
Složitý	Grafické rozhraní (lidský faktor)	3

Po rozdělení participantů do třech kategorií, jsou jejich počty v rámci každé kategorie sečteny a roznásobeny přidělenými váhami. Jednoduchému participantovi je přidělena váha 1, průměrnému váha 2 a složitému váha 3. Nevyrovnaná váha participanta se celkem získá součtem vyvážených hodnot z jednotlivých kategorií.

Tab. 65. Návrh výpočtu nevyrovnané váhy participanta

Typ participanta	Váha participanta	Počet participantů	Celkem
Jednoduchý	1	__ • 1 =	
Průměrný	2	__ • 2 =	
Složitý	3	__ • 3 =	
Nekorigovaná váha participantů celkem (Total unadjusted participants weight - upw)			

B identifikace, klasifikace a přidělení vah procesním diagramům {unadjusted business diagram weights – ubdw}

Protože je metoda BORM široce použitelná pro mapování všech procesů souvisejících se vznikem IS, je v dalším kroku nezbytné identifikovat procesní diagramy, které přímo souvisejí se vznikajícím IS. Procesním diagramům se přidělí váhy závislé na počtu aktivit a/ nebo transakcí. Nejprve jsou zjištěné procesní diagramy rozděleny do kategorií:

jednoduchý – průměrný – složitý.

Rozdělení se provede na základě počtu aktivit a podle konkrétních počtů transakcí (tzn. součet komunikací a datových toků), které jsou hraničními hodnotami mezi definovanými kategoriemi. Hraniční hodnoty jsou zobrazeny v tabulce 66.

Aktivity byly pro určení složitosti procesních diagramů vybrány záměrně, protože představují jednotlivé části chování procesních (business) objektů tak, jak byly rozpoznány technikou OBA. V procesních diagramech se pomocí vybraných aktivit provádějí přechody mezi stavy objektů

a zároveň komunikují s aktivitami spolupracujících objektů. Pro konceptuální analýzu jsou aktivity významné proto, že se z nich odvozují metody .

Stavy nebyly vybrány z toho důvodu, že představují zpracování úkolů daným participantem bez vlivu na ostatní objekty. Provádět odhad na základě scénářů by také nebyl jednoznačný, protože se v praxi pravidelně stává, že jednomu scénáři odpovídá jeden procesní diagram.

Dalším kritériem je počet komunikací mezi jednotlivými aktivitami participantů a zároveň počet přechodů mezi stavy a aktivitami u všech zúčastněných participantů v procesním diagramu. Tyto dva počty se sečtou a použijí se jako druhý hodnotící faktor s nižší prioritou. Dá se říci, že toto kritérium je pouze doplňkové, protože hlavním určujícím hlediskem je počet aktivit. Pouze v případě, že počet aktivit a počet komunikací nespadá do stejných úrovní složitosti, pak je nutné rozhodnout intuitivně. Lze doporučit řídit se kategorií složitosti, do které spadne právě počet aktivit.

Tab. 66. Návrh hodnot vah pro odhad složitosti procesních diagramů.

Typ procesního diagramu	Popis	Váha
Jednoduchý	1 - 5 aktivit nebo 3 – 11 komunikací (komunikace + přechody)	5
Průměrný	6 - 10 aktivit nebo 12 – 18 komunikací (komunikace + přechody)	10
složitý	11 a více aktivit nebo 18 a více komunikací (komunikace + přechody)	15

Po rozdělení procesních diagramů podle počtu aktivit a komunikací, je přidělena jednotlivým úrovním určitá váha. Dále se provede součet počtu diagramu v jednotlivých kategoriích a roznásobení s přidělenou váhou, poté jsou výsledky sečteny po jednotlivých řádcích. Výsledkem tabulky je nekorigovaný počet procesních diagramů (viz tabulka 67).

Tab. 67. Návrh výpočtu nevyrovnané váhy procesních diagramů celkem.

Typ procesního diagramu	Počet aktivit	Váha procesního diagramu	Výpočet	Celkem
Jednoduchý	1 – 5	5	__ • 5 =	
Průměrný	6 – 10	10	__ • 10 =	
Složitý	11 a více	15	__ • 15 =	
Nekorigovaná váha procesních diagramů celkem (Total unadjusted business diagram weight – ubdw)				

C Výpočet nekorigované části BORMp {Unadjusted BORMp}

Celkový počet nekorigovaných BORMp je získán sečtením dvou výše vypočtených částí – nekorigovaná váha participanta (upw) a nekorigovaná váha procesních diagramů (ubdw).

unadjusted BORMp (uBORMp) = upw + ubdw

4.2.2 Technický faktor {Technical Factor}

Pro dokončení výpočtu celkové složitosti projektu je nutné určit tzv. technický faktor, který bude ovlivněn i úrovní technických znalostí zapojených zaměstnanců. Jejich úkolem bude ohodnocení vybraných 13 faktorů, které souvisejí s technickou stránkou vytvářeného IS. Hodnotící stupnice zahrnuje hodnoty od 0 do 5. Pokud bude faktoru přidělena hodnota 0, pak je jeho vliv na IS bezvýznamný, v případě 5 je velmi významný.

Tab. 68. Výpočet hodnoty technického faktoru BORMp

Pořadí	Popis	Váha	Výpočet	Celkem	Komentář
t1	distribuovaný systém	2,5	__ •2,5 =		
t2	doba reakce nebo požadovaná rychlost zpracování/ výkon	1	__ • 1 =		
t3	Efektivnost koncového uživatele	1,5	__ •1,5 =		
t4	složitost vnitřního zpracování	1	__ • 1 =		
t5	znovupoužitelnost kódu	1	__ • 1 =		
t6	jednoduchost instalace	0,5	__• 0,5 =		
t7	jednoduchost užití	1	__ • 1 =		
t8	přenositelnost	2	__ • 2 =		
t9	snadnost změny	0,5	__• 0,5 =		
t10	souběžnost	0,5	__• 0,5 =		
t11	zahrnout speciální bezpečnostní cíle	1	__ • 1 =		
t12	Poskytnout přímý přístup třetí straně	1	__ • 1 =		
t13	Požadavek speciálního tréninku	0,5	__• 0,5 =		
Technický faktor celkem (Total technical factor - tFactor)					X

Metoda BORMp používá pro určení technického faktoru stejnou tabulku jako metoda UCP. Je to z důvodu, že celkem dostatečně pokrývají technickou oblast vývoje softwaru, která je velmi významnou součástí celého projektu.

I v této metodě je velmi důležité klást důraz na faktory, které mají největší vliv na úspěšnou nebo neúspěšnou realizaci projektu. Vysvětlení jednotlivých pojmů z tabulky je následující:

- distribuovaný systém – složitost architektury systému (zda systém běží na jednom či více strojích),
- doba reakce nebo požadovaná rychlost zpracování/ výkon,
- efektivnost koncového uživatele – schopnost uživatele řešit složité úkoly s pomocí IS jednodušeji a rychleji,
- složitost vnitřního zpracování – závisí na složitosti vnitřní struktury IS,
- znovupoužitelnost kódu – znamená další použití celého nebo alespoň části kódu v tvorbě dalších částí či aplikací,
- jednoduchost instalace – stupeň náročnosti jednotlivých kroků při instalaci systému,
- jednoduchost užití - jde o rychlou naučitelnost a srozumitelnost pro uživatele při práci se systémem,
- přenositelnost – jednoduché použití IS na jiných platformách (viz kapitola 2.2.1.6 Přenositelnost {Portability}),

- snadnost změny – úroveň složitosti v případě úprav celého systému nebo jeho jednotlivých částí,
- paralelní vývoj – předpoklady pro tvorbu několika částí systému najednou,
- speciální požadavky na bezpečnost – v případech, kdy od systému vyžadovány specifické bezpečnostní požadavky (cíle),
- přímé zapojení třetí strany – úroveň spolupráce se třetí stranou, za kterou je možno považovat např. nezávislého testéra nebo poradenskou firmu, jejich úkolem je nezávislý pohled,
- požadavek speciálního tréninku – určených pro IT zaměstnance budoucích zákazníků.

Hodnoty (0–5) přidělené každému faktoru se roznásobí přidělenou váhou, dále se provede jejich součet. Takto je získán tzv. technický faktor (tFactor), který je dále dosazen do vzorce pro výpočet složitosti technického faktoru {technical complexity factor – tcf}.

$$tcf = 0{,}4 + (0{,}03 \cdot tFactor)$$

4.2.3 Faktor prostředí {Environment Factor}

Prostředí je v metodě BORMp chápáno z pohledu dodavatelské firmy, proto sem patří např. schopnosti zaměstnanců, nástroje či metody, které jsou pro tvorbu nového IS k dispozici. Na rozdíl od metody UCP došlo ke snížení počtu faktorů, protože faktor „vyváženost požadavků“ byl přesunut mezi zákaznické faktory.

Vliv na odhad doby realizace projektu má také skutečnost související s dovednostmi a znalostmi zaměstnanců. Tyto vlivy jsou zastřešeny tzv. faktorem prostředí. Postup jeho hodnocení je stejný jako u technického faktoru, tzn. nejprve ohodnotíme 7 faktorů týkajících se prostředí, ve které IS vzniká. Máme opět k dispozici stupnici od 0 do 5 (0 – bezvýznamný, 5 – velmi významný dopad).

Tab. 69. Seznam faktorů prostředí BORMp

Pořadí	Popis	Váha	Výpočet	Celkem	Komentář
e1	použitý projektový model	1,5	__ • 1,5 =		
e2	zkušenosti s aplikacemi	0,5	__ • 0,5 =		
e3	zkušenosti s objektovou orientací	1	__ • 1 =		
e4	kapacita vedoucího analytika	0,5	__ • 0,5 =		
e5	Motivace	1	__ • 1 =		
e6	zaměstnanci na částečný úvazek	1	__ • 1 =		
e7	složitost programovacího jazyka	1	__ • 1 =		
Faktor prostředí celkem (Total environment factor - eFactor)					X

Vysvětlení vybraných pojmů v tabulce:

- obeznámení s užitým projektovým modelem – úroveň znalostí metodik jednotlivými zaměstnanci pracující na daném projektů IS,
- zkušenosti s aplikacemi – úroveň znalostí jednotlivých zaměstnanců u aplikací používaných v projektu (aplikace pro vývoj, návrh, realizace, atd.),
- zkušenosti s objektovou orientací – schopnost zaměstnanců pracovat v objektovém prostředí,
- kapacita vedoucího analytika – objem doby, kterou je schopen vedoucí analytik věnovat odhadovanému projektu,
- motivace – úroveň motivace pro dokončení daného projektu včas a ve stanovených nákladech,
- zaměstnanci na částečný úvazek – počet zaměstnanců na částečný úvazek zapojených v projektu,
- složitost programovacího jazyka.

Po ohodnocení faktorů se jim přidělené hodnoty roznásobí váhami a výsledky se sečtou. Touto operací je získána celková hodnota faktoru prostředí {Total environment factor – eFactor}, která je dosazena do vzorce pro výpočet složitosti faktoru prostředí {Environment complexity factor – ecf} jehož výsledkem je hodnota faktoru prostředí.

$$ecf = 1{,}7 + (-0{,}015 \cdot eFactor)$$

4.2.4 Zákaznický faktor {Customer Factor}

Metoda BORMp zavádí do procesu odhadu složitosti vývoje IS nový pohled, jde o tzv. zákaznický faktor, jehož úkolem je zprostředkovat dopad účasti zákazníka na realizovaném projektu. Jak už bylo v úvodu naznačeno nekoordinované požadavky zákazníků mohou výrazným způsobem ovlivnit pracnost vyvíjeného IS.

Postup jeho hodnocení je stejný jako u technického faktoru a faktoru prostředí, tzn. nejprve ohodnotíme 7 faktorů týkajících se zákazníka, s dopadem jeho účasti na vývoj IS. Máme opět k dispozici stupnici od 0 do 5 (0 – bezvýznamný, 5 – velmi významný dopad).

Tab. 70. Seznam zákaznických faktorů BORMp

Pořadí	Popis	Váha	Výpočet	Celkem	Komentář
c1	znalost problematiky IS	0,5	__ • 0,5 =		
c2	kapacita projektového manažera u zákazníka	2	__ • 2 =		
c3	kapacita pracovníků na projektu	1,5	__ • 1,5=		
c4	obeznámení s organizací projektu	0,5	__ • 0,5 =		
c5	návaznost na existující IS	2	__ • 2 =		
c6	složitost nahrazovaných IS	1,5	__ • 1,5 =		
c7	vyváženost požadavků	1	__ • 1 =		
Zákaznický faktor celkem {Total customer factor – cFactor}					X

Vysvětlení pojmů:

- znalost problematiky IS – znalosti z oblasti IT u členů projektového týmu zákazníka,
- kapacita projektového manažera u zákazníka – skutečný čas, který je schopen věnovat projektu,
- kapacita pracovníků na projektu – skutečný čas, který mají členové projektového týmu vyčleněný pro práci na projektu,
- obeznámení s organizací projektu – porozumění organizačnímu uspořádání projektu pracovníky projektového týmu,
- návaznost na existující IS – úroveň spolupráce zaváděného IS s existujícími,
- složitost nahrazovaných IS – jakou roli měl nahrazovaný IS na fungování organizace,
- vyváženost požadavků – schopnost zákazníka definovat své požadavky a četnost jejich pravděpodobných změn.

Po ohodnocení faktorů se jim přidělené hodnoty roznásobí váhami a výsledky se sečtou. Touto operací je získána celková hodnota zákaznického faktoru {Total customer factor – cFactor}, která je dále dosazena do vzorce pro výpočet složitosti zákaznického faktoru {Customer complexity factor – ccf}, jehož výsledkem je hodnota faktoru prostředí.

$$ccf = 0{,}5 + (0{,}01 \cdot cFactor)$$

4.2.5 Faktor produktivnosti {Productivity factor}

Důležitým vstupem pro metody odhadu složitosti vývoje IS je též faktor produktivnosti. Jde o stanovený počet člověkohodin v závislosti na různých vlivech (příkladem mohou být zkušenosti programátorského týmu, velikost vyvíjeného IS, atd.).

Dle odhadů autora práce se dá předpokládat hodnota vyšší než u metody UCP, na základě expertního odhadu byla stanovena hodnota **30 člověkohodin na 1 aBORMp**. Důvodem vyššího počtu člověkohodin u faktoru produktivnosti je zákaznický faktor, který metoda BORMp zavádí.

Metoda UCP používá pro základní určení počtu člověkohodin na jeden Use Case point jednoduché pravidlo (3.4.3.2.4 Faktor produktivnosti {Productivity factor}). To funguje na základě kombinace součtů hodnot faktorů prostředí. Na základě výsledku je doporučena hodnota člověkohodin na jeden Use Case point (podrobnosti viz kapitola 3.4.3.2.4). V dalších fázích vývoje metody BORMp se předpokládá sestavení pravidla obdobného, které bude vytvořeno na základě testování dat již realizovaných projektů.

Námětem dalšího výzkumu metody BORMp může být provázání odhadu pracnosti metodou BORMp na metodu COCOMO. Pomocí metody BORMp by se odhadl objem vytvářeného IS a metodou COCOMO by se následně odhadla skutečná pracnost navrhovaného IS. Důvodem tohoto řešení může být riziko ve vztahu mezi objemem a pracností, který nemusí být ve všech případech lineární.

4.2.6 Celkový počet BORMp {Total BORMp}

Pro celkový počet BORMp předpokládá autor práce dosazení výše vypočtených částí do konečného vzorce, ve kterém se znásobí nekorigovaná část BORMp s technickým faktorem, faktorem prostředí a zákaznickým faktorem a je získán celkový počet BORMp {adjusted BORMp – aBORMp}.

$$aBORMp = uBORMp \cdot tcf \cdot ecf \cdot ccf$$

Složitost je nyní odhadnuta bezrozměrným číslem, které vzejde ze vzorce pro korekční počet BORMp. Aby došlo u metody BORMp k překlenutí od složitosti ke skutečné pracnosti, je nutné ji vynásobit stanoveným faktorem produktivnosti.

$$Pracnost = aBORMp \cdot pf$$

4.3 Testování metody BORMp

Pro ověření výše navržených hodnot a postupů metody BORMp byly použity skutečné projekty realizované poradenskou společností patřící ve svém oboru mezi významné subjekty na českém i mezinárodním trhu. Hlavní překážkou hlubšího testování metody je absence dostatečného množství projektů realizovaných metodou BORM. Pokud však projekt realizován byl, není příliš jednoduché získat skutečnou pracnost daného projektu, protože většina firem se hájí tím, že se jedná o citlivá data, která nemohou poskytnout třetí straně. Proto i u dvou testovaných projektů musí být uvedeny pouze obecné popisy projektů z důvodu ochrany informací.

Projekt 1:

Projekt IS pro významnou českou logistickou společnost. Jednalo se o projekt velkého rozsahu v řádu desítek miliónů korun. Projektový tým se skládal z ekonomické, procesní a informatické části. Tzn. že bylo nutné pro odhad pracnosti IT vybrat participanty a procesní diagramy, které jsou přímo spojené se vznikajícím IS.

Projekt 2:

Projekt interního IS v řádu miliónů korun. Projekt by se dal klasifikovat z hlediska rozsahu jako středně malý. Opět bylo nutné vybrat ty participanty a procesní diagramy, které mají přímý vztah na vytvářený IS.

Na vybraných projektech působil autor práce jako člen projektového týmů pod vedením zkušených projektových manažerů na straně nejmenované poradenské společnosti a proto mohl zhodnotit získané výsledky i osobně.

4.3.1 Provedení odhadu složitosti projektů metodou BORMp

Postup výpočtu složitosti metodou BORMp:

1) U projektů byli v prvním kroku vybráni participanti a procesní diagramy, které přímo souvisejí s vytvářeným IS.
2) Celkový počet nekorigovaných BORMp – zde byli participanti a procesní diagramy ohodnoceny na ordinální stupnici *„jednoduchý – průměrný – složitý“* a sečteny po kategoriích (viz tabulky 71, 72).

Tab. 71. Ohodnocení participantů dle ordinální stupnice

Participant	Ohodnocení do kategorií	
	Projekt 1	Projekt 2
Par. 1	S	S
Par. 2	S	S
Par. 3	S	S
Par. 4	S	J
Par. 5	P	P
Par. 6	S	P
Par. 7	S	J
Par. 8	J	S
Par. 9	P	P
Par. 10	P	P
Par. 11	J	
Par. 12	J	
Par. 13	S	
Celkem		
Jednoduchý	3	2
Průměrný	3	4
Složitý	7	4

Tab. 72. Ohodnocení procesních diagramů dle ordinální stupnice

	Ohodnocení procesních diagramů do kategorií							
	Projekt 1				Projekt 2			
Diagram	Počet aktivit	Počet komunikací	Počet přechodů	Atribut	Počet aktivit	Počet komunikací	Počet přechodů	Atribut
Diag. 1	17	3	30	S	6	5	5	J
Diag. 2	19	10	41	S	6	5	5	J
Diag. 3	8	3	12	P	11	10	8	S
Diag. 4	23	14	43	S	12	12	9	S
Diag. 5	10	5	14	P	11	11	9	S
Diag. 6	4	2	4	J	11	11	9	S
Diag. 7	6	2	5	P	13	12	11	S
Diag. 8	7	2	12	P	12	12	10	S
Diag. 9	11	6	6	S	12	13	5	S
Diag. 10	16	7	21	S	10	8	12	P
Diag. 11	11	7	10	S	10	8	8	P
Diag. 12	8	4	21	P	9	8	8	P
Diag. 13	8	4	27	P	6	5	0	J
Diag. 14	7	5	7	P	8	7	0	P
Diag. 15	3	2	0	J	11	8	0	P
Diag. 16	18	12	30	S	6	5	4	P
Diag. 17	7	4	10	P	6	4	4	P
Diag. 18	8	5	12	P	6	5	4	P
Diag. 19	9	6	13	P	3	2	0	J
Diag. 20	3	2	6	J	7	5	5	P
Diag. 21	15	10	22	S	6	5	5	P
Diag. 22	9	5	14	P	7	5	4	P
Diag. 23	7	5	11	P	13	8	10	S
Diag. 24	6	4	9	P	9	6	8	P
Diag. 25	3	2	0	J	6	4	9	P
Diag. 26	3	2	0	J				
Diag. 27	3	2	0	J				
Diag. 28	3	2	0	J				
Diag. 29	4	5	0	J				
Diag. 30	19	9	32	S				
Diag. 31	9	7	13	P				
Diag. 32	16	11	21	S				
Diag. 33	5	5	4	J				
Diag. 34	23	13	45	S				
Celkem								
Jednoduchý				**9**				**4**

Průměrný	**14**		**13**
Složitý	**11**		**8**

Dále došlo k roznásobení vybranými vahami (viz tabulky 64 a 66), čímž byly získány nekorigované váhy participantů a procesních diagramů celkem (viz tabulka 73).

Tab. 73. Ohodnocení procesních diagramů dle ordinální stupnice

	Projekt 1	Projekt 2
	Participanti	
Jednoduchý	3	2
Průměrný	6	8
Složitý	21	12
Nekorigovaná váha participantů celkem	**30**	**22**
	Procesní diagramy	
Jednoduchý	45	20
Průměrný	140	130
Složitý	165	120
Nekorigovaná váha procesních diagramů celkem	**350**	**270**
Celkový počet nekorigovaných BORMp	**380**	**292**

3) Technický faktor (viz kapitola 4.2.2 Technický faktor {Technical Factor}) – následuje ohodnocení technických faktorů z hlediska realizovaných projektů. Faktory se projdou jeden po druhém a každému se přidělí hodnota z hodnotící stupnice 0-5 (viz tabulka 74).

Tab. 74. Výpočet Složitosti technického faktoru

			Projekt 1		Projekt 2	
Pořadí	**Popis**	**Váha**	**Stupnice**	**Celkem**	**Stupnice**	**Celkem**
t1	Distribuovaný systém	2	3	6	4	8
t2	doba reakce nebo požadovaná rychlost zpracování/ výkon	1	4	4	2	2
t3	efektivnost koncového uživatele	1	2	2	3	3
t4	složitost vnitřního zpracování	1	2	2	2	2
t5	znovupoužitelnost kódu	1	1	1	2	2
t6	Jednoduchost instalace	0,5	1	0,5	0	0
t7	Jednoduchost užití	0,5	2	1	4	2
t8	přenositelnost	2	0	0	0	0
t9	snadnost změny	1	3	3	2	2
t10	paralelní vývoj	1	4	4	0	0
t11	speciální požadavky na bezpečnost	1	3	3	1	1
t12	přímé zapojení třetí strany	1	2	2	0	0
t13	požadavek speciálního tréninku	1	1	1	0	0
Technický faktor celkem (tFactor)			X	29,5	X	22
Složitost technického faktoru (tcf = 0,4 + (0,03 • tFactor))				**1,285**		**1,06**

4) Faktor prostředí (viz kapitola 4.2.3 Faktor prostředí {Environment Factor}) – stejným způsobem jako technické faktory se ohodnotí i faktory prostředí (viz tabulka 75).

Tab. 75. Výpočet Složitosti faktoru prostředí

			Projekt 1		Projekt 2	
Pořadí	**Popis**	**Váha**	**Stupnice**	**Celkem**	**Stupnice**	**Celkem**
e1	obeznámení s užitým projektovým modelem (např. RUP)	2	3	6	1,5	3
e2	zkušenosti s aplikacemi	1	2	2	3	3
e3	zkušenosti s objektovou orientací	1	3	3	0	0
e4	kapacita vedoucího analytika	1	4	4	2	2
e5	Motivace	1	2	2	2	2
e6	zaměstnanci na částečný úvazek	0,5	2	1	1	0,5
e7	složitost programovacího jazyka	2	2	4	1	2
Faktor prostředí celkem (eFactor)			X	22	X	12,5
Složitost faktoru prostředí (ecf = 1,7 + (-0,015 • eFactor))				**1,37**		**1,5125**

5) Zákaznický faktor (viz kapitola 4.2.4 Zákaznický faktor {Customer Factor}) – stejným způsobem jako technické faktory a faktory prostředí se ohodnotí i zákaznický faktory (viz tabulka 76).

Tab. 76. Výpočet Složitosti faktoru prostředí

			Projekt 1		Projekt 2	
Pořadí	**Popis**	**Váha**	**Stupnice**	**Celkem**	**Stupnice**	**Celkem**
c1	znalost problematiky IS	2	3	6	3	6
c2	kapacita projektového manažera u zákazníka	1	2	2	0	0
c3	kapacita pracovníků na projektu	1	4	4	2	2
c4	obeznámení s organizací projektu	1	3	3	2	2
c5	návaznost na existující IS	1	5	5	3	3
c6	složitost nahrazovaných IS	0,5	3	1,5	1	0,5
c7	vyváženost požadavků	2	4	8	2	4
Zákaznický faktor celkem (cFactor)			X	29,5	X	17,5
Složitost zákaznického faktoru (ccf = 0,5 + (0,01 • cFactor))				**0,795**		**0,675**

6) Faktor produktivnosti (viz kapitola 4.2.5 Faktor produktivnosti {Productivity factor}) – při testování bude použita hodnota 30 člověkohodin na 1 BORM bod.

7) Celkový počet BORMp (viz kapitola 4.2.6 Celkový počet BORMp {Total BORMp}) a celkový odhad pracnosti včetně porovnání se skutečnými hodnotami, za které byly oba projekty realizovány (viz tabulka 77).

Tab. 77. Výpočet celkové složitosti a pracnosti metodou BORMp

		projekt 1	projekt 2
Nevyrovnaný počet	Nekorigovaná váha participantů celkem	30	22
	Nekorigovaná váha procesních diagramů celkem	350	270
	Celkový počet nekorigovaných BORMp	380	292
Vyrovnaný počet	Technický faktor	1,285	1,060
	Faktor prostředí	1,370	1,513
	Zákaznický faktor	0,795	0,675
	Celkový počet BORMp	531,832	316,001
	Faktor produktivnosti (člověkohodin/ 1 aBORMp)	30	
	Pracnost celkem (člověkohodin)	**15 955**	**9 480**
	Skutečná pracnost (člověkohodin)	**17 032**	**7 040**

Z tabulky 77 je zřejmé, že hodnota pracnosti dosažená u projektu 1, který byl rozsahem zařazen mezi projekty velké, je nižší než jeho skutečná. Naproti tomu u projektu menšího rozsahu dosáhla metoda BORMp hodnoty vyšší než byla skutečná pracnost. Z tohoto zjištění by se dala vyvodit řada závěrů, v tuto chvíli se lze spokojit pouze s konstatováním, že metoda produkuje poddimenzované odhady pro velké a naddimenzované odhady pro malé projekty, což je zjištění, které by mělo být vzato v úvahu při dalším testování.

Dále se autor práce domnívá, že výsledný odhad u projektu 1 se dá pokládat za celkem dobrý, protože tento odhad by byl proveden v období, kdy není dostatek informací pro přesnější odhady. Při představě, že by se náklady projektu pohybovaly v řádek desítek miliónů korun, pak za těchto okolností je diference +/ -2 000 člověkohodin celkem zanedbatelná. Hůře však dopadlo hodnocení projektu 2, zde získaná diference má podstatný dopad na výslednou pracnost.

Hlavní překážkou dalšího testování je nedostatek projektů realizovaných metodou BORM, které by mohly být použity.

4.4 Softwarové nástroje pro metodu BORMp

Jelikož jde o novou metodu, neexistuje zatím žádný softwarový nástroj, který by podporoval práci s metodou a umožňoval její použití i uživatelům, kteří nemají hlubší znalosti potřebné pro její aplikaci na vybraný projekt. Vývojem softwaru CraftCase [20], který podporuje modelování IS pomocí BORMovské notace, začala společnost eFractal. Její aktivity v oblasti vývoje od minulého roku převzala CRAFT.CASE LIMITED založená ve Velké Británii. V současné době se vyvíjí verze 2.0.

Bylo by velice přínosné, kdyby po dostatečném ověření spolehlivosti výsledků metody BORMp, došlo k zakomponování funkce odhadu složitosti pomocí metody přímo do tohoto nástroje.

Výsledkem by byl uživatelský komfort, kdy by uživatel v průběhu či po finalizaci návrhu IS znal odhad pracnosti svého nově-vznikajícího IS, tyto hodnoty by CraftCase generoval automaticky.

5 Závěry

5.1 Autorova interpretace představených metod

Srovnání v práci představených metod je komplikované. Důvod je prostý, metody BORMp a UCP pracují se vstupy ještě před fází, kdy se rozhoduje o složitosti algoritmů potřebných pro provedení programu. Nedá se tedy jednoznačně určit, zda vykazují lepší nebo horší výsledky při vyšším výskytu netriviálních algoritmů a zpracovávaných logických souborů či naopak, jak je tomu u prvních dvou metod – Function a Feature Points.

V případě srovnání metody UCP a BORMp nastane situace, kdy se porovnávají metody určené pro rozdílné metodologie, tzn. každá z metod je vázána na určitý postup. To ale nebrání možnosti aplikovat obě metody na týž problém. Znamenalo by to ovšem provedení analýzy projektu metodologií UML i BORM. Při srovnání známosti metod se autor práce domnívá, že metoda BORMp je v nevýhodě, protože notace UML je obecně daleko známější a používanější.

Naproti tomu metody Function Points a Feature Points generují stejné početní výsledky u aplikací, kde je přibližně počet algoritmů a logických datových souborů. Pokud se odhaduje počet Function nebo Feature Points u aplikací, které obsahují více algoritmů než souborů, ačkoliv nejde o neobvyklý systémový software, pak metoda Feature Points vygeneruje vyšší celkový počet jednic než metoda Function Points. Naopak, jestliže bude systém obsahovat několik algoritmů a mnoho souborů, které jsou běžné pro některé IS, bude metoda Feature Points generovat nižší celkový výsledek než metoda Function Points.

Metody SPR Feature Points a IBM Function Points jsou metody založené na podobném konceptu, ale v některých případech se jimi poskytnuté výsledky liší, jak bylo zmíněno ve výše uvedeném odstavci.

Na výše uvedených úvahách lze provést shrnutí rozdílů mezi metodami Function a Feature Points [13], které poskytují:

- stejné výsledky u aplikací, kde je stejný počet algoritmů a logických souborů,
- rozdílné výsledky u aplikací, kde je větší počet algoritmů než logických souborů. V těchto případech ačkoliv nejde o neobvyklý systémový software, generuje metoda Feature Points „přesnější“ výsledky (např. MIS),

- rozdílné výsledky u aplikací, kde je menší počet algoritmů než logických souborů. V těchto případech ačkoliv nejde o neobvyklý systémový software, generuje metoda Function Points „přesnější“ výsledky.

Obecně autor práce doporučuje paralelní aplikaci představených metod na konkrétních projektech a zpřesňování výsledků v průběhu životního cyklu projektu. Toto řešení je schopno nabídnout přesnější odhad, ale vyžádá si vyšší dodatečné náklady související s udržováním a aktualizací dvou projektových dokumentací.

5.2 Diskuze

Autor této práce se domnívá, že s rostoucí složitostí vznikajících softwarových nástrojů, bude růst význam odhadů složitosti, který je významným vstupním faktorem pro rozhodování o realizovatelnosti projektů.

Důvodů lze uvést několik. Jedním z nich je, že v počátečních fázích vývoje produktů, kdy je k dispozici pouze minimum potřebných informací, je jakýkoliv jejich zdroj užitečný a poskytuje možnost rozhodovat, zda bude projekt perspektivní nebo ne.

Naproti tomu lze pochybovat o správnosti symetrického členění vah (jednoduchý – průměrný – složitý), avšak je možné jej chápat jako východiskové nastavení. Při opakovaném používání metod na stejných projektech lze doporučit parametrické učení se systému (získané parametry měření se korigují v časovém pořadí) a tedy všechny parametry se budou v čase měnit, což zabezpečí stále přesnější odhad.

V předcházejících kapitolách bylo zmíněno, že je možné v průběhu projektu přehodnocovat získanou pracnost v závislosti na přibývajících vstupních datech, které umožňují počáteční odhady zpřesňovat.

Zdroje firem jsou samozřejmě omezené. Proto je důležité správně určit, do kterých projektů se vyplatí investovat finanční prostředky a znalosti zaměstnanců. Stejně tak jejich včasné stažení z projektů později přehodnocených jako neperspektivní.

V poslední době začínají růst na významu objektové databázové technologie, jejichž perspektivní technologie nabízejí velký užitný potenciál. Je jen otázkou času, kdy se na ně výrobci IS přeorientují.

S rostoucím významem objektových technologií očekává autor i růst zájmu o odhad složitosti objektových systémů, tzn. především metod Feature Points, UCP a BORMp. Autor práce upozorňuje, že je potřeba si pokaždé uvědomit, že se bude vždy jednat spíše o odhad budoucí složitosti než o exaktně přesný výsledek. Snahou všech, kteří se zabývají touto problematikou, by mělo být zdokonalování používaných metod, které budou i lépe reagovat na potřeby tvůrců IS.

V souvislosti s tím autor předpokládá, že vývoj přístupů pro odhad složitosti a pracnosti vývoje IS se bude spíše orientovat na vznik metod, které budou vyvíjeny spíše adresně jednotlivým modelovacím metodologiím než obecné postupy aplikovatelné na jakékoli metodiky.

Důvodem tohoto tvrzení je předpoklad, že příslušné odhady musí vycházet ze vstupních informací určité metody. To znamená, že pro každou metodu bude nutné vypracovat specifický postup odhadu pracnosti, který má šanci být mnohem přesnější než odhad podle „jednotného postupu“. V této práci byly tyto teze aplikovány na velmi perspektivní metodu BORM.

5.3 Zhodnocení dosažených výsledků práce

Předložená práce si kladla za cíl představit a zhodnotit existující metody odhadu složitosti IS. Autor práce považuje za velkou škodu, že metody představené v práci jsou používány spíše sporadicky a praktici i přes některé výjimky působící se k nim staví spíše rezervovaně.

Mezi zcela zásadní přínos práce patří vývoj nové metody BORMp, určené pro odhad složitosti IS navržených v metodě BORM. Jeden z podnětů vedoucí ke vzniku metody BORMp vzešel z praktických požadavků projektových manažerů v jedné z poradenských firem působících České republice. Právě oni při praktické aplikaci metody BORM identifikovali chybějící metodu odhadu složitosti jako možná úskalí dalšího využívání metody BORM.

I přes nedostatek projektů navržených metodou BORM, na kterých by bylo možné provést otestování metody BORMp, se podařilo získat data alespoň u dvou. Porovnání skutečných výsledků a výsledků zjištěných metodou BORMp včetně zhodnocení rozdílů je v kapitole 4.3 Testování metody BORMp.

5.4 Aplikace výsledků práce

Výsledky předložené práce budou uplatnitelné:

I. při výuce odborných předmětů,

II. v akademické sféře, kde dojde k představení metody BORMp co nejširšímu okruhu odborníků,

III. při odhadu složitosti skutečných projektů IS v různých odvětvích lidské činnosti.

5.5 Náměty pro další vědecký výzkum

Testování nové metody BORMp na dvou realizovaných projektech nepovažuje autor za dostatečné. Proto bych chtěl pokračovat v jejím vývoji a zdokonalování v rámci další vědecko-výzkumné činnosti.

Za velmi zajímavý podnět považuje autor této práce myšlenku prof. Vaníčka, který při společných diskusích navrhl přístup, ve kterém by metoda BORMp odhadovala pouze vztah mezi požadavky ze zadání a predikovaný rozsah softwarového systému. A výsledný rozsah získaný metodou BORMp by byl vstupním parametrem metody COCOMO, která by poskytovala celkovou pracnost projektu.

Další oblastí, ve které by mohla metoda BORMp být uplatnitelná, je oblast reinženýringu procesů. Zde by se díky široké uplatnitelnosti metody BORM v tomto oboru mohla aplikovat i logika metody BORMp. Příkladem může být – odhad pracnosti zavedení nově navržených procesů v organizaci. Tento směr vývoje je však nutné podrobit odborné diskusi a hlubšímu zkoumání včetně otestování na v praxi realizovaných projektech.

6 Použitá literatura

6.1 Publikace autora

6.1.1 Zahraniční a cizojazyčné publikace

[Struska 2008] Struska, Z.: BORM Method and Complexity Estimation. SCIENTIA AGRICULTURAE BOHEMICA Published by the Czech University of Agriculture. Prague 2008.

[Struska, Závodný 2008a] Struska, Z., Závodný, M.: Specifics of eGovernment applications and their complexity. in proceedings of 6thEastern European eGov Days 2008 (EEEG 2008), Prague 2008.

[Struska, Vaníček 2007g] Struska, Z., Vaníček, J.: *Complexity Estimation of eGovernment applications development.* International Conference on Theory and Practice of Electronic Governance (ICEGOV) 2007. Macao, China 2007. ISBN 978-1-59593-822-0.

[Struska 2007c] Struska, Z.: *The concept comparison of BORMp method with Use Case Points.* International Conference on Software Engineering Theory and Practice 2007 (SETP-07). Orlando, USA. ISBN 978-0-9727412-6-2.

[Struska, Merunka 2007b] Struska, Z., Merunka, V.: *BORMp - New Concept Proposal of Complexity Estimation Method.* ICEIS 2007 9th International Conference on Enterprise Information Systems, Madeira Portugal. ISBN 978-972-8865-90-0.

[Vraný,Struska,Merunka2006d] Vraný, J., Struska, Z., Merunka, V.: *Object normalization as the contribution to the area of formal methods of object-oriented database design*. ICEIS 2006 Eight International Conference on Enterprise Information Systems, Paphos Cyprus. ISBN 972-8865-43-0.

[Struska, Vaníček 2005b] Struska, Z., Vaníček, J.: *Measurement and rating of information systems quality. Part 3: Design Complexity and Software Engineering Consequences.* PEF ČZU. Praha 2005.

[Struska, Vaníček 2005a] Struska, Z. , Vaníček, J.: *Measurement and rating of information systems quality. Part 2: Quality Model*. PEF ČZU. Praha 2005.

6.1.2 Domácí publikace

[Struska, Brožek 2007f] Struska, Z., Brožek, J.: *Specifika odhadu složitosti eGovernment aplikací.* Sborník konference Objekty 2007. Ostrava 2007. ISBN 978-80-248-1635-7.

[Brožek, Struska 2007e] Brožek, J., Struska, Z.: *BPMN - nový standard pro modelování business procesů.* Sborník konference Agrární perspektivy 2007 16. ročník. Praha 2007. ISBN 978-80-213-1675-1.

[Struska, Brožek 2007d] Struska, Z., Brožek, J.: *Přístupy stanovení koeficientů metody BORMp*. Sborník konference Agrární perspektivy 2007 16. ročník. Praha 2007. ISBN 978-80-213-1675-1.

[Struska 2007a] Struska, Z.: *Srovnání konceptu metody BORMp s metodou Use Case Points*. Sborník doktorské vědecké konference Think Together 2007. Praha 2007. ISBN 80-213-1474-5.

[Struska 2006g] Struska Z.: *Představení konceptu nové metody pro odhad složitosti – BORMp*. Sborník konference Objekty 2006. Praha 2006. ISBN 80-213-1568-7.

[Struska, Pergl 2006f] Struska Z., Pergl, R.: *Srovnání metod COCOMO a analýzy Function Points.* Sborník konference Agrární perspektivy 2006 15. ročník. Praha 2006. ISBN 80-213-1531-8.

[Pergl, Struska 2006e] Pergl, R., Struska, Z.: *Čím mohou přispět nejznámější agilní metodiky ke zlepšení vývojového procesu?* Sborník konference Agrární perspektivy 2006 15. ročník. Praha 2006. ISBN 80-213-1531-8.

[Pergl, Struska 2006c] Pergl, R., Struska, Z.: *Agilní modelování a metodika BORM.* Sborník konference Tvorba software 2006. Ostrava 2006. ISBN 80-248-1082-4.

[Struska, Pergl 2006b] Struska, Z, Pergl, R.: *Metody odhadu pracnosti založené na modelu*. Sborník konference Tvorba software 2006. Ostrava 2006. ISBN 80-248-1082-4.

[Struska 2006a] Struska, Z.: *Metody odhadu pracnosti*. Sborník doktorské vědecké konference Think Together 2006. Praha 2006. ISBN 80-213-1474-5.

[Struska 2005c] Struska, Z.: *Metody odhadu složitosti v objektovém prostředí – Function a Feature Points*. Sborník konference Objekty 2005. Ostrava 2005. ISBN 80-213-0682-3.

[Struska 2005b] Struska, Z.: *Porovnání vybraných přístupů aplikace funkčních jednic*. Sborník konference Agrární perspektivy 2005 14.ročník. Praha 2005. ISBN 80-213-1372-2.

[Struska 2005a] Struska, Z.: *IT Balanced scorecard*. Sborník doktorské vědecké konference 2005. PEF ČZU. ISBN 80-213-1314-5.

[Struska 2004] Struska, Z.: *Projekt informačního systému v podniku*. Diplomová práce 2004. FEK ZČU Plzeň.

6.2 Ostatní použité prameny

[Albrecht, Gaffney 1983] Albrecht, A. J. and Gaffney, J. E., Jr. *Software Functions, Source Line sof Code and Development Effort Prediction.: A Software Science Validation*, IEEE Transactions on Software Engineering (TSE) 9, no. 6 (November 1983).

[Basl 2002] Basl, J. *Podnikové informační systémy. Podnik v informační společnosti*. Grada Publishing 2002. Praha. ISBN 80-247-0214-2.

[Beck 1997] Beck K.: *Smalltalk Best Practice Patterns*, Prentice Hall 1997, ISBN 0-13-476904-X.

[Blaha, Premerlani 1995] Blaha M., Premerlani M.: *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall 1995, ISBN 0-13-123829-9

[Boehm 1975] Boehm, B. W.: *Practical Strategies for Developing System*, Addison-Wesley, Reading, 1975.

[Boehm 1981] Boehm, B. W.: *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, 1981.

[Booch 1994] Booch Grady: *Object-Oriented Analysis and Design with Applications*, Second Edition, Benjamin Cummings 1994, ISBN 0-8053-5340-2.

[Brooks 1975] Brooks, F. P.: *The Mythical Man Month*, Addison-Wesley, 1975.

[Brožová, Houška, Šubrt 2003] Brožová, Houška, Šubrt: *Modely pro vícekriteriální rozhodování.* PEF ČZU Praha 2003. ISBN 80-213-1019-7.

[Brynjolfsson, Hitt 1998] Brynjolfsson, E., Hitt, L. M.: *Beyond the productivity paradox*. Communication of the ACM, August 1998.

[Buchalcevová 2005] Buchalcevová A.: *Metodiky vývoje a údržby informačních systémů*, Grada 2005, ISBN 80-247-1075-7.

[Calcio 2005] D. Ligett, *CALICO 7.0 Calibration Software*, SoftStar Systems, 2005. přístupný na http://www.softstarsystems.com/calico.htm, Květen 2006.

[Carda, Merunka, Polák 2003] Carda A., Merunka V., Polák J.: *Umění systémového návrhu*, Grada 2003, ISBN 80-247-0424-2.

[Costar 2005] D. Ligett, *COSTAR 7.0 Calibration Software*, SoftStar Systems, 2005. přístupný na http://www.softstarsystems.com/demo.htm, Květen 2006.

[Drbal 2002] Drbal P.: *Extrémní programování a metodický přístup*, ve sborníku konference Tvorba softwaru 2002, ISBN 80-85988-74-7.

[Drdla, Rais 2001] Drdla, M, Rais, K. *Řízení změn ve firmě*. Computer Press 2001. Praha. ISBN 80-7226-411-7.

[Edwards 2007] Edwards, R.: Advantages of the COSMICS-FPP method. IFPUG Bulletin, 20. 7. 2007. www.cosmicson.com/advantagecs.asp.

[Garmus, Herron 2001] Garmus, D., Herron, D. *Function Points analysis: measurement practices for successful software project*. Addison Wesley. Upper Saddle River 2001. ISBN 0-201-69944-3.

[Goldberg, Kenneth 1995] Goldberg Adele, Kenneth Rubin S.: *Succeeding with Objects - Decision Frameworks for Project Management*, Addison Wesley 1995, ISBN 0-201-62878-3.

[Henderson-Sellers,Bulthis1998] Henderson-Sellers, B.; Bulthuis, A.: *Object-Oriented Metamethods*, Springer-Verlag New York, 1998, ISBN 0-387-98257-4.

[Hoffman 2000] Hoffman, D. *The darker side of metrics.* Software Quality Methods, LLC 2000. www.softwarequalitymethods.com/papers/darkmets%paper.pdf.

[Hopkins 1992] Hopkins T.: *Animating an Actor Programming Model*, Computer Science Dept. University of Manchester 1992.

[Choppy 2004] Choppy C.: *Improving Use-Case Based Requirements*, in proceedings of FASE – Fundamental Approaches to Software Engineering 2002, pp. 244-261, Springer ISSN 0302-9743.

[Jorgensen, Sjoberg 2004] Jorgensen, M., Sjoberg, D.: *The Impact of Customer Expectation on Software Development Effort Estimates*. http://www.simula.no/departments/engineering/publications/ SE.4.Joergensen.2004.b.

[IFPUG 2002] International Function Points Users Group. *IT Measurement PractialAdvice from the Experts.*Addison-Wesley. Boston 2002. ISBN 0-201-74158-X.

[IFPUG 1999] IFPUG, *Fucntion Pointss Counting Practices Manual – Release 4.1*, International Function Points Users 1999, Wisconsin, USA.

[ISO/IEC 9126] ISO/IEC 9126: *Information technology – Software product quality – Part 1: Quality model.*

[ISO/IEC 2003] ISO/IEC 19761: 2003, *Software Engineering – COSMICS - FPP – A fucntional size method*.

[ITpapers 2003] *Quantitative Methods – Principles of Function Points*. 2003, www.itpapers.com.

[Jacobson et al. 1992] Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G., *Object-Oriented Software Engineering - A Use Case Driven Appro*ach, 1992, Addison Wesley ISBN 0-201-54435-0.

[Jones 2007] Jones, C.: Estimating Software Costs, McGraw-Hill, 2007. ISBN 000-714-8300-01.

[Karner 1993] Karner, G.: Use Case Points - Resource Estimation for Objectory Projects, Objective Systems SF AB (copyright owned by Rational Software), 1993.

[Knott, Merunka, Polak 2000] Knott, Roger P.; Merunka, Vojtěch; Polák, Jiří: *Process Modeling for Object Oriented Analysis using BORM Object Behavioral Analysis*, in Proceedings of Fourth International Conference on

Requirements Engineering ICRE 2000, Chicago 2000. IEEE Computer Society Press ISBN 0-7695-0565-1.

[Knott, Merunka, Polak 2003] Knott, Roger P.; Merunka, Vojtěch; Polák, Jiří: "*The BORM methodology: a third-generation fully object-oriented methodology*", Knowledge-Based Systems 3(10) 2003, Elsevier Science Publishing, New York.

[Liu et al. 2005] Liu L., Roussev B. et al: *Management of the Object-Oriented Development Process*, Virgin Island 2005, ISBN 1-59140-605-6.

[Longstreet 2004] Longstreet, D.: Fundamentals of Function Points Analysis, Software Development Magazine, 2004. www.math.unipd.it/~tulio/IS-1/2004/Approfondiemnti/Function_Point_Analysis.pdf

[Martin, Odell 1995] Martin James, Odell James J.: *Object-Oriented Methods - A Foundation*, Prentice Hall 1995, ISBN 0-13-63856-2.

[McGibbon 1997] McGibbon, T.: *Modern Empirical Cost andSchedule Estimation Tools*. August 20, 1997; www.dacs.dtic.mil/tech/estimation/costtools.pdf.

[Merunka 2004] Merunka, V. *Normalizace objektových databází*. Sborník konference OBJEKTY 2004. Praha 2004. ISBN 80-248-0672-X.

[Merunka, Pergl, Picka 2005] Merunka V., Pergl R., Pícka M., *Objektově orientovaná tvorba software*, ČZU Praha, 2005, ISBN 80-213-1159-2.

[Molnár 2000] Molnár, Z. *Efektivnost informačních systémů*. Grada Publishing 2000, Praha. ISBN 80-7169-410-X.

[Molnár 1992] Molnár, Z. *Moderní metody řízení informačních systémů*. Grada Publishing 1992. Praha. ISBN 80-85623-07.

[Miller, Williams 2000] Miller, S., Williams, R.: Function Points Made Easy. 2000. www.nps.navy.mil/wings/acq_topics/synopsis/archives/Winter%20%/functionpts.ppt

[Nierstrasz, Papathomas 1990] Nierstrasz Oscar, Papathomas Michael: Vie*wing Objects as Paterns of Communicating Agents*, pp 255-266 , in Object Management 1990, Centre Universitaire d' Informatique - Universite de Geneve, also at at ftp://cui.unige.ch/OO-articles/viewingObjectsAsPatterns.ps.Z.

[Ribu 2001] Ribu, K. *Estimating Object-Oriented Software Project with Use Cases.* Universitas Osloensis 2001. http://heim.ifi.uio.no/~kribu/oppgave.pdf.

[Rubin, Goldberg 1992] Rubin K.S., Goldberg A.: *Object Behavioural Analysis*, pp 48-62 Communications of the ACM 35(9) 1992.

[Rubin 1998] Rubin, D. M. *Uses of Use Cases. Methodologies and Practices – White Paper* 1998. Softstar Research, Inc. http://itpapers.com/whitepaper.aspx?&kw=use+case+Points&sortby =date&docid=23877

[Rumbaugh et al. 1991] Rumbaugh James, Blaha Michael, Premerlani William, Eddy Frederic, Lorensen William: *Object-Oriented Modelling and Design,* Prentice Hall 1991, ISBN 0-13-630054-5.

[Shiver, Wegner 1987] Shriver Bruce, Wegner Peter.: *Research Directions in OOP*, MIT Press 1987, ISBN 0-262-19264-0.

[Smith 1999] Smith, J.: *The estimation of Effort Based on Use Cases*. Rational Software Corporation 1999. http://www.bfpug.com.br/Artigos/UCP/Smith-The_Estimation_of_Effort_Based_on_Use_Cases.pdf.

[Software metrics 2002] *Estimating Effects and Test Cases.* http://softwaremetrics.com/Articles/defects.htm.

[Software metrics 2003] Longstreet, D.: *Estimating Software Development.* http://softwaremetrics.com/Articles/estimating.htm.

[SPR 2002] Software Productivity Research: Counting Rules for Function Points and Feature Points, www.spr.com

[Učeň et al. 2022] Učeň, P. a kol.: *Metriky v informatice. Jak objektivně zjistit přínosy informačního systému.* Grada Publishing 2002. Praha. ISBN 80-247-0080-8.

[UML 2004] *The Unified Modeling Language Documents*, http://www.uml.org, prosinec 2004.

[USC 2005] USC, *USC COCOMO II Version 2000.2 User Manual*. University of Southern California, 1999. http://sunset.usc.edu/research/COCOMOII/cocomo_main.html#downloads, Květen 2006.

[Vahalík 2004] Vahalík, T.: *Odhadujete pracnost projektu?*

www.komix.cz/upload/noviny2004c.pdf.

[Vaníček 2008] Vaníček, J.: Mezinárodní normy řady ISO/IEC 250xx pro jakost softwarového produktu, Linux+, 41 2008 (2), s. 68 – 74

[Vaníček 2007] Vaníček, J.: Software quality measures validation in Czech Republic, Agriculture Economics, 53, 2007(3), p. 94 – 100. ISSN 0139-570X.

[Vaníček 2006b] Vaníček, J.: Software and Data Quality, Agriculture Economics, 52, 2006 (3), pp. 138 – 146.

[Vaníček 2006b] Vaníček, J.: Software quality requirements, Agriculture Economics, 52, 2006 (4), p. 177 – 185.

[Vaníček 2006b] Vaníček, J. *Prvky pro měření jakosti softwaru a dat.* Tvorba software 2006. Ostrava 2006. ISBN 80-248-1082-4.

[Vaníček 2005a] Vaníček, J. *Požadavky na jakost softwaru.* Agrární perspektivy 2005 10.ročník. Praha 2005. ISBN 80-213-1372-2.

[Vaníček 2005b] Vaníček, J. *Measurement and rating of information systems quality. Part 1. Cocept, terminology and theoretical background*. PEF ČZU. Praha 2004.

[Vaníček 2004] Vaníček, J. *Měření a hodnocení jakosti informačních systémů.* PEF ČZU, 2004 Praha. ISBN 80-213-0667-X.

[Veselý 2005] Veselý, A.: *Úvod do umělé inteligence*. PEF ČZU Praha 2005. ISBN 80-213-1361-7.

[Volráb 2004] Volráb, O. *Fuzzy přístup v objektové databázi*. Sborník konference Objekty 2004. Praha 2004. ISBN 80-213-0682-3.

[Voříšek 2003] Voříšek, J. *Strategické řízení informačního systému a systémová integrace*. Management Press 2003. Praha. ISBN 80-85943-9.

[Wilkie 1993] Wilkie G.: *Object-Oriented Software Engineering*, Addison Wesley 1993, ISBN 0-201-6276-1.

[Yourdon 1994] Yourdon E.: *Object-Oriented System Design - An Integrated Approach*, Prentice Hall 1994, ISBN 0-13-176892-1.

6.3 Internetové zdroje

[1] sunset.usc.edu/research/COCOMOII/cocomo_main.html–stránky zaměřené na COCOMO 2.0

[2] www.codeproject.com – stránky zaměřené na tématiku odhadu složitosti

[3] www.cosmicson.com – The Common Software Measurement International Consortium.

[4] www.sparxsystems.com.au – stránky společnosti vyvíjející Enterprise Architect

[5] www.globaltester.com

[6] www.charismatek.com.au – stránky společnosti Charismatek zaměřující se odhad složitosti

[7] www.chispl.com – stránky společnosti Chispl

[8] www.ieee.org – Intitute of Electrical and Electronics Engineers, Inc.

[9] www.ifpug.com – International Function Points Users Group

[10] www.itpapers.com – stránky obsahující články z oblasti IT

[11] www.softwaremetrics.com – stránky tématicky zaměřené na softwarové míry

[12] www.softlookup.com – stránky umožňující stažení zkušebních verzí softwarových nástrojů

[13] www.spr.com – Software Productivity Research

[14] www.softwarequalitymethods.com

[15] www.softstarresearch.com

[16] www.softstarsystems.com

[17] www.technologyevaluation.com – stránky obsahující články z oblasti IT

[18] cs.wikipedia.org/wiki/Algoritmus

[19] scim.sbu.ac.uk/law/section1

[20] www.craftcase.com

6.4 Seznam použitých obrázků a tabulek

Použité tabulky

Použité obrázky

7 Přílohy

7.1 Charakteristika metody BORM

BORM lze charakterizovat pomocí následujících tří vlastností [Merunka, Pergl, Picka 2005]:

1) BORM je navržen jako metoda, která pokrývá všechny fáze vývoje softwaru. Velká pozornost je v BORMu věnována úvodním fázím projektu a postupům, jak najít objekty v zadaném problému a zkontrolovat jejich správnost. Techniky z těchto fází BORMu lze používat samostatně pro modelování procesů i takových systémů, které nemají přímý vztah k tvorbě softwaru.

2) BORM pro každou jednotlivou fází životního cyklu využívá v diagramech jen omezenou přesně určenou sadu pojmů. Předpokládá se totiž, že během projektování dochází k postupným přeměnám pojmů na jiné. Například ve fázi analýzy se nepoužívají pojmy jako agregace, jednoduchá či vícenásobná dědičnost, protože tyto pojmy jsou relevantní až pro implementaci. Naopak pojmy jako stav, přechod či asociace jsou používány během analýzy, ale ve fázi implementace, kdy se snažíme model přizpůsobit cílovému implementačnímu prostředí, se s nimi již nepracuje. Nejde jen o postupné zvyšování úrovně detailu ve vytvářeném modelu, ale skutečně o řadu transformací modelu v průběhu životního cyklu.

3) V BORMu je každý pojem reprezentován shodnými symboly bez ohledu na to, zda se jedná např. o diagramy datové struktury nebo komunikaci mezi objekty. BORM používá pro znázorňování konceptuálních a softwarových pojmů většinu symbolů shodně s jazykem UML, ale dovoluje v jednom diagramu znázornit například posílání zpráv mezi metodami různých objektů v různých stavech. Tento přístup dovoluje vyjádřit konzistentním způsobem některé žádoucí detaily softwarové konstrukce, které lze výhodně aplikovat především při návrhu pro čistě objektově orientované programovací jazyky. Tento originální způsob nahrazuje tvorbu více od sebe oddělených třídních, stavových a kolaboračních diagramů a také dovoluje zobrazit větší množství spolu souvisejících informací. Samostatné stavové či iterační diagramy jsou však samozřejmě v BORMu také používány.

BORM rozlišuje 6 fází životního cyklu vývoje systému [Carda, Merunka, Polák 2003]:

1) Strategická analýza. Zde dochází k vymezení samotného problému, je stanoveno jeho rozhraní, jsou rozpoznány základní procesy, které se v systému a také v jeho okolí mají odehrávat.

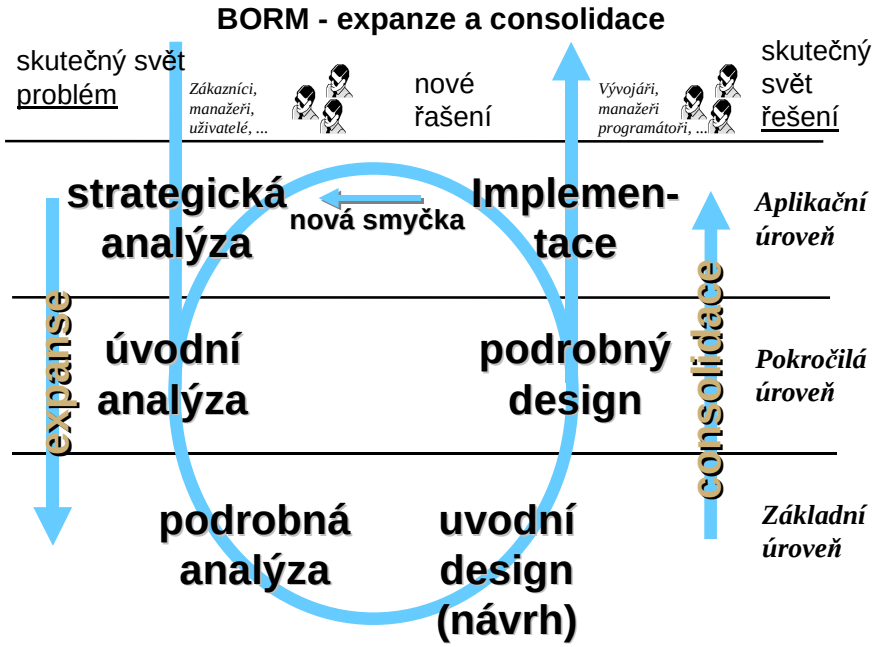
2) Úvodní analýza. Zde dochází k rozpracování samotného problému, jsou mapovány požadované procesy v systému a vlastnosti základních objektů, které se na diskutovaných procesech podílejí.

3) Podrobná analýza. Je rozpracování analýzy do detailů jednotlivých typů objektů (sady objektů, třídy objektů) a objektových vazeb (skládání, dědění, závislosti, …).

4) Úvodní návrh (design). Je to první fáze, ve které se začínáme snažit systém upravit tak, aby byl schopen softwarové implementace. Proto se zde již nehovoří o analýze, neboť z pohledu zadání by mělo již vše být hotovo a rozpoznáno. Úvodní návrh používá shodné nebo velmi podobné nástroje jako předchozí fáze, ale liší se způsobem práce s nimi.

5) Podrobný návrh (design). V této fázi dochází k přeměně prvků již existujícího modelu do takové podoby, která je podřízena cílovému implementačnímu prostředí. V této fázi se zohledňují vlastnosti konkrétních programovacích jazyků, databází apod.

6) Implementace (tvorba, sestavování programu). V této fázi se vytváří (programuje, sestavuje či generuje z CASE nástroje) požadovaný software.



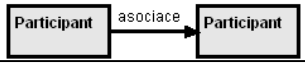
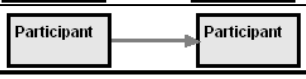
Obr. 45. 6 fází životního cyklu vývoje systému v BORMu. [Carda, Merunka, Polák 2003]

Metoda BORM pokrývá při svém použití dvě úrovně návrhu IS. Jde o tzv. business analýzu IS a konceptuální analýzu IS. Business analýza IS se zaměřuje na mapování současných a budoucích procesů, které popisuje pomocí funkcí, scénářů, architektury a procesních diagramů. Konceptuální analýza navazuje na výsledky business analýzy tím, že transformuje business (procesní) model na zadání pro programátory, které popisuje prostřednictvím diagramu objektů a tříd, diagramu softwarových objektů a diagramu softwarových komponent.

Tab. 78. Vysvětlení pojmů užívaných v metodě BORM. [Carda, Merunka, Polák 2003]

Architektura	Poskytuje komplexní model systému, který se skládá z několika vrstev, které mohou být samy o sobě jednostranně zaměřeným modelem systému. Je to například vrstva procesů, logický model (popis dat, funkcí a pravidel) a model komponent (např. softwarová aplikace nebo organizační struktury). Prvky z různých vrstev architektury jsou mezi sebou vzájemně provázány, a proto je vhodné při každé změně provést rozbor dopadů na prvky jiných úrovní.
Aktivita	Aktivity představují jednotlivé části chování business objektů tak, jak byly rozpoznány technikou OBA. V procesních diagramech se pomocí vybraných aktivit provádějí přechody mezi stavy objektů. Aktivity různých objektů mezi sebou komunikují. Z aktivit se odvozují metody.
Procesní (Business) diagram	Tento diagram představuje obraz všech možných průběhů procesu pomocí současného zobrazení dvou dimenzí tohoto problému. První dimenzí jsou role (=průběh aktivit jednoho objektu v procesu) zúčastněných objektů jako automaty se stavy a přechody. Druhou dimenzí je sled komunikací mezi objekty, který představuje řídící a datové toky mezi objekty v procesu.
Datový tok	Data, která si objekty vyměňují při komunikacích nebo posílání při zpráv, rozlišují se parametry zprávy a návratové hodnoty.
Funkce	Nejjednodušší slovní popis požadovaných procesů v systému podle OBA. Pro pozdější bezproblémovou komunikaci analytiků a zadavatelů je vhodné do tohoto seznamu zahrnout a zvlášť vyznačit i funkce, které popisují, co se modelovat nebude. Ze seznamu funkcí se odvozuje scénář.
Komunikace	Řízení aktivit business objektů. Komunikace je abstrakce zpráv mezi objekty. V pozdějších fázích modelování se zprávy z komunikací odvozují.

Scénář	Scénář systému je podrobnější popis procesu v technice OBA. U scénáře se zvlášť popisuje začátek procesu, vlastní akce procesu, seznam participantů a konec procesu. Mezi scénáři může být hierarchie skládání procesů, hierarchie nadtypů a podtypů a časová návaznost procesů na sebe, pro které se používají stejné značky jako pro objektové diagramy (UML, ORD).
Stav	Stav představuje konkrétní konstelaci automatu v čase. Pokud automat přijme nějakou informaci, může to vyvolat přechod z jednoho jeho stavu do druhého. V BORMu se nahlíží na objekty v procesech jako na automaty, které v různých stavech mohou provádět různé aktivity (role objektu v procesu).

pojem	grafický symbol	význam
Začátek role		Je to začátek sledu činností nějaké role nějakého objektu v procesu.
Konec role		Je to ukončení sledu činností nějaké role nějakého objektu v procesu.
KDO roli vykonává = participant		Participant je účastník procesu, který má v procesu aktivity.
CO se v roli dělá = aktivita		Podle BORMu je každá činnost někým prováděna. Aktivita vyjadřuje aktivní i pasivní (jiným participantem vyvolanou) činnost.
KDY se něco děje = stavy participantu		Vyznačuje místo v čase, kde se čeká nebo něco provádí.
Komunikace		Řídící sled mezi aktivitami objektů. Přeškrtnutím spojení se může vyznačit podmínka, kdy komunikace platí.
Datový tok		Vyměňovaná data, materiál, informace, peníze, …
Přechod mezi stavy		Návaznost v čase od jednoho stavu ke druhému. Přeškrtnutím spojení se může vyznačit podmínka, kdy přechod platí.
VZTAH mezi účastníky procesu = asociace.		Vyjadřuje napojení nebo jinou logickou souvislost (např. vlastnictví, závislost, …) mezi participanty.
Hierarchie účastníků = „IS-A“ taxonomie.		Používá se, když je třeba vyznačit, že jeden druh participantů je zvláštním případem jiného druhu.

Obr. 46. Notace business diagramů v metodě BORM

7.2 Úvod do metody AHP

Tato kapitola představuje teoretické základy metody Analytického hierarchického rozkladu. Ty byly použity v následující kapitole 4.1.1.2 prakticky pro výpočet potřebných vah a jejich koeficientů.

V prvním kroku je nutné nastavit vztahy mezi kritérii (třetí úroveň hierarchie na obrázku 47). Pro tento krok se používá tzv. Saatyho metoda.

A. **Saatyho metoda** [Brožová, Houška, Šubrt 2003]

Jde o metodu kvantitativního párového srovnání kritérií. Pro hodnocení párových porovnání se používá 9-ti bodové stupnice a je možné používat i mezistupně (hodnoty 2, 4, 6, 8):

1 – rovnocenná kritéria *i* a *j*,

3 – slabě preferované kritérium *i* před *j*,

5 – silně preferované kritérium *i* před *j*,

7 – velmi silně preferované kritérium *i* před *j*,

9 – absolutně preferované kritérium *i* před *j*.

Expert porovná každou dvojici kritérií a velikosti preferencí *i*-tého kritéria vzhledem k *j*-tému kritériu zapíše do Saatyho matice $S=(s_{ij})$:

$$S = \begin{pmatrix} 1 & s_{12} & \dots & s_{1n} \\ 1/s_{12} & 1 & \dots & s_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 1/s_{1k} & 1/s_{12} & \dots & 1 \end{pmatrix}$$

Jsou-li *i*-té a *j*-té kritérium rovnocenná, je $s_{ij} = 1$, preferuje-li slabě *i*-té kritérium před *j*-tým, je $s_{ij} = 3$, preferuje-li silně *i*-té kritérium před *j*-tým, je $s_{ij} = 5$, při velmi silné preferenci *i*-tého kritéria je $s_{ij} = 7$, při absolutní preferenci *i*-tého kritéria je $s_{ij} = 9$. Je-li preferováno *j*-té kritérium před *i*-tým, zapíší se do Saatyho matice převrácené hodnoty ($s_{ij} = 1/3$ při slabé preferenci, $s_{ij} = 1/5$ při silné preferenci, atd.)

Matice je čtvercová řádu $n \bullet n$, reciproční, tj. platí, že $s_{ij} = 1/ s_{ij}$ a vyjadřuje vlastně odhad podílů vah *i*-tého a *j*-tého kritéria. Na diagonále Saatyho matice jsou vždy hodnoty jedna (každé kritérium je samo sobě rovnocenné).

Prvky této matice nebývají většinou dokonale konzistentní, tzn. že neplatí $s_{hj}=s_{hi} \bullet s_{ij}$ pro všechna $h, i, j = 1, 2, \dots,$ n. Kdybychom sestavili matici $V = (v_{ij})$, jejíž prvky by byly skutečné podíly vah $v_{ij}= v_i/v_j$, pro prvky této matice by výše uvedená podmínka platila. Míra konzistence se měří například indexem konzistence, který Saaty definoval jako

$$I_S = \frac{l_{\max} - n}{n-1}, \text{ kde}$$

l_{max}…je největší vlastní číslo Saatyho matice

n…je počet kritérií.

Saatyho matice je považována za dostatečně konzistentní, jestliže $I_S<0,1$.

Váhy v_j by se daly odhadnout z podmínky, že matice S by se měla co nejméně lišit od matice V. V obvyklém pojetí by to znamenalo minimalizovat součet čtverců odchylek stejnolehlých prvků obou matic. Pro jejich výpočet by pak bylo nutno vyřešit optimalizační model:

$$F = \sum_i \sum_j \left[s_{ij} - \frac{vi}{vj} \right]^2 \rightarrow \min$$

za podmínky $\sum_{j=1}^{n} v_j = 1$. Jde však o model nekonvexního kvadratického programování, což způsobuje výpočetní potíže.

Saaty proto navrhl několik početně velmi jednoduchých způsobů, pomocí kterých lze odhadnout váhy v_j. Nejčastěji se používá postup výpočtu vah jako normalizovaného geometrického průměru **řádků** Saatyho matice (metoda logaritmických nejmenších čtverců). Vypočteme hodnoty b_i jako geometrický průměr řádků Saatyho matice:

$$b_i = \sqrt[n]{\prod_{j=1}^{n} s_{ij}} .$$

Váhy se potom vypočtou normalizací hodnot b_i

$$v_i = \frac{b_i}{\sum_{i=1}^{n} b_i} .$$

Případy, kdy je Saatyho matice nekonzistentní, jsou velmi časté zvláště u rozsáhlejších úloh. Nekonzistence může být způsobena chybou při zadávání odhadů poměrů vah, když expert neprováděl žádnou kontrolu svých odhadů. V tomto případě je nutné na základě odhadu vah překvantifikovat Saatyho matici tak, aby splňovala požadavek konzistence, a poté provést nový odhad vah. Tímto interaktivním způsobem lze dospět k velmi solidním výsledkům.

Saatyho metodu je možné využít nejen ke stanovení preferencí mezi kritérii, ale i mezi variantami, a to pomocí analýzy původní úlohy, která je přepsána pomocí hierarchického uspořádání.

B. Metoda AHP [Brožová, Houška, Šubrt 2003]

AHP je metodou rozkladu složité nestrukturované situace na jednodušší komponenty, vytváří tedy hierarchický systém problému. Tento hierarchický systém je zobecněním – rozšířením možností vícekriteriálního rozhodovacího systému. Na každé úrovni hierarchické struktury se použije Saatyho metoda kvantitativního párového porovnání. Pomocí subjektivních hodnocení párového porovnání pak tato metoda přiřazuje jednotlivým komponentám kvantitativní charakteristiky vyjadřující jejich důležitost. Syntézou těchto hodnocení se pak stanoví komponenta s nejvyšší prioritou, na níž se rozhodovatel zaměří s cílem získat řešení rozhodovacího problému.

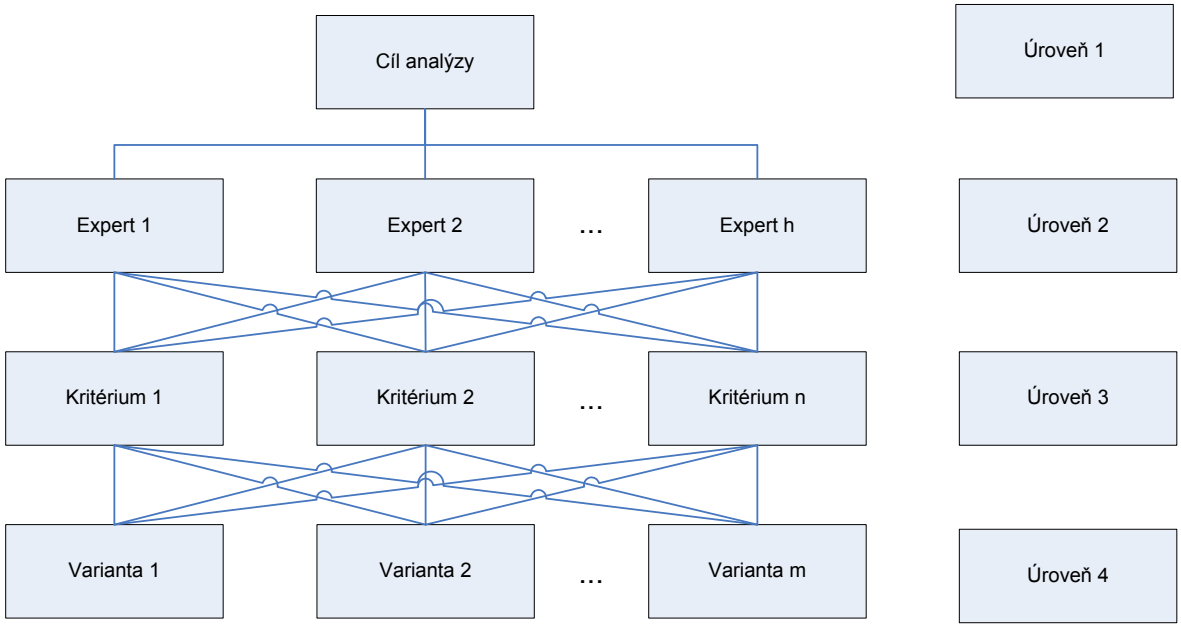
Metodu je možné použít pro jakýkoliv typ informace o preferenčních vztazích mezi komponentami modelu. Jedinou podmínkou je, aby uživatel uměl z této informace určit směr a intenzitu preference mezi všemi páry porovnávaných komponent.

Pod pojmem hierarchická struktura se rozumí lineární struktura obsahující několik úrovní, přičemž každá z nich obsahuje několik prvků. Uspořádání jednotlivých úrovní hierarchické struktury odpovídá uspořádání od obecného ke konkrétnímu. Čím obecnější jsou prvky ve vztahu k danému rozhodovacímu problému, tím zaujímají v jemu příslušející hierarchii vyšší úroveň a naopak. Intenzity vzájemného působení jednotlivých prvků v hierarchii mohou být určitým způsobem kvantifikovány. Nejvyšší úroveň hierarchie obsahuje vždy pouze jeden prvek, který definuje cíl vyhodnocování nebo analýzy. Tomuto prvku lze přiřadit hodnotu jedna, která je potom rozdělena mezi prvky na druhé úrovni. Podobně se hodnota každého prvku dělí i na dalších nižších úrovních hierarchie, až dostaneme ohodnocení prvků nejnižšího stupně – variant.

Typická jednoduchá úloha vícekriteriální analýzy variant obsahuje následující úrovně:

- Úroveň 1 – cíl vyhodnocování, kterým může být uspořádání variant,
- Úroveň 2 – kritéria vyhodnocování,
- Úroveň 3 – posuzované varianty.

Složitější úlohy obvykle mají mezi kritérii a variantami ještě úroveň subkritérií. Úlohy, na jejichž hodnocení se podílí více hodnotitelů, mají mezi cílem a kritérii ještě úroveň hodnotitelů (expertů), jejich hodnocení (váhy) označují míru jejich fundovanosti (viz obrázek 47).



Obr. 47. Hierarchická struktura úlohy vícekriteriální analýzy variant pro hodnocení více experty.

[Brožová, Houška, Šubrt 2003]