ODHAD SLOŽITOSTI SOFTWAROVÉHO PRODUKTU V ETAPĚ SPECIFIKACE POŽADAVKŮ

Ing. Josef Pavlíček Ph.D,
Sun Microsystems, Evropská 33e,
Praha 6 Dejvice 165 00.
Josef.Pavlicek@sun.com

ABSTRAKT:

Práce se zabývá navržením manažerských charakteristik vývoje informačních systémů v etapě specifikace požadavků. Jde především o vymezení pojmu celková složitost softwarového produktu (CSP). Ta je odhadována na základě funkční specifikace na software, případů užití, scénářů a faktorů ovlivňující vznik software. Odhad Celkové Složitosti softwarového Produktu se stává klíčový pro manažerská rozhodnutí typu zda realizovat softwarový projekt, jehož výsledkem je hotový softwarový produkt, či jaké zdroje pro vývoj alokovat, jak vytvořit časový plán pro jeho vývoj a pro úvahy o ceně výsledného produktu.

KLÍČOVÁ SLOVA:

Softwarové inženýrství, Metoda funkčních jednic, Metoda bodů Případu užití, Složitost, Projekt, Případ užití, Diagram užití, Unified Process, Rational Unified Process, Unified Model Language, Perceptron, Neuron, Neuronová síť, Zpětné šíření chyby, Jakost, Atribut, PETRA

Úvod

Cílem tohoto příspěvku je ukázat novou metodu odhadu složitosti softwaru, která by byla použitelná již v etapě specifikace požadavků na software – tedy v okamžiku, kdy už jsou známy požadavky na funkčnost požadovaného softwaru, ale ještě před tím, než začneme softwarový produkt programovat. Umět odhadovat složitost softwarového produktu, znamená vědět, jak řídit vznik softwarového produktu.

Navržené odhady jsou získávány postupy, které byly inspirovány výsledky mezinárodního vědeckého týmu SQuaRE, který navrhl normy pro hodnocení jakosti softwaru. Kromě nepochybné souvislosti mezi jakostí a složitostí, tedy i pracností softwaru je zde také důvod, proč se jakostí zabývat a modelem hodnocení jakosti se inspirovat.

Oba tyto důležité atributy software a systémů obsahujících software je totiž velmi důležité odhadovat co nejdříve, jakmile je to jen možné. Nejlépe již v etapě specifikace. Pro odhady (tak zvané prediktory) potřebujeme z podkladů, které jsou v dané etapě k dispozici, určitou množinu vstupních dat, tak zvaných primitiv pro měření, neboli základních měr. Tyto množiny sice nemusí být shodné, do značné míry se však překrývají.

Pro měření a hodnocení jakosti je k dispozici model podpořený mezinárodními normami a množstvím postupů, jak jakost hodnotit a měřit, viz [1]. Pro odhady jsou navrženy také vnitřní míry. Pro složitost takový model dosud není a ani postupy pro odhad nejsou popsány. Nabízí se tedy cesta pokusit se to, co je vytvořeno pro odhad jakosti, modifikovat tak, aby to bylo použitelné i pro odhad složitosti

Odhad složitosti softwarového produktu

Vymezení pojmu etapa specifikace požadavků

Pod pojmem „etapa specifikace požadavků“ chápejme časový interval od zadání prvních požadavků na software, až po okamžik, kdy je již funkční specifikace na software hotová a schválena zadavatelem. Tudíž známe uživatelské požadavky na chování a funkčnost požadovaného softwarového produktu. Protože však různé metody řízení tvorby softwaru, jako je například metoda Unified Process - viz [2] nebo BORM - viz [3], mají různý počet fází (UP má čtyři) a v jednotlivých fázích může být různé množství různých etap, na jejichž konci jsou nějakým způsobem shrnuty požadavky na vznikající software, navrhuji tento časový interval obecně nazývat etapa specifikace požadavků.

Pro účel metody odhadu složitosti pak navrhuji využít metody řízení tvorby softwaru Unified Process, která má čtyři fáze a etapa specifikace požadavků na software začíná v první fázi Unified Processu – ve fázi Založení a končí v druhé fázi nazývané Rozpracování požadavků na software. V metodě Unified Process se tato etapa nazývá Requirements, což lze přeložit jako seznam požadavků či analýza požadavků.

Běžné problémy při odhadu složitosti

Analýza dostupných zdrojů jednoznačně ukazuje dva trendy v odhadu složitosti softwaru. Buď je odhad složitosti založen na funkcích, které jsou od vznikajícího softwaru požadovány a na podmínkách, které pro vytvoření produktu objektivně jsou. Jde o metody odhadu složitosti na základě funkčních jednic. Nebo jde o metody odhadu, které operují s již hotovým zdrojovým kódem. Odhad složitosti pak provádějí na základě informací dostupných ze zdrojového kódu.

Provedl jsem analýzu především metod Function point a Use Case point. Tyto metody jsou založené na funkčních jednicích pracujících s ordinálními stupnicemi. Porušují pravidla operací s hodnotami naměřenými v ordinální stupnici. Z tohoto důvodu musí být tyto metody odhadu vždy do určité míry nepřesné. Ač se autoři metod snaží korigovat výsledky metod pomocí různých opravných koeficientů, zaváděním konstant atd., nemůže být jejich snaha odměněna dostatečně dokonalou metodou.

Metody Funkčních jednic nijak nezatracuji. Je jistě vhodné a pravděpodobně jediné možné využívat funkčních jednic k odhadu složitosti. Domnívám se však, že je potřeba se pečlivě vyhýbat jakýmkoliv operacím, které budou pro hodnoty naměřené v ordinální stupnici neinterpretovatelné.

Odhad složitosti, prováděný na základě znalosti zdrojového kódu či strukturovaného zadání, je smysluplný. Pokud se podaří zajistit kvalitní tým programátorů, jenž bude programovat ukázněně za předem domluvených konstrukcí, bude možné odhad složitosti provádět.

Problém spočívá v tom, že to bude již pozdě. Odhad složitosti je potřeba provádět co nejdříve. Na to ale nemáme potřebný zdrojový kód.

Naopak, odhad složitosti na základě grafu řízení, který může být zobrazením scénáře zadání, se mi zdá být smysluplný. I zde samozřejmě platí, že scénář může být napsán úmyslně zjednodušeně. Jeho zobrazení grafem řízení bude jednoduché, složitost bude vycházet nízká.

Důležitým poznatkem, který jsem při analýze dostupných zdrojů získal je:

Na složitosti softwarového návrhu se zásadně projevuje vliv prostředí. Proto, vzniká-li jakákoliv nová metoda odhadu složitosti, měla by už ze své podstaty s vlivem prostředí počítat a nějak jej zohlednit ve svých výpočtech.

Kdy složitost odhadovat

Odhadnout čas a náklady na softwarový projekt je účelné pokud možno co nejdříve. To je v tom okamžiku, kdy již máme dostatečně dobře shrnuty požadavky na software. Díky překrývání jednotlivých etap ve fázích dle UP - viz [4] není možné zcela přesně říci, jestli k tomu dochází v první (Založení) či druhé fázi (Rozpracování požadavků na software) životního cyklu projektu. Dokud však nezačneme „naplno programovat“ je vhodné složitost budoucího software nějak odhadnout. Nejprve je ale nutné vymezit pojem „složitost“.

Složitost obecně je viz [1,2] „chápána jako míra úsilí, které potřebuje člověk vynaložit k tomu, aby produkt navrhl, vyvinul, implementoval, provozoval a udržoval v průběhu jeho životního cyklu“.

Za účelem odhadu složitosti softwarového produktu navrhuji chápat složitost jako míru úsilí, které potřebuje člověk vynaložit k tomu, aby produkt navrhl a vyvinul. Přičemž pod pojmem vyvinul budeme chápat stav, kdy bylo dosaženo zvoleného cíle. V případě tvorby programu pro evidenci knih v knihovně budeme pod pojmem vyvinul chápat okamžik, kdy je program:

- naprogramován,
- otestován,
- s uživatelskou dokumentací,
- připraven k předání uživateli.

Samotné nasazení softwaru do provozu a jeho údržbu už do odhadu složitosti nezahrnuji. Složitost odhaduji za účelem rozhodnutí, zda projekt, jehož cílem je finální softwarový produkt, realizovat, jak dlouho realizace bude trvat a kolik bude zhruba stát.

Míra úsilí je ovlivněna faktory vnějšími a vnitřními. Student připravující se na zkoušku z matematiky bude ovlivněn jednak dostupností materiálů, počtem cvičení, kvalitou cvičícího (faktory vnější), ale i vlastní leností, nebo neschopností udržet pozornost (faktory vnitřní). Proto je nutné tyto faktory při odhadu složitosti zohlednit.

Metoda odhadu složitosti softwarového produktu

Jak již bylo uvedeno, složitost chápejme jako míru úsilí nutnou vynaložit za účelem dosažení zvoleného cíle. Tato složitost je ovlivněna faktory vnitřními a vnějšími. Pokud se konečně (zpravidla jednou týdně) odhodlám jít běhat, pak je moje míra úsilí dána cílem, který jsem si

zvolil = uběhnout 5 km. Mimo to se na realizaci tohoto cíle podílejí faktory vnitřní – celková únava, lenost, obezita ale i faktory vnější – počasí, nebo nepředpokládaně rozkopaná silnice na trati. Proto nemůže být míra úsilí vždy stejná.

Proto navrhuji pro účel odhadu složitosti softwarového produktu definovat:

- **Základní složitost diagramu užití** - ZS.
- **Základní složitost softwarového produktu** - ZSP.
- **Faktor (faktory) Složitosti** ovlivňující složitost celého softwarového produktu - FS. Ty navrhuji považovat za Manažerské charakteristiky složitosti vývoje IS. Za účelem jejich klasifikace je navrhuji dělit na:
- **Faktory Složitosti klasifikovatelné** FSk. Ty navrhuji považovat za Manažerské charakteristiky zadání požadavků na IS.
- **Faktory Složitosti klasifikovatelné obtížně** FSo. Ty navrhuji považovat za Manažerské charakteristiky prostředí vývoje IS.
- **Celkovou složitost softwarového produktu** - CSP, která je „nějakou" funkcí základní složitosti softwarového produktu a faktorů ovlivňujících složitost softwarového produktu CSP = f(ZSP,FS). Tu navrhuji považovat za Základní manažerskou charateristiku vývoje IS.

Základní složitost diagramu užití

Základní složitostí diagramu užití budeme chápat počet případů užití v diagramu užití. Případ užití musí být minimálně jeden v jednom diagramu užití.

Základní složitost softwarového produktu

Během procesu tvorby případů užití a jejich grafického zaznamenávání do diagramu užití je téměř vždy potřeba využít většího počtu diagramů užití. Většina softwarových produktů totiž není tak jednoduchá, aby všechny případy užití mohly být zaznamenány v jednom diagramu užití. Proto je vhodné zavést pojem základní složitost softwarového produktu ZSP, který je roven součtu základních složitostí všech vzniklých diagramů užití pro zvolený softwarový produkt ZSP.

$$ZSP = \sum_{i=1}^{n} ZS_i$$

Faktory složitosti klasifikovatelné

Každý diagram užití obsahuje kromě případů užití také aktory a vazby mezi aktory a případy užití. Typ aktorů, počet aktorů a počet vazeb mezi případy užití pak jednoznačně nějak ovlivňuje složitost diagramu užití a tudíž i celého softwarového produktu. Proto je vhodné je při výpočtu složitosti nějak zohlednit. Nazývám je Faktory složitosti klasifikovatelné – FSk. Jejich klasifikací dostaneme Manažerské charakteristiky zadání požadavků na IS. Tyto charakteristiky pak slouží k rozohodování o realizaci softwarového produktu, o jeho ceně atd.

Faktory složitosti obtížně klasifikovatelné

Každý projekt (slovem projekt budeme chápat plánování, sběr uživatelských požadavků na požadovaný software, jejich analýzu, řízení vzniku softwarového produktu včetně jeho otestování, s vzniklou uživatelskou dokumentací) je ještě ovlivněn řadou dalších faktorů, které se klasifikují obtížně. Navrhuji je značit FSo. Na rozdíl od FSk, nejsou tyto faktory patrné z diagramu užití. Jde o faktory zohledňující například:

- Prostředí firmy, která projekt realizuje.
- Typ softwaru, který vzniká.
- Použitý programovací jazyk.
- Druh vývojářského týmu.
- Požadavek na stabilitu vyvíjeného softwaru.
- A řada dalších.

Pro tyto faktory je třeba také zavést nějakou třídu faktorů FSo. Potíž spočívá v tom, že se tato třída bude patrně lišit u každé firmy.

Dalším problémem je to, že tyto faktory by měly být voleny s rozvahou. Proto je pro účel metody důležité navrhnout třídu nějakých základních – obecných faktorů, které každá firma bude nějak splňovat. Tím fakticky navrhnout a vytvořit Manažerské charakteristiky prostředí vývoje IS, které jsou pro účel výpočtu zastupovány faktory FSo.

Každý faktor navrhuji klasifikovat podle ordinální stupnice:

- 0 - Faktor neexistuje nebo nemá vliv.
- 1 – Faktor existuje má střední vliv.
- 2 – Faktor existuje, jeho vliv je zásadní.

Celková složitost softwarového produktu

Jak jsem již uvedl, celková složitost softwarového produktu CSP je “nějakou” funkcí základní složitosti softwarového produktu ZSP a faktorů složitosti FS. Slovo “nějakou” uvádím zcela úmyslně v uvozovkách.

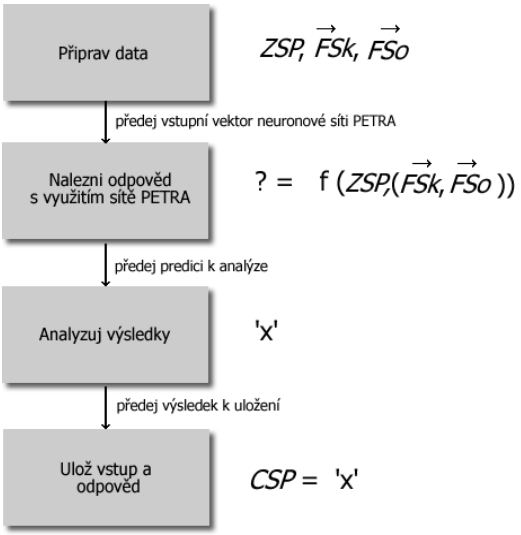
Není to proto, že bych snad chtěl zavádět nějaký vágní pojem do tohoto příspěvku, ale prostě proto, že tato funkce je neznámá. Ze svých zkušeností usuzuji, že není možné nalézt obecně platný vzorec, který by po dosazení vhodných konstant mohl celkovou složitost vyčíslit. Je to dáno jednoznačně tím, že veškeré atributy měření jsou ordinálního typu. Tudíž již od samého počátku je velmi problematické s nimi provádět nějaké operace. Proto navrhuji celkovou složitost softwarového produktu CSP chápat jako funkci základní složitosti softwarového produktu ZSP a faktorů složitosti. CSP pak navrhuji považovat za základní manažerskou charakteristikou vývoje IS, protože dává možnost se na jejím základě rozhodnout, zda realizovat softwarový produkt, jaké zdroje pro jeho vývoj alokovat, či uvažovat o jeho výsledné ceně.

Nalezení vhodné, ale specifické pro daný problém, funkce je úkol pro neuronovou síť PETRA, kterou jsem pro účel odhadu navrhl. Ta je pomocí dat, naměřených z již realizovaných softwarových produktů, adaptována k řešení tohoto úkolu. Její nespornou výhodou je její adaptace na zvolené prostředí. Jedinou podmínkou její úspěšné adaptace je vstupní vektor hodnot a vhodně zvolený učební soubor.

Metoda CSP – postup

V této kapitole popisuji moji metodu odhadu složitosti softwaru na základě výpočtu ZSP a stanovením příslušných faktorů složitosti FS.

Tato metoda odhadu má následující charakter, viz obr.1.



Obr. 1. Metoda odhadu CSP.

1. Prvním krokem odhadu složitosti je nejprve připravit vhodný vstupní vektor naměřených dat. Tím je:
   - ZSP
   - FSk
   - FSo

$$vstup = \langle ZSP, FSk, FSo \rangle$$

2. Druhým krokem je předložení vstupního vektoru neuronové síti. Ta odpoví jednou číselnou hodnotou. Záleží na způsobu naučení sítě, má-li tato hodnota představovat člověkoměsíce, člověkohodiny, počet dní atd.
3. Třetím krokem je analyzovat výsledek. Výsledky lze rozdělit do tří skupin:
4. Výsledek je pravděpodobný. Výsledek uložíme do nějaké báze dat v libovolném formátu.
   - Výsledek je nepravděpodobný. Uložíme jej do báze dat a porovnáme jej po čase se skutečností.
   - Výsledek je nesmyslný. Znamená to, že je síť špatně naučena, nebo jsme zadali špatně vstupní vektor, nebo se jedná o nějaký další problém, který je vhodné zkoumat. Nejčastěji k tomuto problému dochází, je-li síť přeučena.

Architektura a topologie neuronové sítě PETRA

Neuronová síť PETRA (PErcepTRon Artificial inteligence) je perceptronová neuronová síť s jednou skrytou vrstvou, s jedním výstupním neuronem a se sigmoidální přenosovou funkcí jednotlivých neuronů, kterou jsem navrhl a naprogramoval v jazyce Java pro účel odhadu CSP. Architekturu a topologii sítě jsem navrhl na základě studia odborné literatury viz [5] a

na základě testů, které jsem během vývoje sítě a návrhu metody odhadu složitosti softwarového produktu provedl. Odborné problémy, které jsem řešil při konstrukci sítě PETRA, jsem konzultoval s doc. Veselým.

Ukázalo se, že síť se sigmoidální přenosovou funkcí, s jednou skrytou vrstvou, ve které je vždy dvojnásobný počet neuronů než je neuronů ve vstupní vrstvě, které jsou spojeny každý s každým neuronem ve vstupní vrstvě a stejně tak je každý neuron skryté vrstvy spojen s jedním neuronem výstupní vrstvy, dává nejlepší výsledky. Tato architektura a topologie sítě je dostačující k tomu, aby se síť byla schopna naučit rozpoznávat vstupní vzory viz [5]. Proto jsem se ji rozhodl použít pro odhad složitosti CSP.

PETRA se učí pomocí mechanismu zpětného šíření (error backpropagation) chyby viz [5]. Tento mechanismus by viz [5] měl být dostatečně účinný pro její adaptaci na zvolené prostředí firmy.

Navrhl jsem toto základní nastavení sítě PETRA:

- 30 neuronů na vstupní vrstvě (počet neuronů je stejný jako počet prvků ve vstupním vektoru).
- 60 neuronů v skryté vrstvě.
- 1 neuron ve výstupní vrstvě.
- Počáteční váhy neuronů jsou vždy voleny náhodně.

Protože však pro účely metody odhadu složitosti softwarového produktu může být potřeba zvětšit vstupní vektor (rozšířit třídu FSk nebo FSo), je PETRA konfigurovatelná. Uživatel má možnost si dle své potřeby určovat počet vstupních neuronů. Zachovává se však pravidlo, že počet neuronů v skryté vrstvě je vždy dvojnásobný.

Rozpoznání klíčových faktorů složitosti – postup

Rozpoznávání klíčových faktorů složitosti je funkce, která umožní uživateli rozpoznat, který faktor se na realizaci softwarového produktu projevuje zásadním způsobem.

Postup je následující:

- Nejprve předložíme síti vstupní vektor obsahující (ZSP,FS) a provedeme odhad CSP.
- Proběhne-li odhad a je -li odhad rozumný (nevyjde nesmyslná hodnota), přistoupíme k druhému kroku. Zapneme automatickou funkci rozpoznávání klíčových faktorů složitosti, která je zabudována v nástroji PETRA. Dá se ale udělat i mechanicky.

Navrhuji následující postup:

Každý prvek vstupního vektoru má svůj vlastní neuron ve vstupní vrstvě. Má - li vstupní vektor 30 položek, pak má neuronová síť PETRA na vstupu 30 vstupních neuronů. Každý vstupní neuron je spojen se všemi neurony ve skryté vrstvě. Každý spoj mezi neuronem vstupní a skryté vrstvy má určitou váhu. Pro každý prvek vstupního vektoru pak provedeme analýzu tak, že postupně měníme váhy na opačné (zachováme absolutní hodnotu váhy, ale změníme její znaménko) pro jeden prvek vstupního vektoru a sledujeme, zda se po změně vah nějak zásadně nezměnil výstup:

- Zůstává - li hodnota výstupu podobná s původně provedeným odhadem, faktor se nijak zásadně neprojevuje na složitosti softwarového produktu.
- Změní - li se hodnota výstupu, například změnou znaménka, nebo výrazně vroste či klesne (je - li původní odhad 300 člověkodní a hodnota se změní o 50 % nahoru či dolů), podílí se faktor na celkové složitosti softwarového produktu zásadně.

Tato funkce neříká, jakým způsobem se faktor na celkové složitosti softwarového produktu projevuje. Pouze na něj upozorňuje. Je na uživateli, jak s touto informací.

Odhad celkové složitosti projektu s využitím neuronové sítě PETRA v prostředí reálné firmy

Druhým krokem pro uznání metody odhadu CSP za použitelnou je zjistit, jak přesně je PETRA schopna, na základě naučení, odhadovat složitosti projektů. Z tohoto důvodu jsem vytvořil učební soubor z dat získaných ve firmě Sun Microsystems. Z učebního souboru jsem vybral čtyři projekty, pro které jsem se rozhodl odhadovat CSP. Každý projekt byl realizován jiným autorem. Podmínkou pro odhad složitosti bylo, aby v učebním souboru sítě byl vždy alespoň jeden projekt realizován autorem, pro kterého je složitost odhadována.

To znamená:

Autor 1, pro kterého byla predikována složitost v člověkodnech musel mít 1 nebo více realizovaných softwarových produktů v učebním souboru.

Projekty, pro které byla odhadována složitost byly vždy rozdílné od projektů v učebním souboru. Nebyly identické.

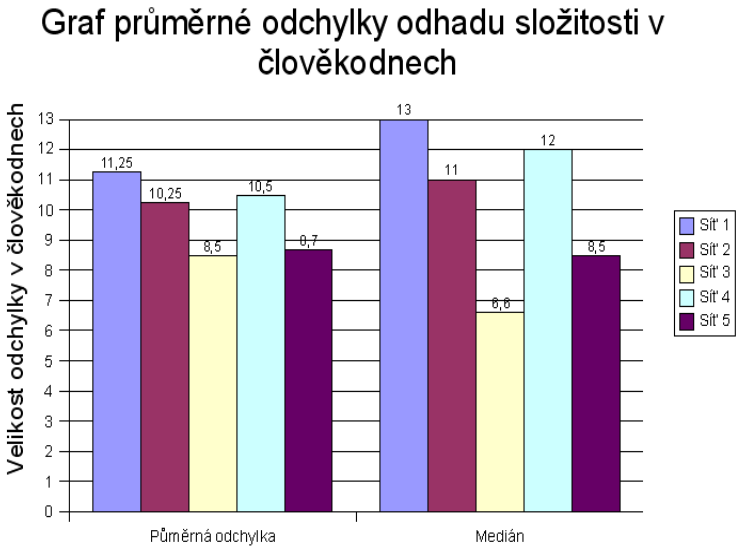
Bylo učeno pět sítí. Každá síť pak odhadovala složitost v člověkodnech pro čtyři různé softwarové produkty. V Tabulce 1 vidíme průměrnou odchylku odhadu složitosti jednotlivých sítí v člověkodnech od skutečné složitosti v člověkodnech a medián odchylky v člověkodnech.

Soft. Produkt	Odchylka 1	Odchylka 2	Odchylka 3	Odchylka 4	Odchylka 5
1	13	13	20	14	1
2	13	9	3	10	2
3	16	16	10	15	3
4	3	3	1	3	4
Průměrná ochylka	11,25	10,25	8,5	10,5	2,5
Medián	13	11	6,5	12	2,5

Tabulka. 1. Odchylka sítí při predikci CSP pro nezámé softwarové produkty.

Ve sloupcích Odchylka 1 až Odchylka 5 vidíme odchylky v odhadu složitosti sítí v člověkodnech pro 4 neznámé softwarové produkty. V řádku průměrná odchylka vidíme průměrnou odchylku sítí v člověkodnech pro všechny čtyři softwarové produkty. V poslední řádce je zobrazen medián odchylek jednotlivých sítí v člověkodnech.

Graf 1 zobrazuje průměrnou odchylku sítí při odhadu složitosti v člověkodnech (vlevo) a medián odchylek v člověkodnech (vpravo).



Graf. 1. Graf jednotlivých odchylek sítí v člověkodnechwarové produkty.

Na základě naměřených hodnot jsem vypočítal maximální a průměrnou procentuelní odchylku sítí. Tabulka 2 zobrazuje Maximální a průměrnou odchylku sítí 1 až 5 v procentech.

	Síť 1	Síť 2	Síť 3	Síť 4	Síť 5
Maximální % odchylka	21,67	21,67	33,33	23,33	23,33
Průměrná % odchylka	15,08	14,38	12,74	14,67	12,91

Tabulka. 2. Maximální a průměrná odchylka sítí v procentech.

Závěr kapitoly

Neuronová síť PETRA byla schopna predikovat CSP s 15 % průměrnou odchylkou. Při tom největší odchylka činila 33 %. Experimenty dokázaly, že neuronová síť pro odhad složitosti software s využitím předepsaných faktorů složitosti je použitelným nástrojem.

Závěr

Výsledky získané experimentem ukázaly, že průměrná odchylka sítí je srovnatelná s odchylkou metod COCOMO nebo FP, u kterých literatura uvádí odchylku přesnosti odhadu v rozmezí 20 až 30 %.

Výsledky odhadu CSP jsou velmi přesné - 15 %. V budoucnu bude vhodné odzkoušet schopnost odhadu sítí na větším vzorku softwarových projektů. Předložený vzorek, byť se jednalo o reálné projekty, byl velmi specifický. Firma Sun Microsystems má velmi přesně řízený vývoj softwarových komponent. To většinou neplatí u běžných softwarových firem. Mimo to, vzorek zahrnoval vývoj softwarových komponent pro prostředí NetBeans. Ty jsou samy o sobě specifické a jistě jejich složitost vývoje je mnohem menší, než tvorba velkých informačních systémů. Data o takových projektech však nejsou volně k dispozici. Proto v

současné době využívám dat naměřených ve firmě IBM. Získaná data nemohu volně uvádět v disertační práci.

Studiem možností odhadu složitosti softwarových produktů s využitím umělé inteligence se budu zabývat i v budoucnosti. O výsledcích své práce budu informovat veřejnost formou vědeckých konferencí a odborných publikacích.

LITERATURA

[1] *ISO/IEC JTC1*: http://www.jtc1.org/,10.9.2006.
[2] Jacobson I.,Booch G., Rumbaugh J., *Unified Software Development Process,* Published: Feb 4, 1999, ISBN 020-1571-692.
[3] Merunka V, Polák J., Carda A., *Umění systémového návrhu*, Grada 2003, ISBN 80-247-0424-X.
[4] Jacobson I.,Booch G., Rumbaugh J., *Unified Software Development Process*, Published: Feb 4, 1999; Copyright 1999,ISBN 020-1571-692.
[5] Veselý A., – *Artificial Inteligence* – Hand out, 2006.
[6] Pavlíček J., – *Objekty 2005, pp.80-248-0595-2.*
[7] Lacko B., – *Aplikace metody RIPRAN v softwarovém inženýrství*, Sborník celostátní konference Tvorba Software Ostrava.