METODY ODHADU PRACNOSTI ZALOŽENÉ NA MODELU

Ing. Zdeněk Struska
Katedra informačního inženýrství, PEF ČZU Praha,
struska@pef.czu.cz

Ing. Robert Pergl
Katedra informačního inženýrství, PEF ČZU Praha,
pergl@pef.czu.cz

ABSTRAKT:
Příspěvek se snaží představit vybrané techniky, které se mohou jednoduše aplikovat v úvodních fázích vývoje projektu. Ač v současné době existuje dostatečná nabídka metod, nabízejících svým uživatelům alespoň orientační odhad pracnosti (složitosti) a s ní spojenou nákladovost vývoje informačních systémů, přesto je stále jejich funkce v procesu vývoje IT podceňována.

KLÍČOVÁ SLOVA:
Složitost, pracnost, odhad složitosti, Analýza function points, Analýza feature points, Use Case Points, nevyrovnaná váha aktoru, nevyrovnaná váha Use Case, technický faktor, faktor prostředí, vyrovnaný počet Use Case.

ABSTRACT:
The paper tries to introduce chosen techniques, which could be applied successfully and easily. Presently sufficient offer of the methods exists, which provide to their users at least checking estimation of the complexity and the costs of the information systems development. Their function is still underestimated in the process of IT development.

KEY WORDS:
Complexity, Estimation of complexity, Function Points Analysis, Feature Points Analysis, Use Case Points, unadjusted actor weight, unadjusted Use Case weight, Technical factor, Environmental factor, adjusted Use Case Points.

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.

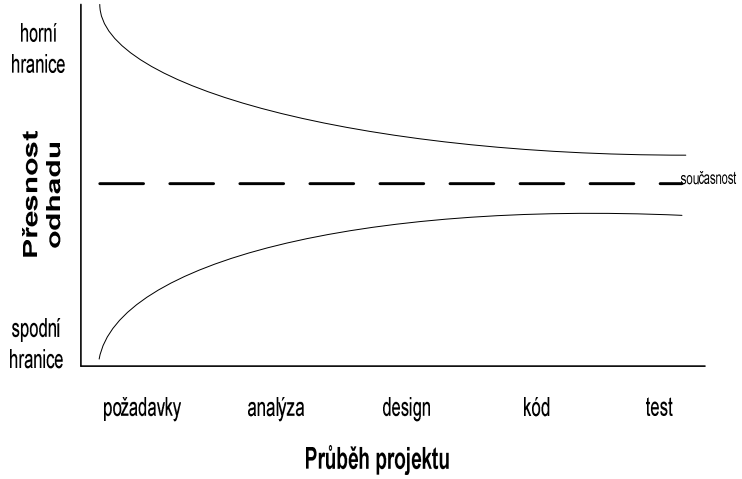
Pro odhad svých šancí na úspěch, musí znát náklady, za které je schopna své produkty vyrábět nebo nabízet. Tato teze platí pro všechny firmy obecně, tzn. i pro ty, které se zabývají tvorbou software. Stejně jako si výrobní firma před začátkem výroby nového či reorganizace výroby existujícího produktu 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 přes odhad 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.

Metody, které článek představuje, se pokouší o alespoň orientační odhad, který umožní přibližný odhad nákladovosti softwaru. Na počátku projektu neexistuje žádný jiný způsob, jak s nákladovostí projektu pracovat. S odhadem, který vznikne na začátku projektu, se dále pracuje a dochází k jeho zpřesňování na základě rozšiřování vstupních informací do vybraných metod.

ODHAD PRACNOSTI

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



Obr. 1: Zpřesňování odhadu pracnosti v průběhu projektu [2]

ODHADY ZALOŽENÉ NA MODELU

A. Analýza Function Points

Jde o nejznámější metodu pro odhad pracnosti, která vytvořila základ pro další výzkum v této oblasti. Společným rysem metod funkčních jednic je to, že složitost odhadují jako součin dvou faktorů. Prvý faktor je založen na **funkcích**, které jsou od produktu vyžadovány, druhý na **podmínkách**, které pro vytvoření produktu objektivně jsou.

Popis výpočtu včetně zhodnocení jejích výhod a nevýhod této metody a porovnání jejích jednotlivých verzí je k dispozici v [11], [12].

B. Analýza Feature Points

Jelikož metoda function points při použití na objektově-orientované systémy neposkytovala dostatečně přesné výsledky, byla vyvinuta její modifikace, jenž měla tento nedostatek odstranit. Vznikla metoda feature points, která bere v úvahu navíc ještě šestý parametr – algoritmus (respektive odhad jeho složitosti). Ten je rozšířením původních pěti parametrů (external inputs, external outputs, external inQuiry, internal logical files, external interface files) metody function points. Metoda feature points také omezuje empirické váhy vnitřních logických souborů (z původní hodnoty 10 na hodnotu 7).

Porovnání dvou verzí metod feature a function points je k dispozici v [12].

C. Use Case Points

Use case points vychází stejně jako výše zmíněné metody z teze, že funkčnost systému z pohledu uživatele je základem pro odhad velikosti informačního systému. Je orientována na objektový přístup k návrhu softwaru.

Nejprve zjistíme počet aktorů a počet use case. Specialitou metody je, že ignoruje případy extend nebo include. Přiřadíme kategorie složitosti (jednoduchý – průměrný – složitý) jednotlivým aktorům a spočteme jejich váhy. Dále zařadíme use case do kategorie složitosti (jednoduchý – průměrný – složitý) podle počtu transakcí (ten může být odvozen z počtu kroků popsaných ve scénáři). Součtem hodnot složitosti z obou kroků dostáváme tzv. nevyrovnanou část use case points. Celkovou část use case points získáme roznásobením nevyrovnané části s hodnotou technického faktoru a faktoru prostředí. Podrobný postup výpočtu je představen v následující kapitole.

ODHAD PRACNOSTI POMOCÍ METODY USE CASE POINTS

Metoda byla vyvinuta v roce 1993 Gustavem Karnerem. Je založena na analýze function points, která vznikla v 70. letech ve výzkumných laboratořích IBM, za jejíhož autora je pokládán Allen Albrecht [1].
Modely use case mohou být jednoduše použity pro odhad složitosti vývoje softwaru a jeho testování. Srozumitelnost use case modelu ovlivňuje jeho struktura.

Výpočet use case points se skládá ze třech částí:

- Nevyrovnaná část UCP.
  - Počet aktorů.
  - Počet use case.
- Technický faktor.
- Faktor prostředí.

1. Nevyrovnaná 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 interface na měřený systém prostřednictvím různých programů (např. standardní aplikační program (nebo API)).

Průměrný – buď další systém, který je na měřený systém napojen prostřednictvím protokolu nebo textu založeném na uživatelském rozhraní. Průměrný aktor spolupracuje se systémem přes protokol (např. http, tcp/ip, atd.) nebo jiným typem aktoru může být datový sklad. Tyto typy aktorů kvalifikujeme jako průměrné, pokud je výsledky spouštěcích testů nutno ověřit manuálně (tzn. aplikace spolupracují automaticky a jejich kontrola se provede manuálně spouštěním sql příkazů uložených v paměti (bufferu)).

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

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

Tabulka 1: Kategorie – počet aktorů

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

Typ aktora	Váha aktora	Počet aktorů	Celkem
Jednoduchý	1	__ x 1 =	
Průměrný	2	__ x 2 =	
Složitý	3	__ x 3 =	
Nevyrovnaná váha aktoru celkem (Total unadjusted actor weight – uaw)			

Tabulka 2: Výpočet nevyrovnané váhy aktoru.

b) *identifikace, klasifikace a přidělení vah use case (unadjusted Use Case weights – uucw)*

Dalším krokem je určení počtu use case v systému. Use case by měly být přiděleny jakési váhy závislé na počtu transakcí a/nebo scénářů. Nejprve rozdělíme zjištěné use case do kategorií:

jednoduchý – průměrný – složitý.

Rozdělení se provede na základě počtu transakcí, konkrétní počty transakcí, které jsou hraničními hodnotami mezi definovanými kategoriemi, jsou v tabulce 3.

Základem kategorizace je, kolik transakcí use case obsahuje, jak je transakce definovaná – jako základní množina aktivit, které jsou buď vykonávány zcela nebo vůbec.

Typ Use Case	Definice	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

Tabulka 3: Souhrn definicí.

Po rozdělení use case dle počtu transakcí mezi tři kategorie, přidělíme jednotlivým úrovním jakousi váhu a provedeme roznásobení, dále součty přes jednotlivé typy use case. Výsledkem tabulky je nevyrovnaný počet use case (viz. tabulka 4).

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

Tabulka 4: Výpočet nevyrovnané váhy use case celkem.

c) Výpočet nevyrovnané části UCP

Celkový počet nevyrovnaných use case points získáme sečtením dvou výše vypočtených částí – nevyrovnaná váha aktoru (uaw) a nevyrovnaná váha use case (uucw).

$$unadjusted\ use\ case\ points\ (uucp) = uaw + uucw$$

2. Technický faktor

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.

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

Tabulka 5: Výpočet hodnoty technického faktoru

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

$$tcf = 0,6 + (0,01 * tFactor)$$

3. Faktor prostředí

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).

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

Tabulka 6: Seznam faktorů prostředí

Po ohodnocení faktorů se jim přidělené hodnoty roznásobí jakýmisi 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 (Environmental complexity factor – ecf) z něhož získáme výslednou hodnotu faktoru prostředí.

$$ef = 1,4 + (-0,03 * eFactor)$$

4. Celkový počet UCP

Pro celkový počet UCP dosadíme výše vypočtené části do vzorce UCP a roznásobením nevyrovnané části UCP technickým faktorem a faktorem prostředí získáme vyrovnaný počet UCP (adjusted use case points – aucp).

$$aucp = uucp * tcf * ef$$

V postupu výpočtu představeném v článku jsme se dopracovali k odhadu pracnosti vyvíjeného IS. Pracnost je zde definovaná bezrozměrným číslem, které vzejde ze vzorce pro vyrovnaný počet UCP. Abychom se u metody UCP překlenuli od složitosti ke skutečné pracnosti, roznásobíme výslednou hodnotu stanoveným počtem člověkodnů nebo můžeme pracovat s pracností rozdělenou do intervalů. V tomto případě vypočteme UCP pro vybrané a krajní hodnoty intervalu.

Stanovený počet člověkodnů 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 20 člověkodnů na jednu *aucp*.

SROVNÁNÍ METOD

Srovnání v článku představených metod je komplikované. Důvod je prostý, metoda UCP pracuje se vstupy ještě před fází, kdy se rozhoduje o složitosti algoritmů potřebných pro výpočetní úkony programu. Nedá se tedy jednoznačně určit, jestli vykazuje lepší nebo horší výsledky při vyšším výskytu algoritmů a logických souborů či naopak, jak je tomu u prvních dvou metod – function a feature points.

Ty jsou vzájemně srovnatelné, pracují totiž s počtem algoritmů a s logickými datovými soubory.

Metody function a feature point generují [12]:

- stejné početní výsledky u aplikací, kde je stejný počet algoritmů a logických souborů,
- rozdílné početní výsledky u aplikací, kde je větší počet algoritmů než logických souborů, ačkoliv se nejedná o neobvyklý systémový software, generuje metoda feature point „přesnější“ výsledky (např. MIS),
- menší počet algoritmů než logických souborů, které jsou běžné pro některé informační systémy, generuje metoda function point „přesnější“ výsledky (např. real time software).

Obecně lze doporučit paralelní aplikaci představených metod na konkrétních projektech a zpřesňování výsledků v průběhu životního cyklu projektu.

ZÁVĚR

Domníváme se, že s rostoucí složitostí vznikajících softwarových nástrojů, bude růst význam odhadů složitosti, ač se jedná pouze o orientační.

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

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

Výsledné rozhodnutí, jak bylo nastíněno v předcházejících kapitolách, je možné v průběhu projektu přehodnocovat v závislosti na přibývajících vstupních datech, které umožňují zpřesňovat počáteční odhady.

Každý ví, že zdroje firem jsou 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.

LITERATURA

[1] Albrecht, A. J. and Gaffney, J. E., Jr. Software Functions, Source Line sof Code and Development Effort Prediction. A Software Science Validation, 1983 IEEE Transactions on Software Engineering (TSE) 9, no. 6 (November 1983)
[2] Estimating Effects and Test Cases. http://softwaremetrics.com/Articles/defects.htm

[3] International Function Point Users Group: IT Measurement. Practical Advice from the Experts. Addison-Wesley. Boston 2002. ISBN 0-201-74158-X.
[4] International Funtion Point Users Group. www.ifpug.com
[5] Longstreet, D.: Estimating Software Development.
http://softwaremetrics.com/Articles/estimating.htm
[6] M. Jorgensen, D. Sjoberg: The Impact of Customer Expectation on Software Development Effort Estimates
[7] Quantitative Methods – Principles of Function Points. www.itpapers.com
[8] Software Productivity Research. www.spr.com
[9] Struska, Z, Vaníček, J. Measurement and rating of information systems quality. Part 3: Design Komplexity and Software Engineering Consequences. PEF ČZU, 2005, Praha.
[10] Struska, Z., Vaníček, J. Measurement and rating of information systems quality. Part 2: Quality Model.PEF ČZU, 2005, Praha.
[11] Struska, Z.. Porovnání vybraných přístupů aplikace funkčních jednic. Agrární perspektivy 2005 10. ročník. Praha 2005. ISBN 80-213-1372-2.
[12] Struska, Z. Metody odhadu složitosti v objektovém prostředí – Functin a Feature Point. Objekty 2005. Ostrava 2005. ISBN 80-213-0682-3
[13] Vahalík, T.: Odhadujete pracnost projektu? www.komix.cz/upload/noviny2004c.pdf
[14] Vaníček, J.: Měření a hodnocení jakosti informačních systémů. PEF ČZU. Praha 2004. . ISBN 80-213-0667-X.
[15] www.globaltester.com
[16] www.cosmics.com
[17] www.softwaremetrics.com