ŠKODA AUTO VYSOKÁ ŠKOLA o.p.s.

Studijní program: Podniková ekonomika a manažerská informatika

Vývoj a oceňování specializovaného softwaru

Diplomová práce

Jana JIRÁNKOVÁ

Vedoucí práce: Ing. Vladimír Beneš, Ph.D.

ŠKODA AUTO VYSOKÁ ŠKOLA o.p.s.
Katedra informatiky

Akademický rok: 2022/2023

ŠKODA AUTO Vysoká škola

ZADÁNÍ DIPLOMOVÉ PRÁCE

Zpracovatelka: **Ing. Jana Jiránková**

Studijní program: Podniková ekonomika a manažerská informatika

Název tématu: **Vývoj a oceňování specializovaného softwaru**

Cíl: Cílem diplomové práce je vysvětlit problematiku projektového řízení vývoje softwaru. Podrobně popsat a porovnat řízení vývoje softwarového projektu se zaměřením na metody vývoje, analýzu, modelování, životní cyklus i výslednou kvalitu softwaru. Teoretická část práce se zaměří také na oceňovací (ohodnocovací) přístupy a metody sloužící pro stanovení hodnoty softwarových projektů. Praktická část se bude zabývat vývojem softwarového prototypu sloužícího pro detekci nespolehlivých plátců DPH. Rovněž bude v praktické části na základě dostupných oceňovacích modelů vyvinut prototyp softwarového nástroje, který může být nápomocný při stanovení hodnoty softwarového projektu.

Rámcový obsah:

1. Vývoj softwaru
2. Oceňování softwarových projektů
3. Vývoj a ocenění specializovaného softwaru

Rozsah práce: 55 – 65 stran

Seznam odborné literatury:

1. KADLEC, V. *Agilní programování: metodiky efektivního vývoje softwaru.* Brno: Computer Press, 2004. 278 s. ISBN 80-251-0342-0.
2. SOMMERVILLE, I. *Softwarové inženýrství .* Brno: Computer Press, 2013. 680 s. ISBN 978-80-251-3826-7.
3. ROUDENSKÝ, P. *Kvalita softwaru: teorie a praxe .* Brno: Computer Media, 2017. 232 s. ISBN 978-80-7402-294-4.
4. SVAČINA, P. *Oceňování nehmotných aktiv.* 1. vyd. Praha: EKOPRESS, 2010. ISBN 978-80-86929-62-0.
5. KADLEC, V. *Učíme se programovat v jazyce C.* Praha: Computer Press, 2002. ISBN 80-7226-715-9.
6. SCHWALBE, K. *Řízení projektu v IT .* Brno: Computer Press, 2011. 632 s. ISBN 978-80-251-2882-4.
7. BRUCKNER, T. – ALENA, B. *Tvorba informačních systémů.* Praha: Grada Publishing, a.s., 2012. 360 s. ISBN 978-80-247-4153-6.
8. ČADA, K. *Chránit či nechránit / to je otázka. Výsledky výzkumu a vývoje, jejich ochrana a komercializace.* Praha: Alevia, 2014. 320 s. ISBN 978-80-905-5380-4.
9. ŘEPA, V. *Procesně řízená organizace .* Praha: Grada Publishing, a.s., 2012. 304 s. ISBN 978-80-247-4128-4.
10. BUCHALCEVOVÁ, A. *Zlepšování procesů při budování informačních systémů.* Praha: Oeconomica, 2018. 227 s. ISBN 978-80-245-2235-7.
11. BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů.* Praha: Grada Publishing, a.s., 2005. 163 s. ISBN 80-247-1075-7.

Datum zadání diplomové práce: duben 2022

Termín odevzdání diplomové práce: květen 2023

L. S.

Elektronicky schváleno dne 25. 4. 2022

Ing. Jana Jiránková
Autorka práce

Elektronicky schváleno dne 25. 4. 2022

Ing. Vladimír Beneš, Ph.D.
Vedoucí práce

Elektronicky schváleno dne 26. 4. 2022

prof. Ing. Jiří Strouhal, Ph.D.
Garant studijního programu

Elektronicky schváleno dne 26. 4. 2022

doc. Ing. Pavel Mertlík, CSc.
Rektor ŠAVŠ

Prohlašuji, že jsem závěrečnou práci vypracoval(a) samostatně a použité zdroje uvádím v seznamu literatury. Prohlašuji, že jsem se při vypracování řídil(a) vnitřním předpisem ŠKODA AUTO VYSOKÉ ŠKOLY o.p.s. (dále jen ŠAVŠ) směrnicí Vypracování závěrečné práce.

Jsem si vědom(a), že se na tuto závěrečnou práci vztahuje zákon č. 121/2000 Sb., autorský zákon, že se jedná ve smyslu § 60 o školní dílo a že podle § 35 odst. 3 je ŠAVŠ oprávněna mou práci využít k výuce nebo k vlastní vnitřní potřebě. Souhlasím, aby moje práce byla zveřejněna podle § 47b zákona č. 111/1998 Sb., o vysokých školách.

Beru na vědomí, že ŠAVŠ má právo na uzavření licenční smlouvy k této práci za obvyklých podmínek. Užiji-li tuto práci, nebo poskytnu-li licenci k jejímu využití, mám povinnost o této skutečnosti informovat ŠAVŠ. V takovém případě má ŠAVŠ právo ode mne požadovat příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to až do jejich skutečné výše.

V Mladé Boleslavi dne 11. května 2023

Děkuji Ing. Vladimíru Benešovi, Ph.D. za odborné vedení závěrečné práce, poskytování rad, připomínek a námětů i informačních podkladů.
Poděkování patří také prof. Ing. Václavu Řepovi, CSc. za cenné rady a podněty, které mi poskytl v rámci odborných předmětů.
Vážím si času, který oba věnovali této diplomové práci.

Obsah

Seznam použitých zkratek a symbolů

AI Umělá inteligence (Artificial Intelligence)

AGI Obecná umělá inteligence (Artificial General Intelligence)

DAST Dynamické testování zabezpečení aplikací (Dynamic Application Security Testing)

DFD Diagram datových toků

EA Enterprise Architecture (Podniková architektura)

ICT Informační a komunikační technologie

IDE Integrované vývojové prostředí (Integrated Development Environment)

IS Informační systém

IT Informační technologie

RUP Rational Unified Process

SAST Statické testování zabezpečení aplikací (Static Application Security Testing)

SW Software

TH Tržní hodnota

UML Unified Modeling Language

Úvod

Software a jiná nehmotná aktiva spolu s kvalifikovaným personálem a jeho know-how lze ve 21. století považovat za silné zbraně nejen v konkurenčních soutěžích a touze o upevnění pozice na trhu, ale i ve snaze o zavedení inovativních řešení, zefektivnění stávajících podnikových procesů či ve snaze o podnikovou digitální transformaci. Díky spojení dostupných technologií a lidské tvůrčí mysli lze docílit pozoruhodných výsledků ve výzkumu a vývoji jedinečných softwarových řešení, které mohou být podnikovou vizitkou splňující vize a cíle klíčových podnikových procesů. V digitální době lze software považovat za druh umění, jehož hodnota je odrazem celé řady aspektů, novátorských myšlenek, osobností vývojářů i doby a okolností, které během tvorby ovlivňovaly softwarový proces.

Cílem diplomové práce je vysvětlit význam vývoje softwaru a oceňovacích metod. Práce poukazuje na důležitost budování IT podniku jako dlouhodobé strategie, na vývoj běžných i specializovaných softwarů jako na jeden díl v řetězci podnikové ICT sítě a na oceňování softwarových projektů jako na součást, která uzavírá pomyslný kruh softwarového procesu. Dílčím cílem teoretické části diplomové práce je upozornit na kvalitu softwaru nejen z hlediska čistoty zdrojového kódu, ale i z hlediska ceny za softwarovou chybu či z hlediska odolnosti vůči temnému světu kybernetických útoků. Cílem praktické části je demonstrovat nejen softwarový proces vývoje softwaru, ale i snahu o zachycení vypovídající hodnoty softwarového projektu a jeho jedinečných specifik na základě dostupných podkladů.

První kapitola se podrobně zabývá vývojem softwaru. Popisuje, jakou roli hraje softwarový proces ve vývoji a které jeho aktivity mají klíčový vliv na úspěch (či neúspěch) výsledného SW řešení. Poukazuje na důležitost vhodné volby metodiky definující systematické postupy, díky jejichž dodržování se lze vyhnout základním procesním a softwarovým chybám. Díky poučení se z chyb se lze celé řadě problémů během vývoje SW vyhnout a úspěšně tak předat výsledný SW projekt, který splní očekávané cíle. Kapitola také vymezuje a definuje vybrané modely vývoje SW projektu, které je možné díky jejich univerzálnosti a propracovanosti brát v úvahu při vývoji běžného i specializovaného softwaru. Vývoj softwaru by měl být časově omezen, a proto nelze opomenout životní cyklus. První kapitola popisuje, kdy životní cyklus začíná a v jaké fázi vývoje končí.

Odpovídá na otázku, jaké aktivity vývoji SW předcházejí a jaké nezbytné kroky po implementaci SW do podnikového prostředí následují. Poslední subkapitola první části diplomové práce se zabývá uměním kvality softwaru. Obohacuje o zajímavé pohledy několika vývojářů a IT myslitelů na kvalitu zdrojových kódů a výsledných SW projektů, které mohou být ve světě informačních technologií považovány za dokonalý klenot s nadnárodním i nadčasovým přesahem.

Druhá kapitola je vyhrazena pro oceňování softwarových projektů. Detailně vymezuje a definuje klíčové pojmy a jednotlivé předpisy legislativního rámce oceňování nehmotných aktiv, které lze považovat za základní stavební kámen stanovení hodnoty SW projektů. Odpovídá mimo jiné na otázku, jaký je rozdíl mezi oceňováním a ohodnocováním aktiv v podmínkách behaviorálního prostředí. Doplňující problematikou je vymezení jednotlivých specifik, která jsou typická pro nehmotná aktiva a která svým způsobem mohou ovlivnit výslednou hodnotu SW projektu. Díky těmto specifickým charakteristikám lze jednotlivým softwarům přiřadit punc jedinečnosti, nadčasovosti či výjimečnosti. Významnou částí druhé kapitoly jsou metody oceňování, které představují jednotlivé přístupy a matematické postupy oceňování SW projektů. Jelikož problematika oceňování závisí na subjektivním posouzení oceňovatele, věnuje se poslední část druhé kapitoly optice pohledu na hodnotu posuzovaného předmětu i funkcím ocenění, které ocenění mohou mít. Kapitola v neposlední řadě odpovídá na otázku: *„Jakou roli může oceňovatel v softwarovém procesu sehrát?“*

Třetí kapitola pomocí praktické případové studie demonstruje vývoj specializovaného softwaru s jeho následným oceněním. Cílem této kapitoly je vyvinout dva prototypy softwarového nástroje. První softwarový nástroj (na základě dat z interní databáze podniku) slouží k automatizované detekci nespolehlivých plátců DPH dle platných legislativních předpisů. Druhý prototyp softwarového nástroje napomáhá k určení hodnoty softwarového projektu na základě metod ocenění se zohledněním specifik vycházejících z posouzení jednotlivých kroků softwarového procesu. Výstupem této kapitoly je, mimo jiné, ilustrace *Ohodnocení softwarového procesu*, která znázorňuje jeho dílčí kroky a zdokumentované výstupy, a soubor úvah, které mohou se zohledněním specifik softwarového projektu pomoci.

Metoda vědeckého zkoumání v teoretické části diplomové práce je založena na analýze a rešerši tuzemské i zahraniční literatury. Naproti tomu praktická část představuje aplikaci teoretických poznatků na vývoji dvou softwarových prototypů, které obsahují popis algoritmu (vývojový diagram) a zdrojový kód. Součástí praktické části je ilustrace *Ohodnocení softwarového procesu*, která může být nápomocná při zohlednění jednotlivých fází vývoje v hodnotě SW projektu.

1 Vývoj softwaru

Svět ekonomiky, byznysu a technologií se od 20. století vzájemně propojuje, a proto je vývoj softwaru[1] považován za jeden z nejdůležitějších a nejdynamičtěji rostoucích odvětví. Organizace mají snahu se s novými výzvami vypořádat zapojením inovativních nástrojů i metodologií, které trh nabízí či které lze vytvořit vlastní činností nebo zadáním specializovanému týmu v takové míře, ve které ji technologie či finanční prostředky umožňují. Řada z nich věnuje nemalé prostředky na vlastní výzkum a vývoj softwarových nástrojů, které by jim pomohly odlišit se od konkurence zrychlením podnikových procesů, zefektivněním komunikace v dodavatelských řetězcích, zkvalitněním služeb pro zákazníky s možností customizace produktů a služeb či jiným zkvalitněním podnikových procesů. V organizacích vznikají samostatná oddělení řešící podnikovou informatiku[2], jejímž cílem je implementovat pořízené softwarové nástroje do podnikových procesů se snahou vybudovat kvalitní podnikovou ICT síť respektující potenciál firmy.

Ovšem s vývojem podnikové ICT sítě souvisí i mnohá úskalí, která na první pohled nemusí být zcela zřejmá. Za chybný i vysoce nákladný přístup lze považovat bezcílnou digitalizaci, díky níž podnik utratí prostředky pouze kvůli snaze jít s dobou a přizpůsobit se tlaku trhu. Taková digitalizace může vytvářet falešnou iluzi změny díky vykázaným nákladům představujících vynaložené finanční prostředky, avšak ve skutečnosti se jedná pouze o číselnou přeměnu[3], která může ve výsledku podnikové procesy zkomplikovat a učinit je chaotičtějšími. Tento jev by se dal

---

[1] *„Software je obecný termín pro sledy instrukcí, které počítač nutí, aby něco užitečného udělal. „Soft“ je v angličtině opakem „hard“ ze slova hardware, které označuje všechny fyzické součásti počítače – to znamená, že software je nehmotný. Spadne-li nám notebook na nohu, všimneme si toho. To o softwaru neplatí.“* (KERNIGHAN, W. B., 2019) Software se mnohdy také přirovnává ke kuchařským receptům, které obsahují seznam surovin, postup kroků a očekávaný výsledek. Software také potřebuje data, se kterými bude pracovat, a soubor pokynů, díky nimž zpracuje požadovaný úkol. Další analogií z běžného života mohou být formuláře daňového přiznání, ve kterých jsou prováděny aritmetické operace, zaokrouhlování, přenášení dat z jednoho řádku na druhý, dochází také k ověřování podmínek na základě platné legislativy. Výsledek je úplný a jednoznačný v předem definovaném formátu (zejména Kč, případně měrné jednotky), který vznikl díky souboru na sobě závislých systematických kroků.

[2] Podnikovou informatiku lze charakterizovat jako soubor zdrojů, procesů a služeb. Za zdroje lze považovat technologickou infrastrukturu (jako je například hardware či počítačová síť), aplikační software, data či personál.

[3] Číselnou přeměnou se rozumí účetní operace, díky které dochází k přesunu peněžních aktiv do nehmotných aktiv. Díky tomu může podnik nabýt falešný pocit digitalizace díky pravidelně vynaloženým financím na rozvoj podnikového ICT a softwarových nástrojů.

přirovnat k syndromu vařené žáby[4], který spočívá v neschopnosti či neochotě uvědomovat si hrozby, které mohou přicházet pozvolně a v čase mohou narůstat do obřích rozměrů. Je důležité si uvědomit, že změna musí přicházet dlouhodobě, promyšleně, systematicky, důsledně a na úrovni celé organizace. Tuto změnu lze nazvat digitální transformací[5]. Cílem digitální transformace by měl být pokus o ideální splynutí informačně-technické[6] podpory s organizačním systémem[7]. Díky tomuto splynutí budou schopny podnikové procesy vytvářet hodnotu.

Aby bylo možné bezprostředně využívat všech inovativních možností, jež přináší technologický rozvoj a digitální transformace, je nutné, aby organizace upustily od tradičního hierarchického řízení a přešly k principům procesně řízené organizace[8]. Díky technologiím je možné i v procesně řízené organizaci udržet maximální možnou informovanost a pravděpodobně i na výrazně lepší úrovni než v případě hierarchické struktury, neboť data mohou být přístupná v reálném čase.

---

[4] Metaforu podle bajky lze vysvětlit následovně: Pokud je žába hozena do vroucí horké vody, uvědomí si riziko a vyskočí ven. Pokud je ale žába vhozena do studené vody a pozvolna ohřívána, díky falešným iluzím (není ještě tak zle) nevyskočí, i přes to, že hrozba přichází pomalu a postupně narůstá. Žába si riziko neuvědomí, neboť si na teplotu postupem času zvykla.

[5] Digitální transformací se myslí využití technologií k radikálnímu zlepšení výkonu nebo rozšíření rozsahu. Digitální transformace závisí na lidském potenciálu, který umí vhodným způsobem kombinovat technologie a digitální aktivity se silným leadershipem. Díky lidem a jejich schopnostem mají společnosti možnost vybudovat silnou vizi, která jim může zajistit digitální vyspělost v éře informační společnosti.

[6] Za informačně-technické prostředky lze považovat informační systém podniku se všemi náležitostmi, které s jeho provozem souvisí. Jako příklad těchto náležitostí lze považovat veškerý hardware (počítače, servery aj.) i softwarové nástroje, kterými podnik disponuje a které ke své činnosti využívá.

[7] Organizačním systémem jsou myšleny lidé a jejich kvalifikace, úroveň využití jejich potenciálu i schopnosti správného přiřazení jejich rolí v rámci týmu či projektu.

[8] *„Procesním řízením se rozumí řízení firmy takovým způsobem, v němž business (podnikové) procesy hrají klíčovou roli.“* (ŘEPA, V. 2012) Jedná se o možnou cestu, jak zefektivnit výkon společností využitím všech dostupných znalostí, zkušeností, dovedností, nástrojů, technik, systémů, standardů a metodik k definování, vizualizaci, měření, kontrole, informování a neustálém zlepšování podnikových procesů s cílem splnit požadavky a představy zákazníků. *„Přemýšlet procesně znamená především důkladně změnit tradiční náhled na téměř cokoliv v životě organizace. Že to znamená například opustit představu hierarchické struktury, jako základu organizace firmy, mýtus „manažerské odpovědnosti“ za práci podřízených a z toho logicky plynoucí neodpovědnosti „podřízených“. Znamená to ale i pochopit podstatu smyslu vývoje technologií a především podstatu jeho role ve vývoji organizace.“* (ŘEPA, V. 2012) Pro takto smýšlející podnik lze použít pojem „e-organizace“, kterým poprvé Gary L. Neilson, Bruce A. Pasternack a Albert J. Viscio nazvali svou procesně řízenou firmu Booz-Allen Hamilton (americká poradenská společnost podnikající v oblasti managementu a informačních technologií).

1.1 Softwarový proces

Vývoj softwarového projektu představuje množinu uspořádaných, promyšlených, systematických a vzájemně provázaných kroků, které dohromady tvoří ucelený softwarový proces. Krokem je myšlena aktivita nebo podproces, který je hierarchickou dekompozicí celého procesu. Aktivit a podprocesů může být celá řada, proto je nezbytně nutná jejich koordinace i ve vztahu k jejich průběhu, který může být chronologický nebo může probíhat souběžně v čase. V ideálním případě je nutná snaha o opakovatelné použití softwarového procesu k jednotlivým softwarovým projektům, čímž lze zajistit vysokou úroveň kvality i bezchybné začlenění do podnikové ICT infrastruktury. Předpokladem stabilně fungujícího softwarového procesu jsou lidé[9], kteří jsou vybaveni znalostmi a schopnostmi i hardwarovými a softwarovými prostředky nezbytnými k realizaci SW procesu.

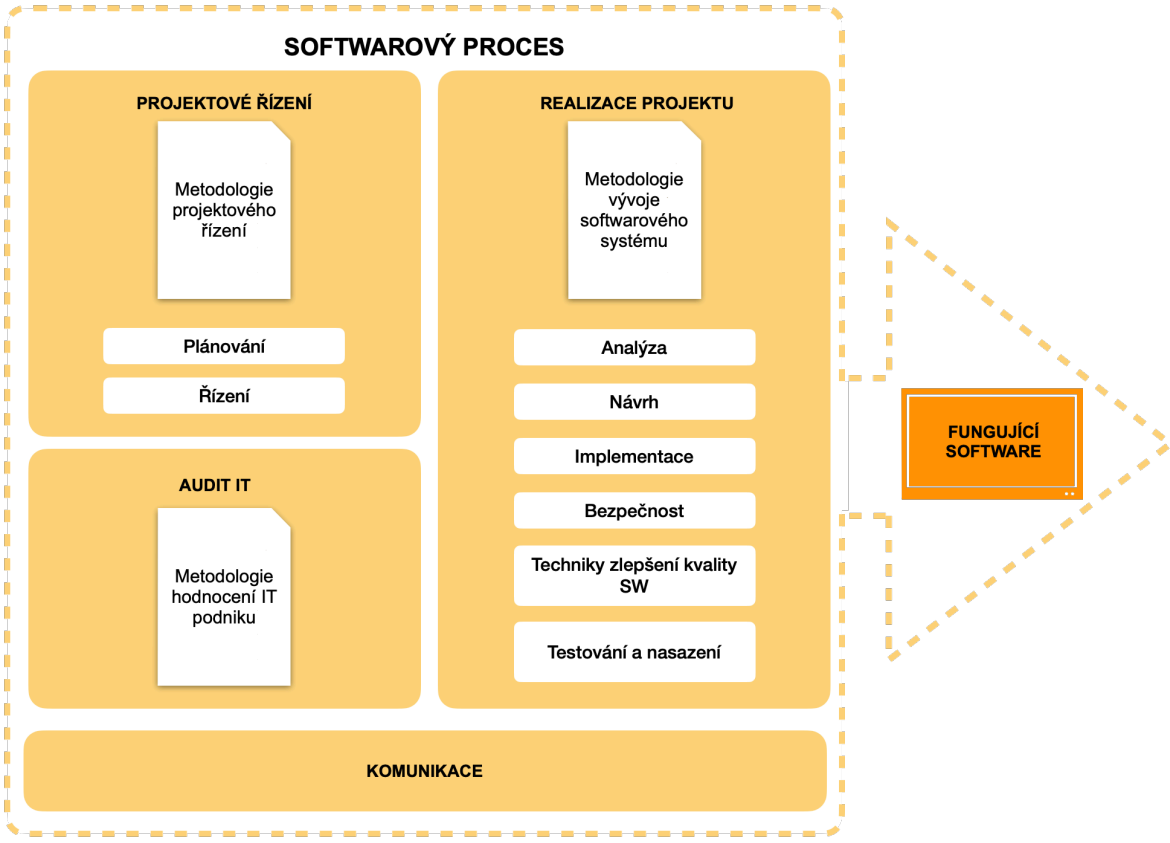
Softwarový proces lze v moderní společnosti považovat za součást podnikové informatiky, díky níž jsou procesy budování IS[10] a jiných SW nástrojů součástí podnikové informační kultury a EA[11]. Úrovní zralosti a vyspělosti podniku se zabývá příloha 1.

[9] Lidé jsou považováni za klíčový prvek i v rámci snahy o jakoukoliv změnu v podniku, nejen tu procesní, a nejen tu radikální a revoluční. Lidé tvoří nezbytný díl skládačky, který v samotném jádru musí být úplný a pevný jako jeden celek. Z tohoto důvodu lze vyvodit závěr, že lidé jsou všeobecně považováni za nejdůležitější prvek jakéhokoliv systému.

[10] Procesy (podpůrné i řídící procesy) spojené s vývojem a provozem IS.

[11] EA (*Enterprise Architecture, Podniková architektura*) je disciplína, která se zabývá popisem podnikových cílů a procesů, které lze podpořit technologiemi. Cílem EA je udržovat komplexní a mnohdy složité SW a IS ve stavu, který je podnik schopen řídit a plánovitě rozvíjet.

Obr. 1 Softwarový proces



Zdroj: autorka

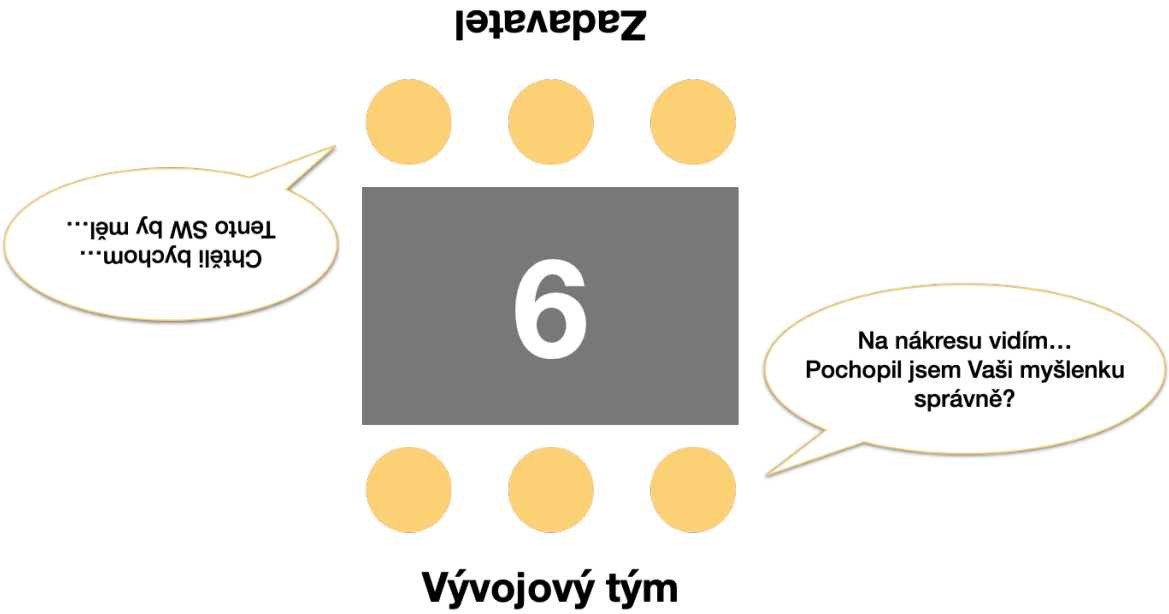
Prvním souborem aktivit je projektové řízení, které lze charakterizovat jako soubor znalostí, dovedností, nástrojů a technik, které je nutné aplikovat do projektových aktivit, aby bylo dosaženo zadání projektu. Díky řádnému projektovému řízení lze prostřednictvím plánování a řízení minimalizovat nesprávnosti ve výsledném projektu, časové prodlevy i navýšení původního finančního rozpočtu. Projektové řízení vychází z metodologie projektového řízení[12]. Druhou skupinou aktivit je realizace projektu, jejímž základem je metodologie vývoje softwarového systému. Třetí skupinou jsou aktivity spojené s auditem informačního systému i celé IT sítě podniku. Ačkoliv se může zdát, že vývoj softwaru je nejdůležitější činnost softwarového procesu, realita je ve skutečnosti jiná. Stavebním kamenem celého softwarového procesu je komunikace, která není upravena metodologiemi ani

[12] Za metodologii lze považovat mezinárodní standardy projektového řízení, například PMI (PMBOK Guide), PRINCE2 (Projekty v kontrolovaném prostředí), IMPA, ČSN ISO 21500 (Management projektů, programů a portfolií – Kontext a koncepce), ČSN ISO 10006 (Management kvality – Směrnice pro management kvality v projektech).

závaznými postupy. Jde o prostou měkkou dovednost člověka, která je získávána zejména praxí a díky které se člověk odlišuje od umělé inteligence.

Následující obrázek 2 poukazuje na skutečnost, proč je důležité komunikovat, vizualizovat a výstupy písemně dokumentovat.

Obr. 2 Proč je komunikace klíčový faktor úspěchu?[13]



Zdroj: autorka

Jelikož do softwarového procesu vstupuje celá řada účastníků, je klíčové veškerý sběr informací řádně dokumentovat a jejich případné úpravy důsledně schvalovat a formálně jej zaznamenat v dokumentaci. Nejvíce problémů s funkčností výsledného softwaru plyne z neformálních informací získaných domněnkami, chybnými předpoklady, zamlčenými představami či nedostatečným definováním požadavků. Je nezbytně nutné, aby na straně zadavatele projektu bylo zapojeno dostatečné množství lidí, kteří by svými vědomostmi a dovednostmi mohli přispět ke kvalitě a předpokládané představě výsledného softwarového řešení. Lidé z klíčových oddělení jako jsou například lidé z výroby či vývoje produktů a služeb disponují expertní znalostí firemního produktu. Mohou například vznést speciální požadavky, aby výsledný výrobní trakt byl plynulý, efektivní nebo aby výsledný SW byl nadčasový a plně v souladu se zamýšleným rozvojem firmy. Lidé z podpůrných oddělení jako jsou například lidé z obchodního nebo marketingového útvaru mohou

[13] Protože každá strana (každý uživatel softwarového procesu) může k jednacímu stolu přicházet s různými představami a s různými pohledy na stejnou problematiku.

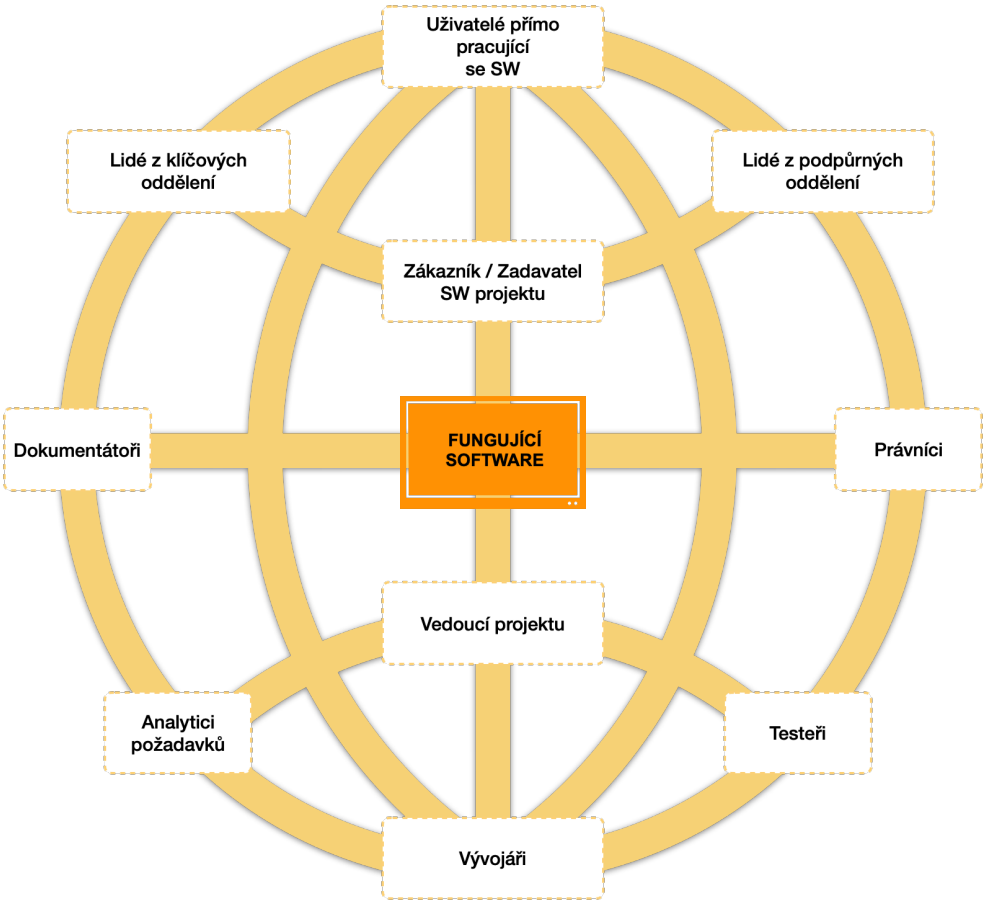
svými připomínkami přispět k vhodnému prezentování produktů uvnitř podniku i navenek. Mohou zabránit případným dezinformacím vyplývajících z mylného či domnělého výkladu informací o firmě a jejich produktech. Uživatelé přímo pracující se SW znají požadavky na jeho fungování na expertní úrovni. Mají přesnou představu, jak by takové SW řešení mělo fungovat a jaké výstupy by mělo poskytovat. Například účetní oddělení, pro které je vytvářen mzdový systém ví, jaké údaje o zaměstnancích jsou nezbytně nutné k výkonu činnosti, jakými funkcionalitami by mělo SW řešení disponovat a jaké právní pozadí se za problematikou skrývá. S výkladem právních předpisů mohou pomoci právníci, kteří softwarovému procesu poskytnou svou znalostní bázi a pomohou se správným výkladem jednotlivých ustanovení. Dokumentátoři by měli formálně zaznamenat veškeré informační toky mezi zúčastněnými uživateli tak, aby dokumentace byla živá[14] a v rámci softwarového procesu maximálně přínosná. Vedoucí projektu po celý softwarový proces vede vývojový tým k úspěšnému dokončení SW projektu a tím pádem k úspěšnému uspokojení zákazníkových potřeb. Za nedoceněné, a mnohdy i opomíjené, poslání vedoucího projektu lze považovat jeho ujištění se, že všechny zúčastněné subjekty znají cenu změn[15] požadavků, změn časové i finanční dotace a jiných nároků na SW projekt. Vývojový tým se skládá z analytiků požadavků, kteří v dokumentaci zaznamenávají požadavky na funkcionality a uživatelské prostředí a předávají je vývojářům. Dalo by se říct, že analytici jsou jakýmsi komunikačním mostem[16] mezi vývojáři a zadavateli SW projektu. Testeři zjišťují, jestli se SW řešení chová tak, jak zadavatel požaduje. Zde se nabízí spolupráce mezi testerem a analytikem, díky které mohou posoudit, jestli SW řešení odpovídá tomu, co zadavatel skutečně potřebuje.

[14] Živá dokumentace je charakteristická ideálním zaznamenáváním všech informačních toků v reálném čase. Principiálně by měla platit přímá úměra: čím je softwarový proces pokročilejší, tím je dokumentace obsáhlejší.

[15] Vedoucí projektu by měl udržovat chladnou hlavu, neboť se může stát, že zadavatele napadne nějaká nová myšlenka a spolu s vývojáři zapracují nový nápad do SW projektu, aniž by se s touto změnou počítalo v rámci SW harmonogramu. Zadavatel musí znát již v počátku projektu, jak drahé mohou být změny. Změny požadavků v pokročilé fázi vývoje SW mohou vyústit ve zdrojový kód, který nemusí být v souladu s nastolenými cíli a který je ve výsledku zcela nekonzistentní. V takovém případě je cena za změnu neúměrně vysoká, neboť vývojový tým může strávit mnoho času úpravami zdrojového kódu, než kdyby pokračoval v projektu dle časového plánu.

[16] Zadavatel projektu říká své představy, analytik je zaznamenává a následně jej tlumočí vývojářovi. Analytik má znalosti jak z byznysového, tak z vývojářského prostředí.

Obr. 3 Komunikační síť všech účastníků SW procesu



Zdroj: autorka

Výsledkem komunikační sítě je průnik všech komunikačních cest a jejich zdokumentovaných výstupů. *„Když se tento průnik pojme dobře, dostanete zajímavý systém, nadšené zákazníky a spokojené vývojáře. Když se pojme špatně, dostanete zdroj nedorozumění, frustrací a konfliktů, které snižují kvalitu systému a jeho hodnotu pro zákazníka.“* (WIEGERS, K. E. 2008)

Softwarový proces vychází ze softwarového inženýrství, které lze definovat jako *„zavedení a používání řádných inženýrských principů tak, abychom dosáhli ekonomické tvorby softwaru, který je spolehlivý a pracuje účinně na dostupných výpočetních prostředcích.“*[17] (KADLEC, V. 2004) Softwarové inženýrství[18]

[17] Uvedenou definici poprvé použil Fritz Bauer, jeden z duchovních otců softwarového inženýrství, na konferenci NATO v roce 1968.

[18] Před nástupem softwarového inženýrství (konec šedesátých let 20. století) otřásla světem softwarová krize. *„Charakteristickými znaky softwarové krize bylo neúnosné prodlužování a prodražování projektů, nízká kvalita programů, nesnadnost/nemožnost údržby a inovací, špatná produktivita práce programátorů, neefektivita vývoje, nejistota výsledku (do posledního okamžiku bylo nejasné nejen to, zda se projekt vyplatí, ale dokonce i to, zda se vůbec dokončí) a řada dalších nepěkných znaků.“* (KADLEC, V. 2004)

poukazuje na správné složení vývojových týmů, správnou volbu vývojového nástroje, důležitost nalezení společné řeči i společného pohledu se zadavatelem, na vhodnou volbu metodiky a uvážlivě sestavený časový plán. Mimo jiné varuje před podceněním hrozeb a rizik nebo před použitím nezvládnutých či nepoužívaných technologií.

1.2 Metodiky vývoje softwaru

Je důležité, aby jednotlivé kroky softwarového procesu byly důsledně promyšlené ještě před samotným psaním zdrojových kódů[19]. Není výjimkou, že nastávají vážné problémy s časem, rozpočtem i funkčností. V praxi se běžně stává, že vývojový tým nestihne předat dokončený software zákazníkovi ve stanoveném termínu nebo neúmyslně přesáhne stanovenou výši nákladů. Za příčinu nesplnění zákazníkových požadavků na fungování softwaru lze považovat například nedostatečnou komunikaci mezi vývojovým týmem a oddělením, pro které je softwarové řešení dodáváno. Mnohdy se i stane, že zákazník v počátku vývoje nemá ujasněny požadavky na funkcionality, které by mělo softwarové řešení splňovat. Často má zákazník pouhou představu o fungování softwaru, což k uspokojivému výsledku nemusí vést.

Těmito problémy se zabývá vědní disciplína zvaná metodologie, která představuje nauku o metodikách. Zabývá se metodami a možnostmi řešení jednotlivých problémů, se kterými se vývojářský tým během vývoje softwarového projektu může setkat.

„Metodika budování IS/ICT definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu informačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení.“ (BUCHALCEVOVÁ, A. 2005) Jedná se o souhrn metod a postupů, které vedou

[19] Steve McConnell (2005) ve svém díle Dokonalý kód uvádí, že *„kolem vývoje softwarových produktů vznikl přímo dezorientující počet metafor. David Gries říká, že psaní softwaru je věda (1981). Donald Knuth zase říká, že je to umění (1998). Watts Humphrey napsal, že je to proces (1989). P. J. Plauger a Kent Beck říkají, že je to jako řídit auto, ačkoli oba došli téměř ke zcela opačným závěrům (Plauger 1993, Beck 2000). Alistair Cockburn říká, že je to hra (2002).“*

Softwarové metafory mohou vést ke skutečnému pochopení hluboké podstaty vývoje softwaru. Další příklad metafory lze uvést softwarové farmaření, které je vysvětleno na postupném pěstování systému. To znamená, že farmář (vývojář) zasadí semínko (definování problému, který má být SW nástrojem řešen), které farmář svou činností pěstuje (vyvíjí) dokud ze semínka nenaroste výsledný produkt požadovaných kvalit (splnění cílů SW projektu).

k realizaci vývoje softwarového projektu, a tím i k dosažení vytyčených cílů a ke splnění požadavků. Cílem metodiky je formálně vymezit postupy a etapy, a přesně definovat odpovědnost v určitých fázích vývoje softwaru.

V praxi by se měla kvalita SW odvíjet od kvality použitého vývojového procesu. A proto, i vzhledem k různorodosti vývojářského prostředí, lze rozlišovat řadu odlišných[20] metodik. Za rozlišovací způsobilost lze považovat zaměření, rozsah, váhu metodiky či typ a přístup k řešení. Jako příklad zaměření metodiky lze uvést rozdíl mezi globální a projektovou metodikou. Globální metodika bude použita pro budování komplexního IS podniku. Projektovou metodiku lze použít v té samé společnosti pro vývoj softwarového nástroje sloužícího pro vizualizaci výstupních dat z IS podniku.

Vývoj informačních systémů a softwarových nástrojů by měl být vždy v souladu s podnikovou architekturou (*Enterprise Architecture*)[21], ve které by měl mít podnik jasně definované dlouhodobé cíle a poslání firmy s propojením na podnikové ICT. Ve 21. století by měla být EA společnosti natolik flexibilní a pružná, aby bylo možné zcela jednoduše a efektivně vytvářet požadavky na štíhlý[22] vývoj SW. Kvůli dynamickému prostředí vznikl tlak na využívání agilních[23] metod vývoje SW, kdy vývojový tým pravidelně komunikuje se zadavatelem a projednává dílčí výsledky SW projektu. Agilní metoda je takovou ideální verzí, ovšem za předpokladu, kdy zadavatel SW projektu má alespoň základní technické i metodické znalosti a zná detailně svůj byznys a související potřeby a požadavky na vyvíjené SW řešení. Problémem tohoto přístupu může být právě ona komunikace se zadavatelem SW projektu, který nemusí dobře porozumět dílčím výsledkům práce vývojového týmu,

---

[20] Příčinami odlišných metodik mohou být lidé, firemní kultura, použité technologie, priorita projektů, velikost týmu, zvolený způsob komunikace (fyzická přítomnost na poradách či online schůzky), i ostatní specifikující požadavky (např. přesně stanovený rozpočet nebo konkrétní termín dodání, bezpečnostní kritéria při vývoji softwaru pro státní správu).

[21] *„Cílem podnikové architektury je provázat podnikové cíle, strategie a stávající podnikové procesy s ICT technologiemi. Výsledkem podnikové architektury by měl být efektivní systém, díky kterému v dlouhodobém horizontu podnik uspoří finanční prostředky, zefektivní a zautomatizuje podnikové procesy, optimalizuje využití stávajících lidských zdrojů i sesbíraných (dosud nevyužitých) podnikových dat. Dalo by se říci, že modelováním podnikové architektury vzniknou schémata, která jednoduše a srozumitelně vyjádří složitou realitu podniku a jeho strategických cílů.“* (JIRÁNKOVÁ, J. 2022)

[22] Štíhlý vývoj se snaží o realizaci SW v nejkratším možném čase se snahou o dodání správně fungujícího SW s minimální pravděpodobností pozdějších rozsáhlých úprav po plném zavedení do podnikových procesů.

[23] Agilní metody umožňují rychlý vývoj SW i snadnou reakci na případné změny požadavků i během průběhu vývojového cyklu.

díky čemuž může odsouhlasit část, která nebude přímo korespondovat s jeho potřebami a požadavky na vyvíjený SW nástroj. Vývojový tým jako dodavatelská firma zároveň může tímto odsouhlasením dílčích fází vývoje přesunout odpovědnost za nesprávně fungující SW řešení na zadavatele projektu. Právě díky modernímu přístupu agilního vývoje mohou vzniknout nekonzistentní a nesystematické SW řešení, která v dlouhodobém horizontu firmu nejenže v technologickém rozvoji neposunou, ale naopak ji velmi zatíží. Dalším zásadním problémem agilního vývoje může být ono překročení termínu finálního předání SW řešení, neboť jakákoliv další změna učiněná v průběhu vývoje prodlužuje čas potřebný k dokončení SW i jeho původní finanční rozpočet.

1.3 Modely vývoje softwaru

Z metodologie vychází cyklus, který celý proces vývoje softwaru dělí na etapy. Jednotlivé fáze představují posloupnost kroků vedoucích ke splnění stanovených požadavků v odpovídajícím termínu s minimální pravděpodobností výrazného překročení stanoveného rozpočtu a s uspokojením potřeb zákazníka vztahujících se na správné fungování softwarového řešení. Průběh jednotlivých fází a s ním spojený soubor procesů, činností, úloh a představ o tvorbě softwaru, je upravován konceptuálními modely popisující životní cyklus vývoje SW.

Jednotlivé modely životního cyklu vývoje SW odráží dobu, ve které vznikaly. V minulosti byly modely založeny na jednoduchosti a sekvenční chronologii. Postupem času, od vzniku softwarového inženýrství, vznikla celá řada tradičních i moderních modelů, které mají (nebo by měly mít) společné čtyři klíčové pilíře (SOMMERVILLE, I. 2013), které jsou pro vývoj softwarů zásadní:

- specifikace softwaru;
- návrh a implementace softwaru;
- validace softwaru;
- evoluce softwaru.

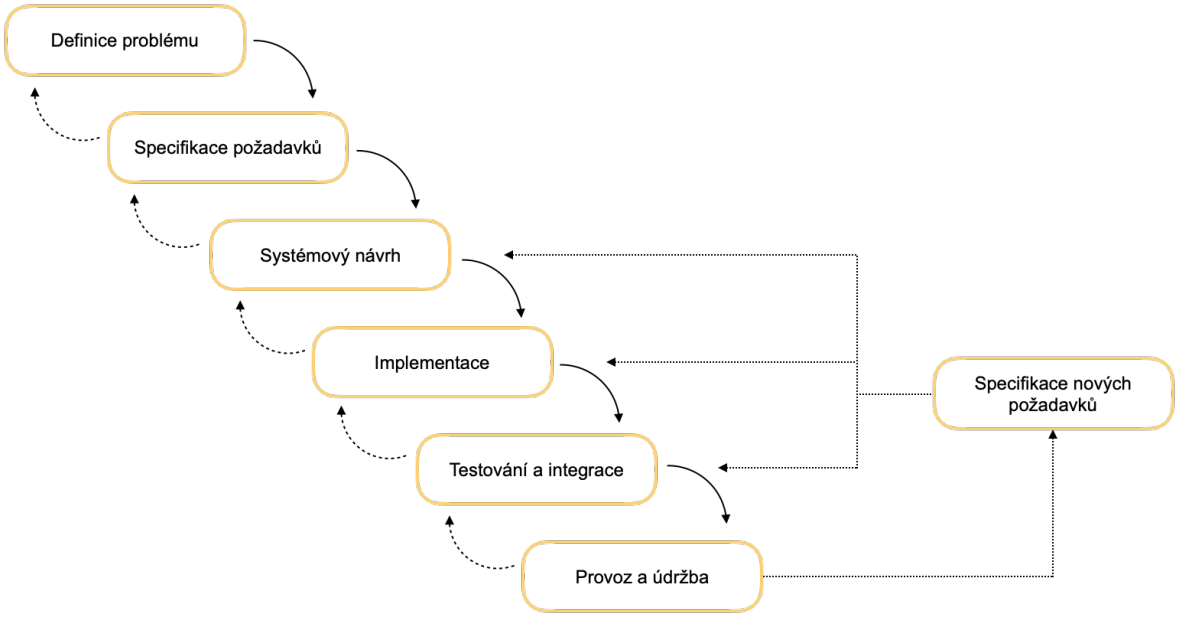
Dalo by se říct, že důslednost a preciznost při dodržování těchto pilířů vývoje softwaru odráží stupeň kvality výsledného softwarového projektu.

Za tradiční modely životního cyklu, které jsou z hlediska historie softwarového inženýrství považovány za významné milníky a podle kterých vznikla celá řada licencovaných i méně známých SW, lze považovat:

- vodopádový model,
- spirálový model,
- Rational Unified Process (RUP).

Vodopádový model[24] se vyznačuje sekvenčními fázemi, díky kterým je krok po kroku vyvíjen SW. Vývojový tým se posouvá vždy o jeden krok vpřed nebo zpět. Jednotlivé fáze vývoje znázorňuje následující schéma.

Obr. 4 Schéma vodopádového modelu



Zdroj: WIDEMAN, R. M. *A literature review – The Role of the Project Life Cycle (Life Span) in Project Management* (vlastní úpravy)

Softwarový projekt je vyvíjen postupně, od definice problému a specifikace požadavků až po testování a integraci a následný provoz SW v běžné praxi.

Zásadní slabinou vodopádového modelu je zpětná vazba a kontrola správného fungování SW až ve finální fázi provozu a údržby, kdy jsou případné úpravy

[24] Vodopádový model byl v roce 1970 formálně popisován v odborném článku Winstona W. Roycea jako nepružný, nefungující a chybně uchopený model používaný k vývoji software.

a opravy, vyplývající z nově požadovaných specifikací požadavků, nákladným a časově náročným řešením.

Jednoduchost tohoto modelu se striktně danou chronologií může být na škodu zejména u složitých projektů. V případě, že by se ve fázi testování zjistil nový požadavek na funkcionality, musel by se vývoj softwarového řešení vrátit na úplný začátek. Další zásadní nevýhodou je absence zapojení zadavatele projektu, který nemá dohled nad jednotlivými fázemi projektu, a tudíž nemůže poskytovat připomínky již v průběhu vývoje SW řešení.

Spirálový model[25] více rozvíjí vodopádový model a vyplňuje jeho zásadní nedostatky, neboť v procesu vývoje je kladen důraz na interaktivní přístup a důsledné zohlednění rizik[26] v raném stádiu vývoje SW i v jeho průběhu. Životní cyklus spirálového modelu prochází čtyřmi základními oblastmi, které pomáhají dlouhodobě udržovat konzistentnost a směr vyvíjeného SW projektu (BOEHM, B. W. 1988):

- Určení cílů, alternativ, omezení (*Determine objectives, alternatives, constraints*).
- Vyhodnocení alternativ, identifikace a řešení rizik (*Evaluate alternatives, identify, resolve risks*).
- Vývoj a verifikace další úrovně produktu (*Develop, verify next-level product*).
- Plánování následujících fází (*Plan next phases*).

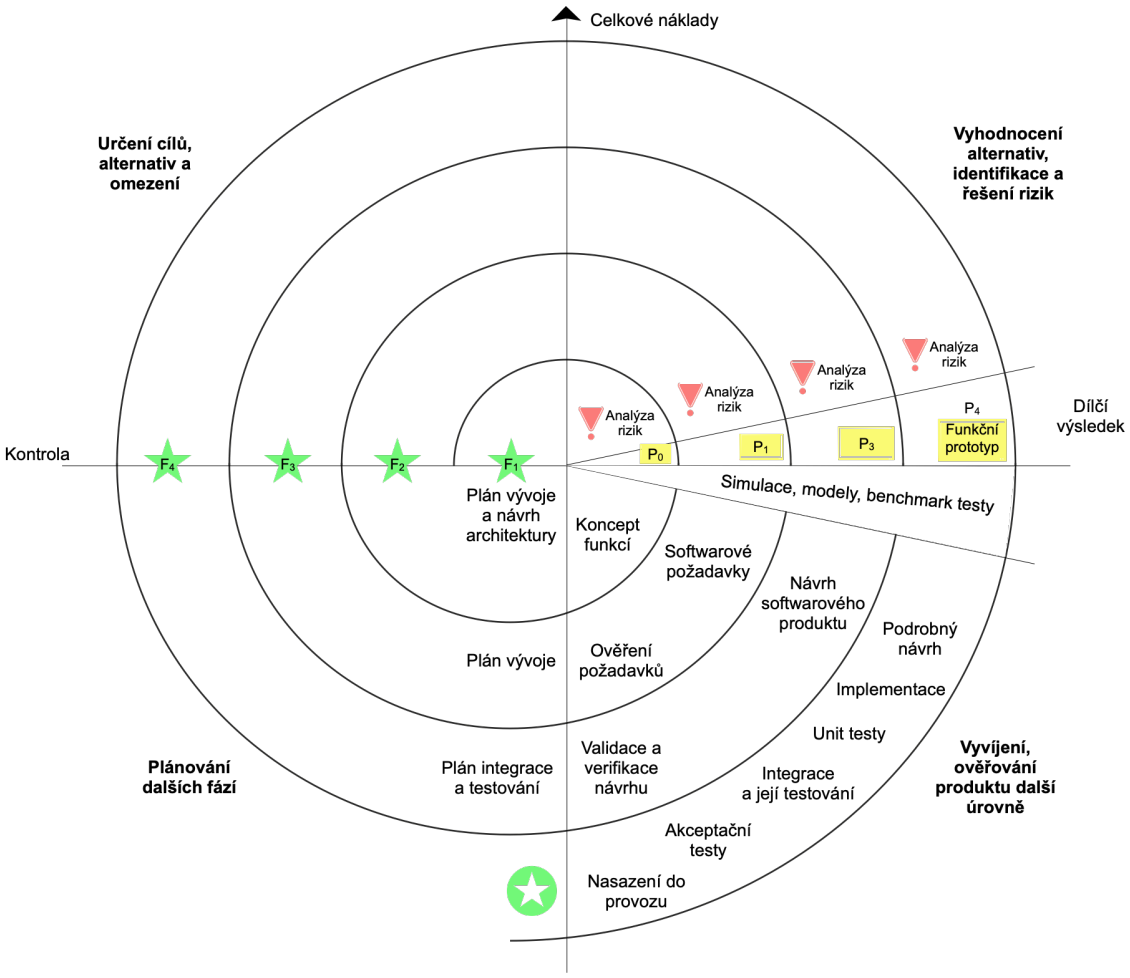
Spirálový model vychází z celkového plánu a z návrhu architektury. Z tohoto výchozího bodu postupně zohledňuje jednotlivá specifika vyvíjeného SW, která jsou v každém cyklu detailně charakterizována. Jelikož se v každém cyklu provádí analýza rizik, je snadnější určit další směr vyvíjeného softwarového projektu na základě přiřazených hodnot pravděpodobnosti výskytu rizika a jeho nebezpečnosti vůči projektu. Právě díky těmto dvěma ukazatelům lze včas zjistit možná ohrožení budování softwarového projektu, díky čemuž lze připravit možné scénáře řešení všech krizových situací. Díky pravidelným analýzám rizik je vysoká

[25] Tvůrcem spirálového modelu je Barry Boehm, který jej poprvé definoval ve svém vědeckém článku *A Spiral Model of Software Development and Enhancement* v roce 1986.

[26] V tomto pojetí jsou rizika vnímaná jako veškeré myslitelné situace, které by mohly ohrozit vývoj softwarového projektu.

pravděpodobnost, že dojde k zabránění vývoje špatného nebo nesprávně fungujícího SW projektu, který neodpovídá celkovému plánu a výchozím požadavkům. Pokud by se měl vývoj ubírat špatným směrem, dojde k odhalení vždy v průběhu samotného procesu vývoje SW, nikoliv až v jeho finální fázi jako například u vodopádového modelu.

Obr. 5 Schéma spirálového modelu



Zdroj: BOEHM, B. W. *A Spiral Model of Software Development and Enhancement. 1988* (vlastní úpravy)

Nevýhodou spirálového modelu pro malé či méně náročné projekty je jeho komplexnost a složitost. Naopak u rozsáhlých[27] softwarových projektů mohou být

[27] Softwarový projekt lze rozdělit dle rozsahu různě. Za nejvíce zřejmé a hmatatelné metriky lze považovat finanční rozpočet (*„Kolik Kč je zadavatel ochoten do projektu pustit?“*), vstupující zdroje (*„Jaké lidi a jejich znalosti a dovednosti, zařízení a nástroje má vývojový tým k dispozici*?“), velikost rozsahu (*„Kolik use-case obsahuje analytický dokument?“*), geografický záběr (*„V jakých zemích bude SW nástroj využitelný?“*).

tyto dvě slabé stránky základním stavebním kamenem kvalitního vývoje SW, neboť klade důraz na důsledné plánování, pravidelně opakující stanovení důležitosti a nebezpečnosti možných rizik a včasné vyloučení všech nepoužitelných řešení. Obecně lze spirálový model považovat za nepružný, těžkopádný a komplikovaný, a proto se v současné době uplatňuje jen zřídkakdy.

Rational Unified Process[28] je detailně propracovaná a systematická metodika vývoje softwarových produktů, která provází vývojáře interaktivními cykly a jejich dílčími fázemi. Metodika je určena zejména pro rozsáhlé softwarové projekty, neboť klade důraz na plánování, řízení lidských zdrojů, vizuální modelování, podrobné analýzy a pravidelnou dokumentaci. Vývojový tým má k dispozici webovou aplikaci, která ho provází vývojem SW. Model je založen na několika principech, které se v předchozích dvou tradičních modelech nevyskytují:

- iterační vývoj SW řešení,
- správa požadavků na SW řešení,
- využití již existujících SW komponent,
- vizualizace modelu SW řešení,
- průběžné ověřování kvality SW řešení,
- řízení změn.

Iterační vývoj SW řešení je založen na principu opakování určitého cyklu aktivit v měnícím se kontextu s cílem upřesňovat cílovou podobu prostřednictvím postupného rozšiřování z hrubé verze do výsledné podoby. SW nástroj je tak vyvíjen v postupných verzích, jejichž správnost lze pravidelně ověřovat a případné změny zanést do následující iterace. Výsledkem každé iterace je k dispozici spustitelný zdrojový kód, a proto lze ověřování správného fungování vyzkoušet testováním na reálných duplicitních podnikových datech s následnou konzultací výsledků testování se zadavatelem SW projektu.

Správa požadavků na SW řešení je klíčovým a nezbytným krokem, bez kterého nelze dosáhnout kvalitních výsledků. Metodika RUP popisuje, jakým způsobem zjišťovat požadavky na SW řešení u zadavatele, jak tyto požadavky systematicky

[28] Rational Unified Process (RUP) je považována za komerční metodiku, kterou vyvinula softwarová společnost IBM se snahou poukázat na možnosti systematičtějšího přístupu vzniku SW.

dokumentovat a efektivně spravovat v rámci softwarového procesu a jak s nimi v iteracích soustavně pracovat napříč vývojovým týmem.

Využití již existujících SW komponent je v posledních letech zcela běžnou záležitostí. Vývoj SW nástrojů se přesouvá do oblasti skládání nově vznikajícího SW nástroje z prefabrikovaných znovupoužitelných komponent. *„Při stavbě softwaru je však snaha vytvořit opravdu originální díla méně efektivní než snaha o opětovné použití existujících návrhových myšlenek, kódů a testovacích případů použitých v předchozích projektech.“* (MCCONNELL, S. 2005) Díky využití již existujících SW komponent lze získat čas, ušetřit finance a minimalizovat vznik chyb mající vliv na nefunkčnost či bezpečnost SW.

Vizualizace modelu SW řešení je založena na modelovacím jazyku UML[29], díky němuž lze vizualizovat strukturu a chování výsledného SW řešení, jeho procesů a souvisejících komponent v podnikové architektuře (EA). Díky této vizualizaci lze odhalit případné chyby v aplikační struktuře.

Průběžné ověřování kvality SW řešení je nedílnou součástí softwarového procesu, neboť zasahuje do všech aktivit a týká se všech zainteresovaných účastníků. Lze využít různých metrik, od objektivních měření až po kritéria kvantifikující kvalitu výsledného SW.

Řízení změn je velmi důležitou činností, přičemž ovlivňuje průběh jednotlivých iterací, koordinaci účastníků softwarového procesu i výslednou kvalitu SW. Kvalitní řízení změn dokáže implementovat mnohdy radikální a časté změny do následné iterace. Jako ukázkový příklad lze uvést účetní SW, který může být vystaven nutné změně v souladu novelizované účetní legislativy. Aktualizaci legislativního rámce účetního SW provedou ti vývojáři, kteří na této dílčí části SW pracovali. Nikoliv ti, kteří pracovali na podobě uživatelského rozhraní.

V posledních letech se dostávají do popředí agilní metodiky[30] vývoje softwaru, které upřednostňují vývoj SW projektu před jeho detailním plánováním.

[29] UML (*Unified Modeling Language*) hraje významnou roli ve vývoji SW. Vznikl na základě požadavku vytvořit jazyk, který by byl méně chaotickým popisem stále složitějších SW produktů. Jeho cílem je zjednodušovat složitosti, řešit problémy s podnikovou architekturou, zlepšit kvalitu práce či snižovat náklady a čas.

[30] Základním pilířem agilního přístupu vývoje softwaru je manifest z roku 2001 (The Agile Manifesto), který vychází ze dvou základních tezí (KADLEC, V. 2004):

Bývá charakteristickým pravidlem, že agilní způsoby vývoje SW mají zafixovaný rozpočet a termín odevzdání[31]. Rozsah funkcionalit bývá variabilní, což znamená, že vývojový tým může softwarový projekt na základě požadavků zadavatele upravovat v průběhu softwarového procesu i několikrát. Ovšem, neztrácí vývojový tým právě díky těmto možnostem onen nadhled na SW v rámci business prostředí a onen pohled na skutečné potřeby klienta? Agilní metody vývoje SW by mohly fungovat v ideálním světě, kdy zadavatel má jasnou a správnou představu o svých potřebách a požadavcích na vyvíjené SW řešení a dokáže tyto požadavky srozumitelně a včas předat vývojovému týmu. Příkladem agilního vývoje SW může být extrémní programování (XP)[32] nebo Lean Development[33].

Agilní přístup vývoje SW a jeho nízký stupeň formalizace softwarového procesu může sehrát významnou roli i při oceňování SW projektu. Nelze obecně říct, že kvůli neobsáhlé projektové dokumentaci, nízké formalizaci či jednoduchosti[34] by byl SW nekvalitní či bezcenný. V této rovině bude pravděpodobně záviset na odborných znalostech a zkušenostech oceňovatele, aby dokázal posoudit, jakých kvalit a hodnot posuzovaný SW dosahuje. Tato skutečnost potvrzuje pravidlo, že některé aktivity softwarového procesu (jako například oceňování či IT audit) by neměly být

1. Přijmout a umožnit změnu je mnohem efektivnější, než pokoušet se jí zabránit.
2. Je třeba být připraven reagovat na nepředvídatelné události, neboť ty bezpochyby nastanou: tento bod lze formulovat také tak, že jedinou jistotou je změna.

Agilní manifest se *„v roce 2001 stal prostředkem k vyjádření nespokojenosti s tradičními metodami vývoje softwaru a svazováním vývojářské kreativity byrokratickými procesními koncepty a přemírou dokumentace. Obdobně brojí proti upozadění reálných potřeb koncového uživatele jako klíčového činitele v procesu vývoje softwaru. Dalším základním tématem je připravenost na změnu. Mění se požadavky, lidé v projektu, používané technologie. Proto ani vývojový proces nemůže zůstat statický, ale pomocí krátkých smyček zpětné vazby musí docházet k jeho přizpůsobení.“* (BUREŠ, M. a kol. 2016)

[31] Jedná se o smluvní typ Fixed-Time Fixed-Price (FTFP). V českém právním prostředí je ekvivalentní Smlouva o dílo dle § 2586, odst. 1, nového občanského zákoníku.

[32] Extrémní programování je považováno za jednu z nejznámějších metodik agilního vývoje SW. Je založeno na vývoji SW v krátkých cyklech s důrazem na dodání jednoduchého a pro zadavatele potřebného SW řešení, a také na úzké spolupráci mezi zadavatelem a vývojovým týmem s pravidelnou potřebou zpětné vazby.

[33] Lean Development je přístup zaměřený na efektivní a rychlý proces vývoje s eliminací všech zbytečností, které zadavateli nepřinášejí žádnou přidanou hodnotu, s důrazem na převzetí odpovědnosti.

[34] Jak uvádí Ka Wai Cheung (2013), vývojář a designer, ve své knize Vývojářův kód: *„Když vytvoříme něco jednoduše, připadá nám, že tomu něco chybí. Vsugerujeme si, že zákazník za své peníze nedostává odpovídající protiváhu. Skoro se zdá, že jednoduchá věc, kterou je zároveň snadné vyrobit, nemá žádnou hodnotu. Snadno realizovatelný nápad se většinou za „převratný“ nepovažuje.“*

vykonány pouze na základě striktně daných pravidel a tabulek bez důsledného zhodnocení specifik a přínosů plynoucích pro konkrétní subjekt.

Z principů agilního vývoje vychází přístup DevOps, který lze definovat jako soubor pravidel a doporučení s důrazem na kontinuální zlepšování komunikace, spolupráce a integrace mezi vývojovým týmem (Dev) a provozem (Ops). DevOps přístup není jen o dodržování stanovených postupů, ale o změně myšlení a o důsledném hledání všech vhodných řešení vedoucích ke spokojenosti zadavatele. Vývojový tým má možnost se na požadavek zadavatele dívat ze všech myslitelných úhlů pohledu, například z pohledu provozu. Jen díky poznání byznys procesů zadavatele může vývojový tým dostatečně posoudit, jestli má zadavatel správnou představu o svých potřebách.[35]

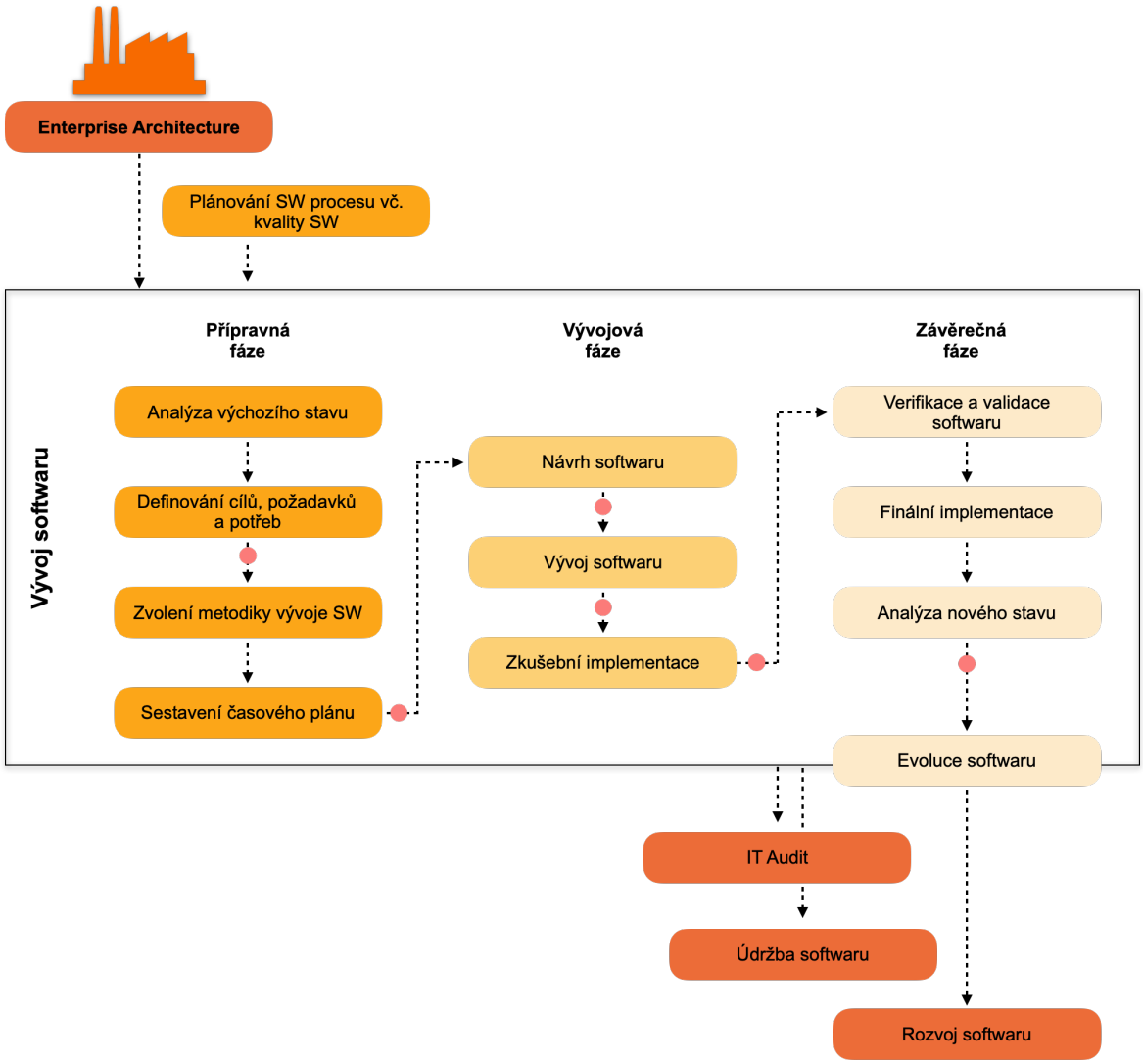
V praxi není vyloučena možnost kombinování metodik při vývoji jednoho SW projektu, neboť je mnohdy SW rozdělen do několika modulů. Z tohoto důvodu je možné na jednotlivé moduly použít tu metodiku, která bude pro vývoj daného modulu efektivní ze všech třech různých hledisek (rozpočet, čas, rozsah) a bude dosahovat kýžených výsledků kvality. Nelze tedy vyloučit přístup vývoje, který bude založen na vhodné synergii tradičních a agilních metod.

1.4 Životní cyklus vývoje softwaru

Životní cyklus si lze představit jako souvislou návaznost jednotlivých kroků vývoje SW. V rámci životního cyklu lze rozlišit vlastnosti jednotlivých etap životního cyklu softwarového procesu, mezi které patří například cíl jednotlivých etap, jejich účel a obsah, předpokládané období zahájení etapy či kritéria jejího ukončení. Součástí je i veškerá související dokumentace a návaznosti jednotlivých činností. Všechny tyto vlastnosti souvisí s důkladným řízením projektu v rámci softwarového procesu.

[35] Z tohoto důvodu by mohli být multioboroví členové vývojového týmu značným přínosem.

Obr. 6 Životní cyklus vývoje softwaru[36]



Zdroj: autorka

Vývoj specializovaného SW či dílčích softwarových nástrojů, doplňujících již zavedený IS, by měl být v plné, pokud možno ideální, symbióze s již fungujícím podnikovým informačním systémem a podnikovou architekturou (EA). Vývojový tým by měl brát v potaz dosavadní ICT vyspělost zadavatele, již zavedený podnikový informační systém a jeho doplňující specializované SW. Klíčovou úlohou

[36] Tvar ● představuje klíčový kontrolní milník, díky němuž může vývojový tým zhodnotit dosavadní práci a detekovat případné softwarové chyby nebo nesrovnalosti v softwarovém procesu.
Za vhodné kontrolní metody lze považovat například testování a jeho řádné dokumentování. *„Dostatečná, správná a kontinuální dokumentace testovacích případů však pro projekt přináší řadu výhod.“* (ROUDENSKÝ, P. a kol. 2013) Za výhody lze považovat například vyšší pravděpodobnost odhalení nedostatků ve specifikaci či požadavcích nebo možnost pravidelné kontroly nad testováním a odhalením případných nedostatků. Dokumentace o testování může sloužit jako podklad pro oceňovatele SW projektu.

vývojového týmu by měl být, mimo jiné, odhad[37], jakým směrem se v rámci technologické vyspělosti bude firma nadále rozvíjet. Díky tomuto uvědomění by mohl vývojový tým nalézt efektivní cestu vývoje SW řešení, které bude nadčasové a použitelné i v době technologického zrání firmy.

Plánováním SW procesu a požadované kvality výsledného softwaru se sice automaticky nezajistí vyšší pravděpodobnost úspěšně dokončeného SW projektu a jeho kvality, nicméně dojde napříč všemi zúčastněnými subjekty k ujasnění požadavků a dílčích kroků, které je nutné k úspěšnému cíli učinit. Právě díky plánování lze předejít nepředvídatelným a neujasněným změnám. Za změnu lze považovat přidání vlastnosti, opravu chyby, refaktorizaci či optimalizaci. Na druhou stranu, *„software je něco, co není nikdy hotovo“* (HEROUT, P. 2016), a proto se nelze v digitálním dynamickém světě změnám zcela vyhnout.[38] Podle organizace The Standish Group[39] je až 31 % SW projektů přerušena, u poloviny[40] SW projektů dojde k navýšení původně zamýšlených nákladů až o dvojnásobek a pouze 9 % SW projektů je splněno v dohodnutém termínu.

Analýzou výchozího (současného) stavu podniku a podnikových procesů se rozumí vymezení stávajících podnikových postupů a měřitelných i neměřitelných kritérií,

---

[37] Tento odhad bude založen na důsledné konzultaci mezi vývojovým týmem a zadavatelem, na jejich vzájemném porozumění a následném, pokud možno, ideálním pochopení byznysu, smyslu a filosofie zadavatelské společnosti.

[38] V takovém případě je nutné uvědomit si, že jakákoliv změna může být vzhledem k zachování funkčnosti SW riskantní. S tímto vědomím a položením si několika otázek (FEATHERS, C. M. 2009) lze snížit riziko změny:

1. *„Jaké změny musíme provést?“*
2. *"Jak víme, že jsme je provedli korektně?“*
3. *„Jak víme, že jsme nic jiného nepokazili?“*
4. *„Kolik změn si můžeme dovolit, pokud jsou změny riskantní?“*

Lze říci, že účelná schopnost změny v dynamickém IT světě je pro vývoj SW zásadní. Z tohoto důvodu se od roku 2001 stává populární agilní přístup vývoje, který má schopnost změny při zachování funkčnosti zakomponovanou v Agilním manifestu (ROUDENSKÝ, P. a kol. 2013):

- Lidé a jejich vzájemná interakce jsou důležitější než procesy a nástroje.
- Fungující software je důležitější než vyčerpávající dokumentace.
- Spolupráce se zákazníkem má přednost před vyjednáváním smluv.
- Je důležitější reagovat na změny než dodržovat plán.

„Neznamená to ovšem, že hodnoty na pravé straně si agilní vývoj necení. Agilní Manifest pouze hlásá, že hodnot vlevo si máme cenit víc. Jde ve své podstatě o návod na rozhodování.“ (BŘEZINA, A. 2020)

[39] Nezávislá IT společnost, která se mimo jiné zabývá i průzkumy a analýzami (ne)úspěšnosti SW projektů.

[40] Z této poloviny úspěšně dokončených projektů se pouze necelých 80 % SW projektů skládá z původně zamýšlených funkcionalit.

ze kterých vychází požadavky na fungování vyvíjeného softwaru a k definování konkrétních cílů. Mezi takové problémy mohou patřit například rutinní činnosti, činnosti s vysokou pravděpodobností selhání lidského faktoru (lidské chyby), vysoké náklady na provoz, neefektivní komunikace mezi dodavateli a odběrateli, či mezi firmou a státem, ukazatele poukazující na finanční zdraví podniku (tržby, marže aj. finanční ukazatele), výrobní ukazatele (výrobní cyklus, efektivita výroby aj.). Účelem nasazení nově vznikajícího softwaru do podnikového procesu by mělo být splnění specifických potřeb subjektu.

První fáze by neměla opomenout ani na analýzu výchozího stavu ICT zralosti podniku, která se v různých odvětvích, velikostech podniků i v jejich zaměření může značně lišit.

Druhým nezbytným krokem je definování požadavků a cílů, kterých má být pomocí nově nasazeného softwaru do podnikového procesu dosaženo. Za cíle lze považovat například zefektivnění podnikových procesů, úsporu nákladů či konkurenční výhodu (zajišťující silnější pozici na trhu). Mezi požadavky na nově vznikající systém lze zařadit funkcionality, vizuální podobu SW, plynulou návaznost na informační systém podniku či jiné podnikové softwary. V rámci tohoto kroku vývoje je nesmírně důležité znát (a během softwarového procesu si neustále uvědomovat) odpověď na otázku *„Proč se SW vyvíjí?“.* Po zodpovězení otázky by měli všichni účastníci vývoje dojít k závěru, že vyvíjený SW je nejdůležitější částí projektu. *„Jakékoliv rozhodnutí o vlastnostech programu může být podrobeno jednoduché otázce: „Bude díky tomu aplikace lepší?“ Osobní pocity a záměry nás ani klientů nejsou rozhodující. Pokud není produkt sám v popředí zájmu, klienti se mohou uchýlit k ospravedlnění žádosti o nový prvek jiným způsobem:*[41]*“* (CHEUNG, W. K. 2013)

Volba metodiky vývoje softwaru představuje utřídění procesů, praktik, rolí i technik vývoje softwarového projektu. Výstupem je písemný metodický pokyn, kterým by se měli účastníci vývoje softwaru řídit ve všech fázích. Metodický pokyn by měl být v souladu s prvky kybernetické bezpečnosti.

[41] *„… je to cool, viděl jsem to na jedné stránce.“; „… protože v mé knížce o použitelnosti psali, že obsah, kvůli jehož zobrazení se musí hýbat posuvníkem, nikdy nikdo nečte.“; „…protože panu řediteli se líbí růžová!“*

Za jeden z výstupů metodiky lze považovat časový plán, který představuje harmonogram jednotlivých činností. Aby se vývojový tým nedostal pod neúnosný časový tlak, je důležité jednotlivé kroky plánovat realisticky a počítat s chybami[42]. Lze uvést příklad nasazení SW projektu do podnikového prostředí. Pokud projektový manažer vhodným způsobem nenaplánuje finální implementaci, může se snadno stát, že se SW projekt řádně neotestuje, neodstraní se chyby způsobené implementací, nebo dojde k omezení provozu podniku zadavatele.[43] I v této fázi je klíčová komunikace mezi vývojovým týmem a zadavatelem, neboť by v rámci časového plánu měly být známy odpovědi na následující klíčové otázky:

- Jaký je termín implementace SW projektu do podnikového prostředí?
- Co se stane, když SW projekt nebude do daného termínu implementován?
- Co se nestane, když SW projekt nebude do daného termínu implementován?

Návrhování softwaru je považováno za vědní disciplínu i umění, u kterého je vždy nutné *„vynaložit určitý stupeň kreativity, což je obzvláště pravdivé ve chvíli, kdy před vámi stojí úkol navrhnout architekturu pro nějaký zcela nový systém, pro který dosud neexistuje žádný vzor.“* (EELES, P. a kol. 2011) Výsledkem navrhování je architektura, která byla v roce 2004 Philippem Kruchtenem, uznávaným softwarovým inženýrem, přirovnána k součtu všech důležitých rozhodnutí.[44] Podrobněji jej definoval v roce 2000 jako sadu *„zásadních rozhodnutí, týkajících se organizace softwarového systému, výběru prvků struktury a jejich rozhraní, vytvářejících celý systém, spolu s jejich chováním daným specifikací spolupráce mezi těmito prvky, postupnou skladbou těchto prvků do zvětšujících se subsystémů a architektonickým stylem, jímž je celá tato organizace řízena – tj. tyto prvky a jejich*

---

[42] Jak říká latinské přísloví římského filosofa Seneca: *„Chybovat je lidské.“* V rámci projektového řízení je důležité počítat s chybami, nejen technickými, ale i lidskými (neúmyslné selhání lidského faktoru). Pokud bude časový plán „našponovaný“, nebude prostor pro operativní úpravy a nápravy.

[43] Na časovém plánu lze vidět kvalitu řízení projektu. Projektový manažer by měl zvážit, jestli je implementace v plánovaný čas reálná, zvládnutelná a jestli existuje dostatek času na finální otestování.

Vhodnou dobou pro nasazení SW do podnikového prostředí je například doba pracovního klidu (víkend, svátky a jiné dny plánované odstávky provozu), kdy má vývojový tým dostatek prostoru na (ničím nerušenou) finální implementaci a závěrečné otestování.

[44] *„Software architecture = the sum of all important decisions. Important decisions are all decisions that are difficult to change in the course of further development.“* (LILIENTHAL, C. 2019)

rozhraní, jejich spolupráce a jejich skladba. (Kruchten 2000)“[45] (EELES, P. a kol. 2011) Jelikož se jedná o komplexní a velmi důležitý krok softwarového procesu, je nutné k této činnosti přistupovat obezřetně a komplikované části dokumentovat a popisovat. Toho lze docílit modelováním řešení problému pomocí vývojových diagramů a algoritmů[46] v nich obsažených. Vizuální znázornění SW a jeho fungování může usnadnit komunikaci mezi vývojovým týmem a zadavatelem SW projektu, díky čemuž se minimalizuje riziko vzájemného neporozumění a tím i prostor případných softwarových chyb. Díky vizualizacím může dojít, mimo jiné, i ke zefektivnění[47] algoritmu či procesu vývoje SW, neboť mohou sloužit jako kontrolní prvek při hledání efektivní cesty řešení problému, při psaní zdrojového kódu nebo při hledání softwarových chyb.

Je důležité si před samotným vývojem softwaru uvědomit, že *„předělávání v přeneseném smyslu výroby vede k nákladům, ale předělávání v oblasti návrhu vede k vyšším hodnotám.“* (MARTIN, C. R. 2009)

Samotný vývoj softwaru je jeden z nejnáročnějších a finančně nejnákladnějších kroků v projektovém vývoji softwaru. Software je psán v softwarovém (programovacím) prostředí pomocí programovacích jazyků[48] podle návrhu softwaru. Po překladu (kompilace) zdrojového kódu a následném sestavení (linker)

---

[45] Existuje celá řada odborných definic, ovšem všechny mají společné určité prvky. „Architektura se zabývá jak strukturou, tak i chováním; dále je zřejmé, že architektura se zabývá pouze zásadními prvky; může odpovídat nějakému stylu, je ovlivněna investory systému a prostředím, a je vyjádřením rozhodnutí, vycházejících ze základů.“ (EELES, P. a kol. 2011)

[46] *„Algoritmus je informatická verze pečlivě vypracovaného, přesného a jednoznačného*“ (KERNIGHAN, W. B. 2019) postupu s posloupností kroků, přesně danými vstupními daty a konkrétními představami, jakého výsledku má být dosaženo. *„Každý krok je vyjádřením některé ze základních operací, například „sečti dvě celá čísla“, jejíž význam je plně specifikován. Není zde žádná víceznačnost. Povaha vstupních dat je daná. Jsou pokryty všechny možnosti; algoritmus nikdy nemůže dojít do situace, kdy není jasné, co se má udělat dál. Když je informatik puntičkář, obvykle dodá ještě jednu poznámku: algoritmus se musí nakonec zastavit.“* (KERNIGHAN, W. B., 2019) Pojem algoritmus lze také vyjádřit jako abstraktní a idealizovaný popis, jehož výsledkem je program.

[47] Mnohdy vývojový tým napadnou *„dva postupy řešení, z nichž jeden je pomalejší a vystačí si s velmi malou pracovní pamětí, zatímco druhý je rychlejší, ale má vyšší paměťové nároky. Stává se totiž, že ke zrychlení výpočtu musíme použít nějakou pomocnou datovou strukturu, v níž budou uchovávány předem spočtené a připravené hodnoty nebo mezivýsledky. Nejrychlejší algoritmus tudíž nebývá optimální z hlediska paměťové složitosti a naopak algoritmus s nejmenšími paměťovými nároky zase nebývá nejrychlejší.“* (TÖPFER, P. 2002)

[48] Programovací jazyky lze rozdělit podle různých kritérií, například vyšší a nižší programovací jazyky, interpretované a kompilované programovací jazyky, objektově orientované nebo strukturované programovací jazyky. V posledních letech lze jazyky dělit subjektivně jako moderní a zastaralé. Jako příklady programovacích jazyků lze uvést C, C++, Java, Python, Ada, Cobol, Fortran, Pascal či jeden z nejstarších programovacích jazyků Assembler.

jednotlivých modulů relativních adres vznikne výsledná spustitelná (exe) verze[49] vybraného a realizovaného algoritmu. I při samotném vývoji softwaru by měla být prioritou kybernetická bezpečnost, neboť případné budoucí útoky mohou ohrozit funkčnost vyvíjeného softwaru či bezpečnost celého podnikového ICT i dat v něm obsažených.

Zkušební implementací softwarového projektu je myšleno nasazení softwaru na reálná duplikovaná podniková data. Cílem tohoto mezikroku je snaha o zabránění omezení běžných podnikových procesů a o řádné otestování SW projektu.

Zároveň lze v rámci zkušební implementace otestovat SW z hlediska jeho kybernetické bezpečnosti, což lze provést simulací kybernetických útoků, které prověří, jestli nemůže být SW řešení tím slabým článkem v celém podnikovém ICT. V praxi se může stát, že i skvělý a nákladný SW projekt může být slabým místem, neboť nemusí být správně implementován nebo může obsahovat skryté (neúmyslně vzniklé) softwarové chyby. Během simulace je nutné poznat mechanismus a proces kybernetických útoků a pokusit se vžít do pozice hackera[50]. *„To však není jednoduché, protože v reálném světě existuje obrovská množina variant jednotlivých útoků, která závisí nejen na konfiguraci sítí, ale především na znalostech a zkušenostech útočníků.“* (POŽÁR, J. 2012) Simulaci kybernetických útoků je nezbytně nutné provést před nasazením SW projektu do provozu, neboť *„jakmile se jednou útočník dostane do systému, je velmi těžké ho odtud dostat.“* (MCCLURE, S. a kol. 2003) I přes to, že by se vývojovému týmu podařilo po skutečném kybernetickém útoku nalézt způsob hackerského průniku a následně odstranit jeho příčiny (softwarové chyby), pravděpodobně bude i nadále SW zranitelný. Hackeři si pomocí svých mechanismů tvoří zadní vrátka, díky kterým se

---

[49] Takové spustitelné verzi se v odborné terminologii říká objektový kód, který je vhodnou podobou původního zdrojového kódu pro procesor. Díky objektovému kódu může procesor vykonávat jednotlivé příkazy.

[50] Dovednost „umět myslet a psát zdrojový kód jako hacker“ může mít pro řadu vývojářů přidanou hodnotu na trhu práce, neboť takový vývojář se považuje za člověka s inovativním duchem a hlubokou znalostí technologií. *„Hacking je termín užívaný jak těmi, kteří píší kód, tak těmi, kteří ho exploitují. I když mají tyto dvě skupiny odlišné cíle, obě používají podobné techniky k řešení problémů. A protože chápání programování pomáhá těm, co exploitují a chápání exploitování pomáhá těm, co programují, mnoho hackerů dělá oboje. Existují zajímavé hacky, které můžete najít jak v technikách sloužících k psaní elegantního kódu, tak v technikách sloužících k psaní exploitů. Hacking je v podstatě jen hledání chytrého a neintuitivního řešení daného problému.“* (ERICKSON, J. 2005)

Exploit je výraz v informatice používaný pro speciální software nebo sekvenci příkazů, jejichž cílem je využít softwarovou chybu k získání prospěchu.

mohou do systému kdykoliv nepozorovaně vrátit. Nalezení a odstranění těchto zadních vrátek je ve skutečnosti téměř nemožné[51].

Prvním krokem závěrečné fáze je verifikace a validace softwarového projektu. Verifikací se rozumí ověření softwaru, jestli vyvíjený softwarový projekt odpovídá návrhu softwaru. Verifikace odpovídá na otázku „*Je softwarový projekt vytvořen v souladu s návrhem softwaru a požadovanými cíli?“.* Aby byla verifikace splněna, je nutné softwarový produkt vyvinout správně (podle specifikace). *„Přičemž ale specifikace může být špatně a je to bohužel velmi častá příčina chyb.“* (HEROUT, P. 2016) Úlohou verifikace je najít chyby podle specifikace SW, nikoliv ověřovat (a zajistit) správnost návrhu SW. Mezi metody verifikace patří testování[52] softwarového projektu, revize zdrojového kódu, simulace a analýzy výstupních dat. Validace hodnotí správnost fungování softwaru a splnění požadavků stanovených zákazníkem. Výstupem validace je odpověď na otázky *„Pracuje softwarový projekt správně? Má všechny potřebné funkcionality?“.* Mezi validaci lze zařadit i posouzení srozumitelnosti uživatelského prostředí zadavatele projektu. Verifikace a validace na obou úrovních prověřují, jestli softwarový projekt odpovídá návrhu softwaru i požadavkům zadavatele projektu. *„Nalezení a opravení chyb nepomůže, pokud vytvořený systém nesplňuje potřeby a očekávání uživatelů a tím je de-facto sice bezchybný, ale prakticky nepoužitelný.“* (HEROUT, P. 2016) Během verifikace a validace by mělo rovněž dojít k vyhodnocení, jestli softwarový projekt nenarušuje kybernetickou bezpečnost již stávajícího podnikového ICT systému.

Pokud verifikace a validace není splněna, vrací se softwarový projekt zpět na úroveň vývoje, aby došlo k odstranění nežádoucích chyb a nepřesností. V případě, že by se při revizi zdrojového kódu objevily chyby umožňující vstup hackerů

---

[51] *„Prakticky jedinou možností, jak dostat systém do původního stavu, je reinstalace z originálních médií a obnova konfigurace a dat ze záloh. Kompletní obnova je velmi složitá, zvláště když byl systém unikátně nakonfigurován a jeho konfigurace není dokumentována.“* (MCCLURE, S. a kol. 2003)

[52] Testování má dvě fáze. První z nich je stanovení očekávaného výsledku testu, který může nabývat binárních hodnot („prošel“ nebo „neprošel - selhal“). Subjekt (fyzická osoba nebo počítač), který stanoví očekávaný výsledek je nazýván orákulem (oracle). Subjekt, který stanoví skutečný výsledek, je tester.

Jelikož náklady na testování softwaru vstupují do celkových nákladů na vývoj SW, je nutné stanovit bod zlomu. *„S hledáním dalších chyb by se mělo přestat v okamžiku, kdy náklady na nalezení a opravení chyby jsou v průměru stejné nebo větší než náklady na ponechání chyby. Samozřejmě, pokud se nejedná o bezpečnostně kritický SW, kde si chybu nemůžeme dovolit nechat, protože náklady na ponechání chyby by byly u kritických systémů velmi vysoké. A to tak vysoké, že je nutné ujistit se o neexistenci závažných chyb.“* (HEROUT, P. 2016)

do podnikového systému, je nutné softwarový projekt vrátit zpět vývojovému týmu k opravení. SW projekt se vrací o několik kroků zpět do vývojové fáze (návrh a vývoj SW), což má za následek vícenáklady a nedodržení časového plánu. Z těchto důvodů je vhodné zvážit zařazení simulace kybernetického útoku do vývojové fáze.

Pokud softwarový projekt po verifikaci a validaci softwarového projektu obstál, lze přejít k finální implementaci softwarového projektu do běžného podnikového provozu. K finální implementaci dochází buď za provozu, nebo mimo běžný chod společnosti.

Analýza nového stavu poukáže, nakolik byl softwarový projekt pro zlepšení podnikového stavu potřebný. Sledovanými ukazateli, jako v analýze výchozího stavu, lze měřitelně vyjádřit přínosy softwarového projektu na podnikové procesy i podnikový ICT systém jako celek. Tato analýza a její výsledky mohou být oceňovateli nápomocné při stanovení hodnoty SW projektu. Díky výsledkům analýzy může oceňovatel vyhodnotit, nakolik bylo SW řešení pro podnik přínosné, tzn. nakolik se jeho podnikové procesy zefektivnily, jaké jsou finanční úspory, jak se zlepšila kvalita dodávaných produktů zákazníkovi či jaké inovace SW řešení podniku skutečně přineslo. Jako příklad lze uvést VR vizualizace architektonických návrhů domů či komerčních objektů. Klient se díky těmto vizualizacím dokáže přenést do budoucího stavu své nemovitosti, díky čemuž může architektovi poskytovat přínosnou zpětnou vazbu, na jejímž základě lze upravovat vizuální návrhy finální podoby. I díky těmto VR vizualizacím má klient představu, jak jeho objekt bude vypadat a díky tomu může v mezičase s vizualizací určitým způsobem dále nakládat. Například, hledat vhodné řešení strojového vybavení továrny a jeho uspořádání v prostoru.

Údržbou softwaru se myslí udržovací fáze softwarového projektu v provozuschopném stavu. Do údržby softwaru spadá i aktualizace, díky níž lze odstranit případné chyby, které by mohly narušit funkčnost a konzistentnost podnikového ICT systému. I v této fázi je vhodné provádět test kybernetické bezpečnosti nejen softwarového řešení, ale i celého podnikového ICT.

Rozvojem softwaru se rozumí jeho přirozená evoluce, na kterou by mělo být pamatováno již v rámci softwarového procesu. Pokud byl softwarový proces řádně dokumentován, pokud byly požadavky na SW řešení v dokumentaci dostatečně

vymezeny a pokud byl zdrojový kód výsledného softwaru napsán srozumitelně, jasně a bez významných chyb, je možné SW řešení brát jako nadčasový prvek podnikového ICT, který lze nadále rozvíjet ve prospěch firmy.

Za důležitou součást vývoje SW lze považovat IT audit softwarového projektu v kontextu podnikového ICT. Může být významný zejména u podnikových či účetních softwarů, jejichž případné chyby, ať už úmyslné nebo neúmyslné, by mohly mít významný vliv na účetní závěrku společnosti i na její běžný chod. Z tohoto hlediska by se IT audit[53] mohl považovat za výraznou přidanou hodnotu finančního auditu i ICT bezpečnosti firmy. Podrobněji je příklad demonstrován v příloze 2.

IT audit, případně dílčí audit programového vybavení (SW projektu), může být dalším podkladem[54] oceňovatele při stanovení hodnoty SW projektu. IT audit nemusí být brán jako nutné zlo, nýbrž jako přidaná hodnota ICT systému podniku. Jeho úkolem je „*vytváření a prověřování komplexního systému kontrol z hlediska toho, zda podporují cíle podniku, zda stanovené standardy jsou aktuální a vyhovující a zda procesy v organizaci jsou efektivní a produktivní.*“ (SVATÁ, V. 2012) IT audit nepředstavuje kontrolu dílčích aspektů. Zdůrazňuje nejvyšší úroveň formalizovaného a nezávislého prověření konaného komplexně na všech možných úrovních a vazeb ICT podnikového systému, objektivně vzhledem k uznávaným standardům a zkušenostem IT auditora, s důrazem na nezávislé spojení IT auditora a zadavatele IT auditu, se snahou o pružné přizpůsobení se prostředí a všem myslitelným okolnostem s cílem systémově prosadit zájmy vlastníka. S jinými formami auditu[55] se překrývá, čímž utváří nejvyšší stupeň ujištění vzhledem k cílům a filozofii podnikového řízení. Úspěšný IT audit s příznivými výsledky auditovaného podniku přináší mimo jiné i ekonomickou přidanou hodnotu, která se může projevit při dostatečném zveřejnění informací o výsledcích provedeného IT auditu růstem cen akcií na burze a zvyšující se důvěrou externích subjektů v auditovaný podnik.

---

[53] Je důležité podotknout, že IT audit by neměl být prováděn striktně podle dané kuchařky a předepsaných tabulek. Každý SW projekt by měl být posouzen individuálně.

Mohlo by se totiž stát, že malý a vysoce kvalitní SW projekt by mohl být pouhým stínem nesrovnatelně většího SW projektu.

[54] *"Součástí závěrečné zprávy o provedeném auditu je popis oblasti a rozsahu provedeného auditu, hodnocení stavu zabezpečení IS a přehled nalezených nedostatků včetně jejich možného negativního dopadu pro společnost a návrh opatření k jejich odstranění.“* (MLÝNEK, J. 2007)

[55] Jako je například finanční či operační audit.

Vidina úspěšného IT auditu by mohla sloužit jako motivační faktor kladoucí důraz na důsledné dodržování softwarového procesu a všech jeho aspektů. IT audit může vytvářet pomyslný tlak na zadavatele projektu i na vývojový tým, díky čemuž může vzniknout snaha o kvalitní a kyberneticky bezpečný vývoj SW, kterým si lze dopomoci použitím vhodně zvolených nástrojů integrovaných do vývojového rozhraní. Integrace kybernetického zabezpečení přímo do vývoje SW nástrojů by mohla být cesta k úspěšnému dřívějšímu odhalení rizik a následnému ošetření všech možných zranitelností. K této integraci lze využít již řadu dostupných DevSecOps nástrojů[56], jejichž základním úkolem je včasné informování vývojářů o potenciálních hrozbách, díky čemuž lze předejít značným problémům. *„Pro DevSecOps je ústředním principem řešit zabezpečení již od počátku vývojového procesu, a nikoli, jak to dříve bývalo, až po vytvoření aplikace, nebo ještě hůře až po zavedení do produkčního prostředí.“* (HULME, G. V., MARTIN, J. A. 2022) Mnoho let bylo zvykem, že čas hrál pro firmy klíčovou roli. Z tohoto důvodu se uchylovalo k posuzování, hodnocení a následnému výběru vývojového týmu dle rychlosti dodání spustitelného zdrojového kódu i následnému finálnímu předání SW projektu. Na bezpečnost dodávaného SW se nekladl žádný důraz. Mnohdy se ona bezpečnost řešila až po dodání SW projektu a jeho integraci v rámci podnikového prostředí či se odpovědnost za případné kybernetické zranitelnosti přesouvala na jiné, většinou na zadavatele projektu. V průběhu let docházelo k řadě méně významným i velmi zásadním útokům, ovšem za nejznámější lze považovat případ SolarWinds[57]. Od tohoto incidentu se začal brát zřetel na důkladné posuzování a certifikování bezpečnosti zdrojového kódu. První zmínka o odpovědnosti vývojářů za nasazení bezpečného SW řešení do procesu byla publikována v americké vládní

[56] Nástroje lze rozdělit do několika kategorií. Za základní kategorii lze považovat nástroje, které plní informační funkci (patří mezi ně Pagerduty, xMatters, Alerta, ElastAlert). Druhým stupněm jsou nástroje, které dokážou kontrolovat bezpečnost zdrojového kódu a v případě výskytu zranitelností jej dokážou odstranit přímo v integrovaném vývojářském prostředí. Jedná se například od Checkmarx SAST, Burp Suite Enterprise Edition či platforma Verocode. Za další vylepšení nástrojů lze považovat grafické vizualizace bezpečnostních informací, automatizace bezpečnostních testů v každé fázi vývoje SW, tvorbu simulovaných útoků. Mezi tyto vylepšené nástroje lze zařadit Chef InSpec, Gauntlt, RedHat Ansible Automation či GitLab.

[57] V roce 2022 došlo k bezprecedentnímu útoku na více než osmnáct tisíc předních světových společností a vládních úřadů, kdy hackeři vložili kód (backdoor) do legitimní knihovny, čímž si zajistili vzdálený přístup do infrastruktury všech uživatelů používající a pravidelně aktualizující software SolarWinds Orion. Mezi zasažené subjekty patřily Microsoft, Siemens, Federální rezervní fond (FED), tajné služby, vládní administrativa USA či celá řada letišť, nemocnic.

S nástupem umělé inteligence přichází i hrozba v podobě HackAI, která bude schopna porozumět návrhu bezpečnostních systémů v podniku a vymyslet způsob, jak jej obejít.

směrnici. Postupně se klauzule odpovědnosti začala prosazovat i u řady jiných podnikatelských, neziskových a veřejnoprávních subjektů.

S nárůstem důrazu na bezpečnost zdrojových kódů se dostaly do popředí nástroje umožňující staticky[58] i dynamicky[59] testovat zabezpečení SW řešení. *„Oba typy nástrojů mohou pomoci vývojářům s nasazováním bezpečného kódu, ať již v rámci oficiálního procesu DevSecOps nebo prostřednictvím přesunu větší části odpovědnosti za zabezpečení blíže k místu vytváření aplikací.“* (BREEDEN, J. II. 2022) Statické nástroje SAST jsou používány během vývojového procesu, čímž lze zajistit, aby nově napsaný zdrojový kód neobsahoval neúmyslné zranitelnosti či jinak nepoškozoval již odladěný funkční software. Některé nástroje SAST jsou součástí integrovaného vývojového prostředí (IDE), což lze považovat za ideální způsob vývoje bezpečného kódu v reálném čase. Vývojář je během psaní zdrojového kódu upozorněn na bezpečnostní chyby, díky čemuž jej může ihned opravovat, aniž by se k chybám musel někdy v budoucnu vracet. Dynamické nástroje DAST jsou používány po dokončení zdrojového kódu jako vnější tester a scanner, který ověřuje odolnost a hledá zranitelnosti typických pro nejčastější kybernetické útoky.

V posledních letech se stále častěji uplatňuje vývoj SW řešení uvnitř organizací s využitím interních zaměstnanců, u kterých je předpoklad podnikové expertízy, znalosti problému a jeho požadovaného řešení. Ve většině případů se jedná o podnikové analytiky či řadové zaměstnance pracující s danou problematikou. Těmto neprogramátorům trh nabízí nástroje a frameworky[60] pracující na bázi low-code či no-code, které umožňují vytvářet SW nástroje a aplikace či jejich uživatelské rozhraní uchopením a přesunem komponent funkcionalit bez psaní zdrojových kódů.

Výrazný posun vývoje SW může přijít s využitím umělé inteligence v roli programátora, která bude schopna psát zdrojové kódy ve zlomku sekund a učit se

[58] SAST (*Static Application Security Testing*) je nástroj pro statické testování zabezpečení zdrojových kódů SW. Na trhu jsou dostupné následující SAST nástroje: Checkmarx SAST, CyberREs Fortify, Veracode Static Analysis SAST.

[59] DAST (Dynamic Application Security Testing) je nástroj pro dynamické testování zabezpečení zdrojových kódů SW. Za výborné DAST řešení lze považovat Acunetix DAST, Synopsys Managed DAST, Tenable.io Web App Scanning.

[60] Na Trhu jsou dostupné platformy FileMaker, Microsoft PowerApps, Google App Maker.

z již vytvořených SW řešení. Dalo by se říci, že uživatel naznačí svůj problém, který by chtěl pomocí SW nástroje řešit a umělá inteligence[61] mu zanalyzuje a nasimuluje komplexní technické řešení a za použití různých programovacích jazyků jej zhmotní do funkční softwarové podoby. Člověk jako programátor bude mít v softwarovém procesu funkci kontrolní, kdy bude dohlížet nad průběhem vývoje SW a testovat, jestli výsledný SW splňuje stanovený cíl a požadavky kladené na jeho fungování. Je možné, že díky umělé inteligenci se celý životní cyklus vývoje SW značně zkrátí a zefektivní. Je pravděpodobné, že výsledný SW bude bez sémantických chyb a překlepů, svižný, efektivní. Otázkou ale je, jestli softwarový proces a výsledný SW bude bezpečný a jestli budou moci uživatelé a programátoři umělé inteligenci opravdu důvěřovat? Před rizikem umělých dělníků varoval v roce 1920 i Karel Čapek ve svém utopickém díle R.U.R., ze kterého lze čerpat prvky rizik a zakomponovat jej do současného trendu vývoje umělé inteligence a informačních technologií. Příkladem může být situace, kdy by si dvě umělé inteligence vytvořily svůj, pro člověka nesrozumitelný, tajný unikátní jazyk, ve kterém by spolu komunikovaly a který by používaly i pro vývoj SW[62]. Pro člověka by se taková komunikace s umělou inteligencí stala doslova nemožnou a nemožná by byla i kontrola správnosti výsledného softwaru. Je třeba si uvědomit, že umělá inteligence funguje pomocí algoritmu tak, jak byla stvořená a naprogramovaná člověkem. Z tohoto faktu vyplývá i další riziko hackerského ovládnutí umělé inteligence a využití jej ve prospěch hackera či zájmových skupin. Například pro vytvoření škodlivých softwarů[63], šifrovacích nástrojů či k ovládnutí, umělou inteligencí vytvořených, softwarových nástrojů a databázových uložišť. Je otázkou,

[61] Pravděpodobně by se jednalo o obecnou umělou inteligenci (AGI), která je založena na strojové inteligenci a systému umožňujícímu nabývat vlastního vědomí, vnímání a automatickému učení.

[62] Pro představu lze použít vhodnou asociaci termínů zdrojového a objektového kódu. Zdrojový kód je pro programátora čitelný a srozumitelný, lze ho zkoumat, upravovat a rozvíjet na základě nových myšlenek. *„Naproti tomu objektový kód už prošel takovou proměnou, že v něm obvykle nelze odkrýt nic aspoň vzdáleně podobného původnímu zdrojovému kódu, získat z něj vodítko pro tvorbu jeho variant, a často se z něj ani nedá pochopit, jak funguje.“* (KERNIGHAN, W. B. 2019)

[63] Příkladem je událost z prosince 2022, kdy hackeři využili AI jazykový model ChatGPT (z dílny neziskové organizace OpenAI) k vytvoření škodlivých malwarů a šifrovacích nástrojů. Tento škodlivý malware je v kyberprostoru považován za méně rizikový, neboť byl vytvořen k vyhledávání běžných souborů (dokumentů ve formátu .doc a PDF či obrázků) a k jejich kopírování do adresáře sloužícího k následnému nešifrovanému odeslání přes webové rozhraní. V lednu 2023 obdobně stvořený malware vyřadil 1 000 norským plavidlům připojení k pobřežním serverům. I přes to je důležité považovat tyto incidenty za klíčový precedent, neboť je pravděpodobné, že v budoucnu budou vznikat sofistikované i velmi rizikové kybernetické hrozby stvořené umělou inteligencí.

kdo by v takovém případě nesl odpovědnost za chování a bezpečnost AI programátora? Bezpochyby by člověk jako tvůrce měl v počáteční fázi vývoje AI programátora myslet na bezpečnost technologie a v neposlední řadě i na způsob, jak onu technologii vyřadit z provozu dříve, než napáchá nedozírné škody. Z tohoto důvodu je nutné důsledně zvážit, jestli vznik oné technologie posune společnost kupředu v prosperitě, efektivitě, bezpečnosti s maximálním možným využitím nejmodernějších technologií 21. století na straně jedné, a adekvátní cenou za hrozby z ní vyplývající na straně druhé.

1.5 Umění kvality softwaru

Alan Mark Davis (1993) či Dean Leffingwell (1997) ve svých dílech již v 90. letech 20. století poukazovali na chyby způsobené během softwarového procesu. Uváděli, že chyby, nedostatečné porozumění a definování potřeb či jejich nedůsledná dokumentace již během analýzy požadavků mohou za 40–60 procent všech neúspěšných či špatně fungujících SW projektů. *„Kvalitní software by měl uživateli poskytovat požadované funkce a výkon a měl by být spravovatelný, spolehlivý a snadno použitelný.“* (SOMMERVILLE, I. 2013)

Jak uvádí Robert Cecil Martin (2009) ve svém významném díle Čistý kód, dle grafiky „WTFs/m“[64] Thomase Holwerda je jediná skutečná míra kvality kódu: počet průšvihů za minutu. Thomas Holwerdy na jednoduché ilustraci dvou dveří představujících kontrolní testování zdrojového kódu poukazuje na přímou úměru: Čím méně průšvihů zdrojový kód vygeneruje, tím je kvalitnější.

Kvalitu softwaru lze vyjádřit i čistotou kódu. Čistý kód je charakterizován mnoha významnými programátory[65] jako podmanivý, jednoduchý, kultivovaně psaný, čitelný, důmyslný, přímočarý, elegantní, účinný, dobře funkční s optimálním výkonem. *„Stručně řečeno, programátor píšící čistý kód je umělcem, který začne s čistým plátnem a během série transformací skončí u systému s elegantním kódem.“* (MARTIN, R. C. 2009) Dave Thomas spojuje čistotu zdrojového kódu

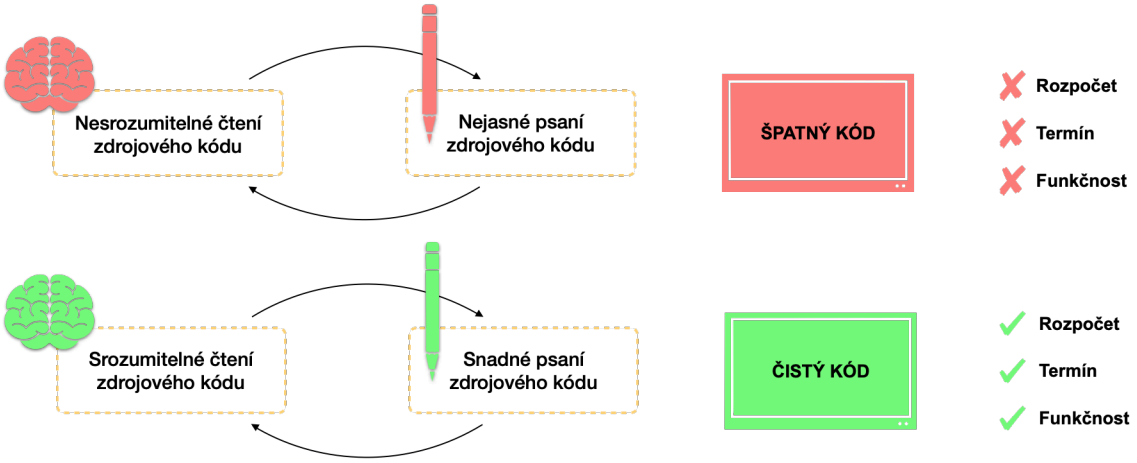
[64] Ilustrace je v originále dostupná na následující adrese: https://www.osnews.com/story/19266/wtfsm/

[65] Například Bjarnem Stroustrupem (tvůrce jazyka C++), Gradym Boochem (autor knihy Object Oriented Analysis and Design with Applications), Michaelem Feathersem (autor knihy Working Effectively with Legacy Code), Davem Thomasem (zakladatel OTI, kmotr strategie Eclipse), Ronem Jeffriesem (autor knihy Extreme Programming Installed, nebo Extreme Programming Adventures in C#) či Wardem Cunninghamem (tvůrce Wiki nebo spolutvůrce eXtrémního programování).

s testováním. To znamená, že zdrojový kód, který projde všemi zamýšlenými testy je čistý. Ron Jeffries se soustředí nejvíce na opakování prvků a částí zdrojového kódu. *„Pokud se něco neustále opakuje, je to známka, že máme v hlavě myšlenku, která není v kódu dobře vyjádřena.“* (MARTIN, R. C. 2009) Je tedy nutné, aby vývojář ve spolupráci s analytikem prověřil, zda zcela a správně porozuměl zadání a zda chápe podstatu požadavku zadavatele. Ward Cunningham tvrdí: *„Kód můžete označit za nádherný, když vypadá, jako by byl použitý jazyk pro daný problém stvořen.“* (MARTIN, R. C. 2009)

Pohledů na čistotu kódu je celá řada. Objektivní význam je ale jednoznačný. Čistý kód je radostné číst a následně i rozvíjet. Zároveň lze předpokládat vysokou míru pravděpodobnosti dodržení plánovaného finančního rozpočtu, termínu dokončení i zajištění správné funkčnosti výsledného SW řešení.

Obr. 7 Rozdíl mezi špatným a čistým kódem



Zdroj: autorka

Předpokladem čistého zdrojového kódu jsou i komentáře před každou ucelenou částí kódu. Komentáře jsou považovány za klíčové nosiče užitečných informací mající za cíl informovat čtenáře kódu o použitých funkcionalitách, ale i o použitých vychytávkách, které autor zdrojového kódu vlastní tvůrčí činností vymyslel, a které by nemusely být pro cizího čtenáře jednoznačně srozumitelné.

Čistý zdrojový kód[66] již při jeho tvorbě neobsahuje funkční chyby, a proto je vysoká pravděpodobnost jeho včasného dokončení a předání funkčního SW řešení zadavateli.

[66] Příklady pravidel čistého zdrojového kódu jsou uvedeny v příloze 5.

Umění psaní čistého zdrojového kódu by bylo možné považovat za měkkou dovednost, kterou se mohou vývojáři naučit praxí. Pro vývojáře může tato dovednost představovat přidanou hodnotu na trhu práce, neboť čistý zdrojový kód je nejen cenově efektivní, ale představuje i profesionální přežití. To znamená, že u čistého zdrojového kódu nebudou muset být opravovány softwarové chyby.

Kvalitu softwaru lze posuzovat i dle stanovených externích atributů kvality softwaru, mezi které patří *„bezchybnost, použitelnost, efektivita, spolehlivost, integrita, přizpůsobivost, přesnost, robustnost, udržovatelnost, flexibilita, přenositelnost, znovupoužitelnost, čitelnost, testovatelnost a srozumitelnost“*. (MCCONNELL, S. 2005) Vývojový tým by měl dojít nejen k dokonalé rovnováze mezi zmíněnými externími atributy, ale i k dokonalé rovnováze mezi interními[67] a externími atributy. Je ale důležité si uvědomit, že nelze maximalizovat jeden atribut na úkor atributu druhého. Pokud by například vývojový tým maximalizoval robustnost, jistě by posílil i rozsah použitelnosti a přizpůsobitelnosti, ale rozhodně by uškodil bezchybnosti, efektivnosti, spolehlivosti, integritě a přesnosti. Jako příklad lze uvést informační systém, který by byl vytvořen pro všechny subjekty bez ohledu na jejich formu podnikání, velikost či oborovou působnost. Pokud by byla hlavním cílem efektivnost, mohlo by dojít k potlačení bezchybnosti, spolehlivosti, integritě, přizpůsobivosti a přesnosti. Zároveň je důležité zmínit, že nelze docílit rovnováhy externích atributů bez snahy o maximalizaci interního atributu, tzn. kvalitního vývoje SW. *„Software, jemuž vnitřně nerozumíte nebo jejž nejste schopni vnitřně udržovat či spravovat, oslabuje vaše možnosti opravy případných chyb, což zásadním způsobem ovlivňuje externí atributy bezchybnosti a spolehlivosti. Nepružný software nelze na základě požadavků uživatelů dále rozšiřovat, což ovlivňuje vnější atribut použitelnosti.“* (MCCONNELL, S. 2005) Z toho vyplývá, že některé atributy hrají významnou roli zejména pro uživatele, zatímco jiné znaky kvality SW usnadňují práci vývojovému týmu. *„Nalezení optimálního řešení na základě často protichůdných a vzájemně neslučitelných vlastností je aktivita, která z vývoje softwaru dělá skutečnou inženýrskou disciplínu.“* (MCCONNELL, S. 2005)

V rámci softwarového procesu je rovněž nutné zaměřit se i na techniky zlepšení kvality softwaru. Jedná se o plánované a systematické kroky sloužící k dosažení

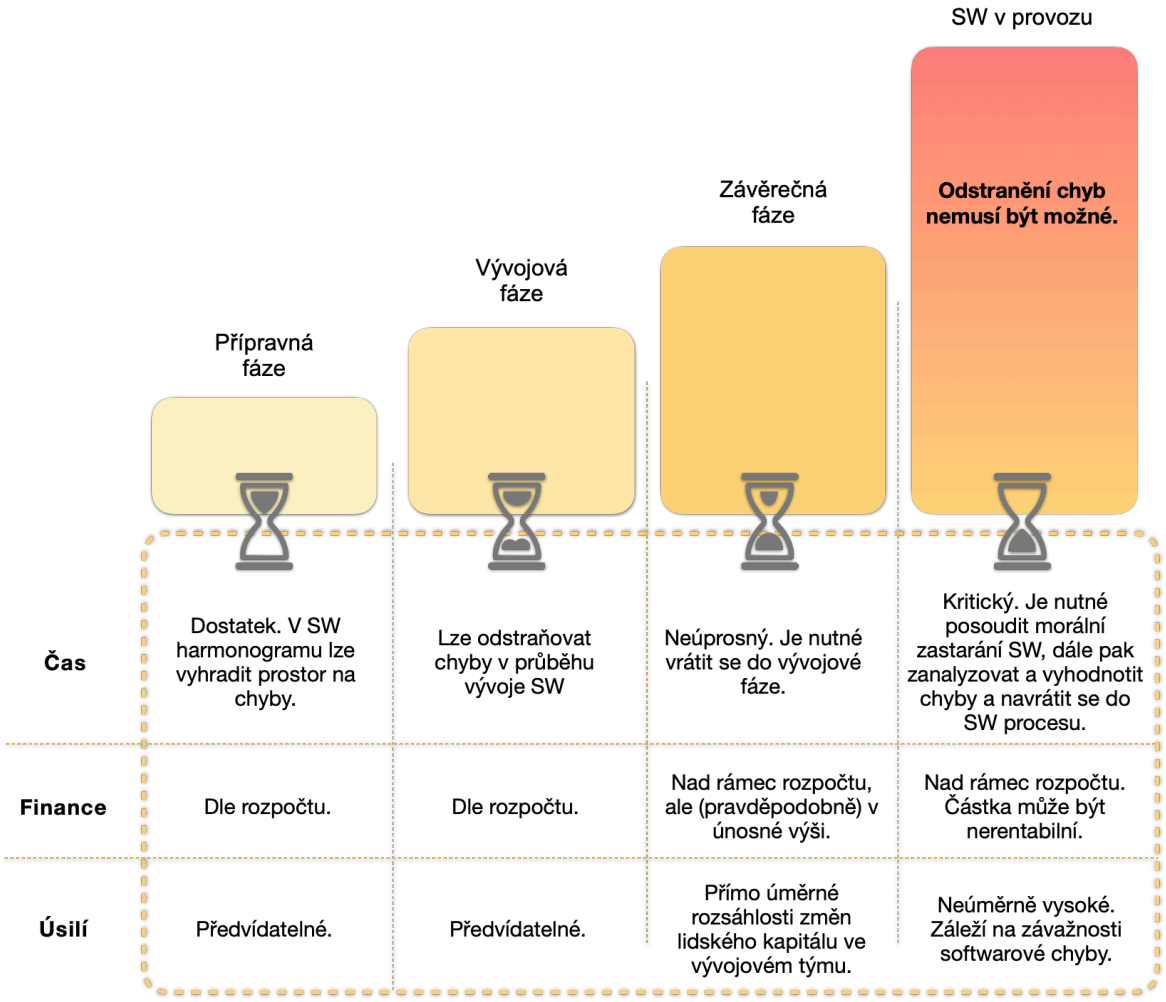
[67] Interními atributy jsou všechny kroky vedoucí k ideálnímu vývoji kvalitního softwaru. Jako příklad lze uvést disciplínu psaní čistého zdrojového kódu.

požadovaných vlastností SW. Za kontroly kvality se považují neformální a formální technické revize i externí IT audity. Neformální technické revize mohou vývojáři provádět pravidelně během vývoje SW. Zahrnují ruční kontrolu návrhu nebo zdrojového kódu, případně procházení dokumentace, kontrolních seznamů[68] a zdrojového kódu s dalšími spolupracovníky vývojového týmu. Formální technická revize tkví v periodicky prováděných testech a revizí, *„…jež určí, zda je kvalita produktu na takové úrovni, aby se proces vývoje produktu mohl přesunout do další fáze“.* (MCCONNELL, S. 2005) Formální technická revize určuje, zda jsou požadavky dostatečně splněny a zdrojový kód dostatečně funkční, aby se mohlo pokračovat ve vývoji SW. Specifikování metriky „dostatečné“ je plně v gesci vývojového týmu. Může se jednat například o splnění určitého procenta nejdůležitějších požadavků nebo určitých detailů SW nástroje.

Chyby, nedostatky a vady se do SW vkrádají ve všech fázích vývoje, proto by se vývojový tým měl soustředit na techniky zjišťování a odstraňování chyb po celou dobu SW procesu. Cenu za chyby lze vyjádřit součtem nákladů na nalezení chyb a nákladů na jejich odstranění. Podle Boehmova prvního zákona obecně platí, že čím dříve se chyba nalezne, tím méně bude její odstranění stát. Zároveň platí, že čím déle chyba v programu zůstane, tím více exponenciálně rostou náklady na její odstranění.

[68] Kontrolní seznamy obsahují soubor otázek a kontrolních bodů, které vývojový tým provází jednotlivými fázemi softwarového procesu. Jedná se například o kontrolní seznam požadavků (specifické funkční a nefunkční požadavky, kvalita a úplnost požadavků), kontrolní seznam hlavních postupů vývoje softwaru (kódování, zjištění kvality), kontrolní seznam programovacích nástrojů, kontrolní seznam technik tvorby dobrých komentářů, kontrolní seznam plánu zajištění kvality a jiné. Kontrolní seznamy by měly být obsaženy v dokumentaci.

Obr. 8 Cena za softwarovou chybu



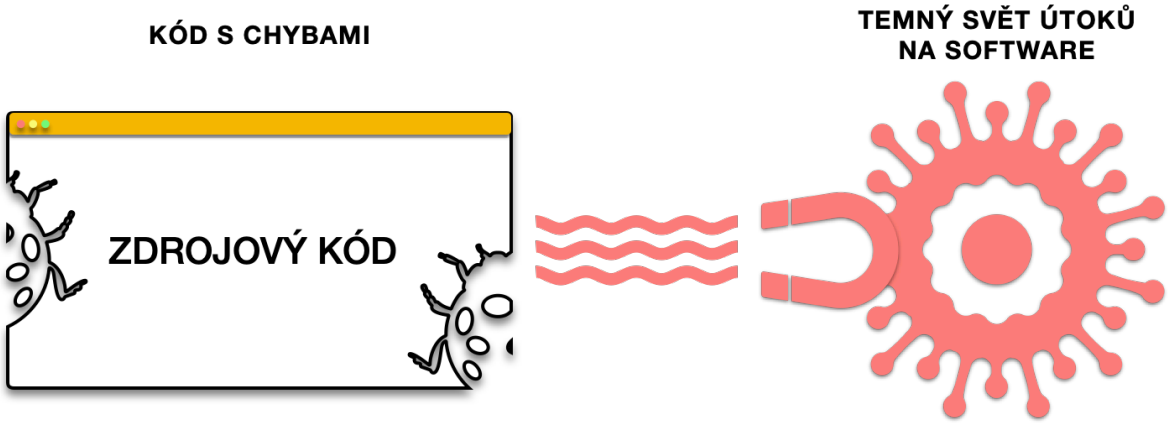
Zdroj: autorka

Pokud vývojový tým nastaví mezníky (brány jakosti), u kterých bude posuzovat kvalitu SW hotové části, předejde tak značným problémům. Obrázek 8 posuzuje cenu za softwarovou chybu z hlediska třech různých aspektů. Obecně lze říct, že čím více bude softwarový proces v pokročilé fázi, tím časově i finančně náročnější bude odstraňování chyb a tím pádem bude třeba i vyššího úsilí na dosažení kýženého cíle. Kritická situace může nastat ve chvíli, kdy bude SW řešení již v provozu. Vlivem morálního zastarávání softwaru nemusí být odstranění softwarových chyb možné. Z toho vyplývá, že čím dříve vývojový tým v softwarovém procesu zavede brány jakosti, tím s vyšší pravděpodobností dojde k brzkému odhalení chyb a tím pádem dojde i ke zvýšení kvality výsledného SW řešení. I přes *„notoricky známé prohlášení Barbary Ann Kitchenham: Kvalitu softwaru je těžké definovat, nemožné měřit a snadné rozpoznat.“* (ROUDENSKÝ, P. 2016) je důležité do rozpočtu softwarového projektu zahrnout i náklady na kvalitu. *„Právě vyjádření*

kvality prostřednictvím nákladů umožňuje jednotný pohled na její různé aspekty.“ (ROUDENSKÝ, P. 2016) Náklady na kvalitu včetně jejich rozpisu[69] se zabývá příloha 3.

Kvalitu softwaru bezpochyby zvyšuje i jeho bezpečnost[70], kterou je důležité pojmout již v počáteční fázi softwarového procesu například vzděláváním vývojového týmu a vytvořením modelu hrozeb. Stuart McClure[71] a kol. (2007) během své praxe vyvodil obecný poznatek, že za všemi bezpečnostními problémy jsou chyby[72] obsažené ve zdrojových kódech softwaru. Chyby, kterým lze předejít tvorbou bezpečného zdrojového kódu s cílem vyhnout se temnému světu útoků na software.

Obr. 9 Vliv chyb na kvalitu a bezpečnost zdrojového kódu



Zdroj: autorka

Příklady softwarových chyb, které mohou stát za příčinou kybernetického útoku, jsou popsány v příloze 4.

Michael Howard a David LeBlanc (2008) ve své knize Bezpečný kód tvrdí, že nelze vybudovat bezpečný systém, pokud vývojový tým nepochopí, jakým hrozbám je vystaven. Pokud vývojový tým již v rané fázi pojmenuje, kterým hrozbám může být SW vystaven a jakým způsobem je nutné kybernetické hrozby potlačit, zvýší tím

---

69 Rozpis nákladů na kvalitu může být vhodným podkladem pro oceňovatele.

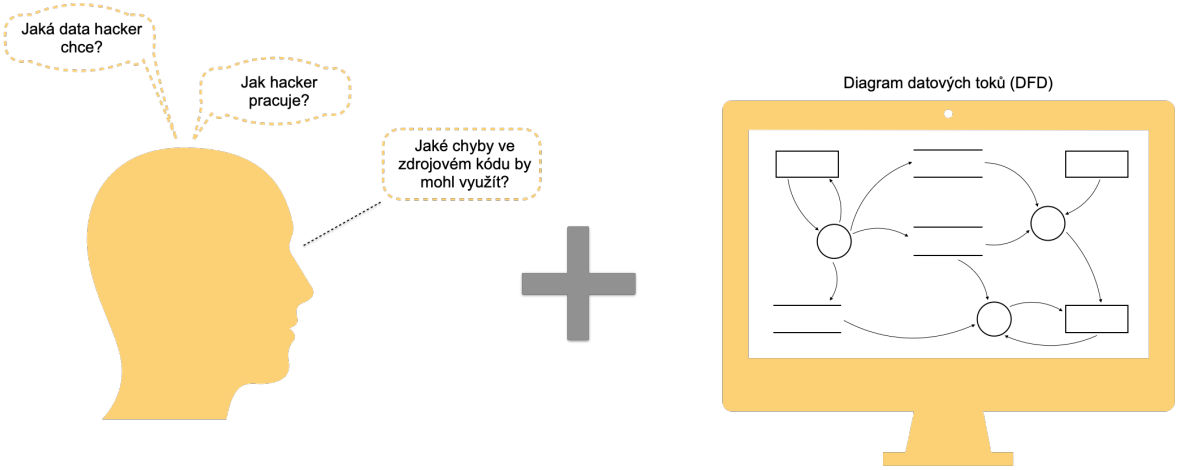
70 Bezpečnost softwaru nebo informačního systému lze definovat jako *„úroveň odolnosti informačního systému proti událostem, které mohou ohrozit důvěrnost, integritu nebo dostupnost zpracovávaných informací.“* (MLÝNEK, J. 2007)

71 Stuart McClure je považován za jednoho z předních představitelů kybernetické bezpečnosti. Pomáhá firmám zotavovat se z kybernetických útoků, čímž jim dokáže ušetřit čas i peníze.

72 *„Tyto „zkrátka chyby“ způsobí každý rok škody za miliardy dolarů. Zabezpečovací aplikace a služby se snaží zakrýt jejich důsledky, ale jediné skutečné řešení je zbavit se jich přímo na místě vzniku.“* (MCCLURE, S. a kol., 2007)

pravděpodobnost bezpečnosti kódu a tím pádem i kvalitu výsledného SW. Otázkou ale je, jakým způsobem vytvořit model hrozeb?

Obr. 10 Modelování hrozeb



Zdroj: autorka

Model hrozeb by šel definovat dvěma rovinami. První rovinou je subjektivní pochopení, jakým způsobem hacker myslí[73], jakého cíle chce svým útokem dosáhnout a kvůli které slabině by útok mohl úspěšně realizovat. Druhou, formálnější, rovinou lze docílit konkrétního vymezení hrozeb pomocí modelování datových toků nebo pomocí diagramu aktivit jazyka UML. Díky DFD lze znázornit tok dat, zdroje dat a jejich uživatele. Díky této vizualizaci lze určit hrozby, kterým je software vystaven, přiřadit jim hodnotu rizika a následně určit techniky, kterými dojde k potlačení hrozeb. Peter Szor[74] (2006) ve svém díle Počítačové viry: analýza útoku a obrana uvádí, že analýza hrozeb obsahuje obvyklé vzorce, které se může programátor snadno naučit a které vedou k výrazné efektivitě při modelování hrozeb. Základem úspěšného modelování hrozeb[75] může být neustálé

[73] Dovednost myšlení hackera si lze osvojit, například hrou Války o jádro: bojující programy. Hru kolem roku 1966 navrhl Robert Morris St. se svými přáteli pro matematiky, programátory a také hackery. *„Cílem hry je zabít programy oponenta jejich přepsáním.“* (SZOR, P. 2006) Jazyk Redcode, ve kterém byla hra naprogramovaná, umožňoval soupeřícím tvorbu záludných programů a tím pádem i různorodý sofistikovaný a tvůrčí boj.

[74] Peter Szor (1970 - 2013) byl významný maďarský antivirový výzkumník a autor antivirového programu Pasteur. K této disciplíně se dostal v září roku 1990, kdy jeho nový počítač napadl prostřednictvím infikované diskety vir „Stoned“ pocházející z Nového Zélandu. Virem Stoned byly infikovány počítače například i v laboratořích. Szor tehdy dokázal díky studiu knihy Petera Nortona (Programmer’s Guide to the IBM PC) analyzovat hrozbu a následně v Turbo Pascalu vytvořit program na detekci a odstranění viru.

[75] K bezpečnému a efektivnímu analyzování hrozeb mohou vývojáři použít celou řadu pomůcek, jako například disassemblery, debuggery, emulátory, testovací sítě pro viry, dekódovací nástroje, virtuální stroje, systémy určené k šíření virů.

vzdělávání a zdokonalování dovedností vývojářů, díky čemuž se mohou přiblížit k dokonalému *„porozumění virových zvyků v dostatečném předstihu, umožňujícím prevenci a zároveň odezvu zajišťující zvládnutí situace v případě, kdy daná hrozba propukne.“* (SZOR, P. 2006) Podle knihy Zranitelný kód, autora Sverre H. Husebyho (2006), lze za nedílnou součást modelování hrozeb považovat důsledné zaznamenávání[76] veškerých postupů, klíčových myšlenek a výsledků testů nejen během vývoje softwaru, ale i po dobu jeho provozu v prostředí podniku.

Již na počátku 21. století uváděl James C. Foster ve své knize Buffer Overflow, že *„více než 70 % útoků proti firemní síti se odehrává na „aplikační vrstvě“, nikoliv na síťové či systémové vrstvě.“* V době, kdy se do popředí dostává umělá inteligence se schopností vytvořit nepředstavitelné množství sofistikovaných škodlivých kódů v řádech minut, se snahy vývojářů o neustálé[77] zdokonalování se ve vývoji bezpečných zdrojových kódů mohou zdát marné. Nicméně důsledné zaznamenávání všech poznatků a následné sdílení svých analýz prostřednictvím internetu, odborných publikací a článků lze vytvořit silnou informační síť[78], ve které se vývojáři mohou učit nejen z vlastních chyb, ale i z chyb cizích. Z tohoto důvodu by reálným a splnitelným cílem vývojářů měla být snaha o vytvoření zdrojového kódu bez slabých míst při použití svých znalostí a všech dostupných prostředků a nástrojů.[79]

S kvalitou SW souvisí i kvalita systému jako celku[80], který má svoji podstatu, svůj účel a cíl. Není výjimkou, že jednotlivá softwarová řešení jsou v podniku sdružována do podnikového informačního systému[81], kde dochází k jejich vzájemné interakci. Z tohoto důvodu je nesmírně důležité, aby jednotlivá softwarová řešení dosahovala

---

[76] Nevyčíslitelná hodnota záznamů může vzniknout například při personálních změnách nebo během softwarového procesu jako podklad pro zdokonalování bezpečnosti SW. *„Záznamy jsou cenné i tehdy, když se podaří útočníka odhalit a je proti němu zahájeno soudní řízení.“* (HUSEBY, S. H. 2006)

[77] Podle Stuarta McClura (2007) se každých tři až pět let objeví zcela nová hackerská technika.

[78] Je známo, že v době globální informační společnosti jsou za nejcennější zdroje považovány znalosti a informace.

[79] Na druhou stranu, Murphyho zákon říká: *„Bezchybný program je jako kvadratura kruhu. Člověk si myslí, že je to možné, ale nikomu se to nikdy nepovedlo.“* (GRAF, J. 1998)

[80] *„Mnohé složité věci jsou jako celek více než jen souhrn částí, ze kterých se skládají [ARISTOTELES].“* (BRUCKNER, T. a kol. 2012) Pro takový soubor složitých věci se ustálil výraz systém.

[81] *„Úkolem informačního systému je zajištění správných informací na správném místě ve správný čas.“* (BRUCKNER, T. a kol. 2012) Ona správnost může být zjistitelným ukazatelem kvality SW.

nejvyšší možné kvality. Dalo by se obecně říct, že celek podnikového ICT dosahuje takové kvality, jaké kvality dosahuje jeho nejslabší část SW řešení.

2 Oceňování softwarových projektů

Oceňování aktiv lze považovat za dynamickou a stále rozvíjející se disciplínu, která má v České republice mezi oceňovateli poněkud nejasný výklad a s tím související vrtkavé praktické využití. Od devadesátých let se v odborné praxi ustálil nosný pojem oceňování, který *„zcela přesně neodpovídá dlouhodobě ustálené zahraniční praxi“* (KRABEC, T. 2009) a pravděpodobně neodpovídá ani podstatě oné práce související s ohodnocením daného aktiva. *„Zatímco „oceňování“ primárně souvisí se stanovením ceny – tj. nikoliv odhadem hodnoty – pro daný, konkrétní účel, například pro určení daňové povinnosti, v drtivé většině případů toto ocenění není potřeba a není ani na místě.“* (KRABEC, T. 2009) V tomto kontextu by bylo užití pojmu ohodnocování aktiva vhodnější, neboť se v praxi jedná o stanovení odhadnuté peněžní částky[82] pro tržní transakce mezi soukromoprávními subjekty. Pro ohodnocení jakéhokoliv aktiva, nejenom nehmotného, je klíčové porozumět terminologii, obecně platným definicím vycházejících z obecně uznávaných standardů a souvisejících předpisů.

„Klíčovou výzvou pro oceňovatele nehmotných aktiv je správné porozumění předmětu, jehož hodnotu má stanovit, zejména obsahu práv, která jsou s daným nehmotným předmětem spojena a která mohou být zdrojem rizika dosažení přiměřeného výsledku ocenění.“ (SVAČINA, P. 2010) Z tohoto důvodu je nezbytně nutné vymezit související pojmy, všechny možné okolnosti a nejistoty, které by mohly mít zásadní vliv na ohodnocování softwarového projektu v dynamickém IT prostředí.

Softwarový projekt je z účetního hlediska považován za nehmotné aktivum podniku. České účetní předpisy i Mezinárodní standardy účetního výkaznictví[83] jej definují jako nepeněžní aktivum nehmotné povahy, které je výsledkem minulých událostí, lze jej snadno identifikovat, kontrolovat, spolehlivě ocenit a u nějž je pravděpodobnost budoucího ekonomického prospěchu. *„Důraz na definici nehmotného aktiva není rozhodně samoúčelný, neboť není-li splněna některá z uvedených podmínek, může to znamenat, že předmět není samostatně ocenitelný, byť se záležitost může na první pohled jevit odlišně.“* (SVAČINA, P.

[82] Nebo částky vyjádřené peněžními ekvivalenty.

[83] Definice podle standardu IAS 38 – *Nehmotná aktiva*.

2010) Pokud by soudní znalec ocenil aktivum, které není samostatně ocenitelné a převoditelné, ovlivnil by tak hodnotu celkových aktiv, umožnil by společnosti ovlivňovat hospodářský výsledek formou účetních a daňových odpisů, což by mělo dopad i na zdanění a výplatu podílů na zisku. Jako příklad nesplnění testu klasifikace nehmotného aktiva lze uvést firemní know-how, který vytvořili zaměstnanci podniku na základě svých znalostí bez náležitého zhotovení dokumentace. Pokud k podnikovému know-how chybí dokumentace, nelze takové aktivum považovat za dostatečně identifikovatelné. Pokud by zaměstnanci ukončili pracovní poměr, není aktivum dále kontrolovatelné.

Softwarový projekt spadá i do oblasti duševního vlastnictví (*Intellectual Property*) v rámci systému autorského práva[84] (*Copyright*). Smyslem vzniku a existence práv duševního vlastnictví je vymezení vlastnických práv k nehmotným aktivům. Autorské právo chrání formu nehmotného aktiva k okamžiku jeho ztvárnění do objektivně vnímatelné podoby. V případě softwarového projektu se jedná o nosič dat, na kterém je umístěn zdrojový kód softwarového projektu. Softwarový projekt je chráněn celosvětově bez ohledu na hranice států. Autoři softwarového projektu jsou majiteli práv po dobu svého života a následně 70 let po jejich úmrtí. Další možnou ochranou spadající do oblasti duševního vlastnictví by mohla být patentová ochrana[85], jejíž předmětem jsou výsledky technické tvůrčí činnosti splňující kritéria patentovatelnosti[86]. Zde ovšem software naráží na ustanovení 3, odst. 2, písm. c,[87] zákona o vynálezech, průmyslových vzorech a zlepšovacích návrzích, které software (počítačový program) za patent nepovažuje.

Pro podnik mohou mít nehmotná aktiva značný význam[88], neboť mohou výrazně ovlivnit hodnotu vykázaných aktiv společnosti a tím i postavení a hodnotu firmy

---

[84] Autorskoprávní ochrana vychází z toho, *„že dílem ve smyslu autorského zákona je mimo jiné též počítačový program, a to v případě, že je původní – je tedy autorovým vlastním duševním výtvorem.“* (ČADA, K. 2014)

[85] *„V případě počítačových programů často nelze snadno oddělit myšlenku o vlastní podstatě řešení problému, kterou jistě není možné autorskoprávně chránit, od jejího vyjádření v určité vědecké či literární formě, ztvárněné do podoby zdrojového textu nebo ve strojně čitelné podobě do binárního kódu vlastního počítačového programu.“* (ČADA, K. 2014)

[86] Jedná se o novost, vynálezecká činnost a průmyslová využitelnost. Aby byl výsledek tvůrčí činnosti způsobilý průmyslověprávní ochrany, musejí být tyto tři podmínky splněny.

[87] *„Za vynálezy se nepovažují plány, pravidla a způsoby vykonávání duševní činnosti, hraní her nebo vykonávání obchodní činnosti, jakož i programy počítačů.“* (§3, odst. 2, písm. c)

[88] Jako příklad lze uvést pohled na nehmotná aktiva Nicholase Negroponta (2001), řecko-amerického informatika, techno-optimisty a architekta, který ve svém díle Digitální svět uvedl:

na trhu. Softwarový projekt může podnik získat a v účetním výkaznictví vykázat následujícími způsoby (SVAČINA, P. 2010):

- koupí,
- nabytím práv k výsledkům duševního vlastnictví tvořivé činnosti (užívat v licenci),
- vytvořením vlastní činností,
- nepeněžitým vkladem jiného subjektu,
- převodem v rámci právních přeměn/fúzí,
- bezúplatným převodem (darováním), či převodem z osobního užívání do podnikání.

Zároveň existuje celá řada případů, kdy je nutné stanovit hodnotu již zavedeného softwarového projektu v podniku. Za takové případy lze považovat:

- nákup, prodej nebo licencování mezi spřízněnými i nespřízněnými stranami,
- pojištění kybernetických rizik s následným pojistným plněním[89],
- nepeněžitý vklad společníka do společnosti,
- likvidace a insolvence společnosti,
- přeměna[90] společnosti,
- vyčíslení škody z neoprávněného nakládání s nehmotným aktivem,

---

„Nedávno jsem navštívil ředitelství jednoho z největších amerických výrobců integrovaných obvodů. Požádali mě, abych se zapsal, a přitom se mě zeptali, zda u sebe nemám přenosný počítač. Pochopitelně jsem jej měl. Recepční si zapsala model a sériové číslo a pak se mne zeptala na přibližnou cenu přístroje. „Něco mezi jedním a dvěma miliony dolarů," řekl jsem. „To není možné, pane," odpověděla. „Jak jste to myslel? Mohu se na něj podívat?" Ukázal jsem jí svůj starý PowerBook a ona odhadla jeho cenu na dva tisíce dolarů. Zapsala si ji a pustila mě dovnitř. Pointa je jasná: atomy často ani moc velkou cenu nemají, ale bity mohou mít cenu obrovskou."

Zároveň dodává, že *„bit nemá barvu, rozměr ani váhu a může se pohybovat rychlostí světla. Jde o nejmenší složku, z níž je v přeneseném slova smyslu tvořena DNA každé informace."* (NEGROPONT, N. 2001) Z těchto a mnoha dalších důvodů mohou mít nehmotná aktiva pro podnik nevyčíslitelný význam, aniž by byl promítnut v účetním výkaznictví či v jinak dostupných informací.

[89] Podhodnocení SW projektu v pojistné smlouvě má vliv na vyplacení případného pojistného plnění.

[90] Zákon o přeměnách obchodních společností považuje za přeměnu fúze, dělení obchodních společností, převod jmění na společníka a změny právní formy společnosti.

- a spoustu jiných účetních a daňových účelů (např. ověření, zda nedošlo k nadhodnocení majetku a jiné).

Způsob ocenění závisí na způsobu nabytí softwarového projektu. Jako příklady lze uvést následující situace:

- Při koupi se použije pořizovací cena (pořizovací cena se skládá z ceny pořízení (cena za softwarový projekt) a vedlejších pořizovacích nákladů (náklady na zastoupení při zprostředkování obchodu)).
- Při vkladu se využije služeb soudního znalce[91], který zhotoví znalecký posudek dle závazných právních předpisů.
- Při vytvoření vlastní činností se využije způsobu ocenění vlastními náklady (přímé náklady vynaložené na vývoj softwarového projektu, a nepřímé náklady vztahující se k vývoji softwarového projektu).

2.1 Právní pozadí oceňování nehmotných aktiv

Oceňovatel svou prací při stanovování hodnoty nehmotných aktiv bezprostředně ovlivňuje účetní i daňový obraz podniku, a proto je klasifikace i následné stanovení hodnoty nehmotných aktiv zakotveno v řadě právních předpisů a standardů:

- Zákon č. 563/1991 Sb., o účetnictví.
- Vyhláška č. 500/2002 Sb., kterou se provádějí některá ustanovení zákona o účetnictví, pro účetní jednotky, které jsou podnikateli účtujícími v soustavě podvojného účetnictví.
- Český účetní standard pro podnikatele č. 013 – Dlouhodobý nehmotný a hmotný majetek.
- Zákon č. 586/1992 Sb., o daních z příjmů.
- IFRS účetní standard IAS 38 – Nehmotná aktiva.
- IFRS 3 – Podnikové kombinace.[92]

[91] V určitých případech lze využít i služeb obecně uznávaného odborníka. Tyto případy jsou specifikovány v § 59a, odst. 2 obchodního zákoníku.

[92] IFRS 3 zařazuje softwarový projekt (software i databáze) mezi technologie.

- Mezinárodní oceňovací standardy (IVS).[93]
- Evropské oceňovací standardy (EVS).

Na základě zákona o účetnictví, prováděcí vyhlášky, českých účetních standardů i zákona o daních z příjmů i IFRS předpisů oceňovatel odpovídá na otázky:

„Je posuzovaná věc zařaditelná mezi nehmotná aktiva a tím pádem ocenitelná jako nehmotné aktivum?“

„Za jakých podmínek lze vykázat posuzovanou věc v bilanci jako nehmotné aktivum?“

„Za jakých podmínek lze posuzovanou věc účetně i daňově odepisovat?“.

V rámci znalecké činnosti jsou Mezinárodní oceňovací standardy považovány za klíčový standardizační předpis. K ocenění nehmotných aktiv se vztahuje:

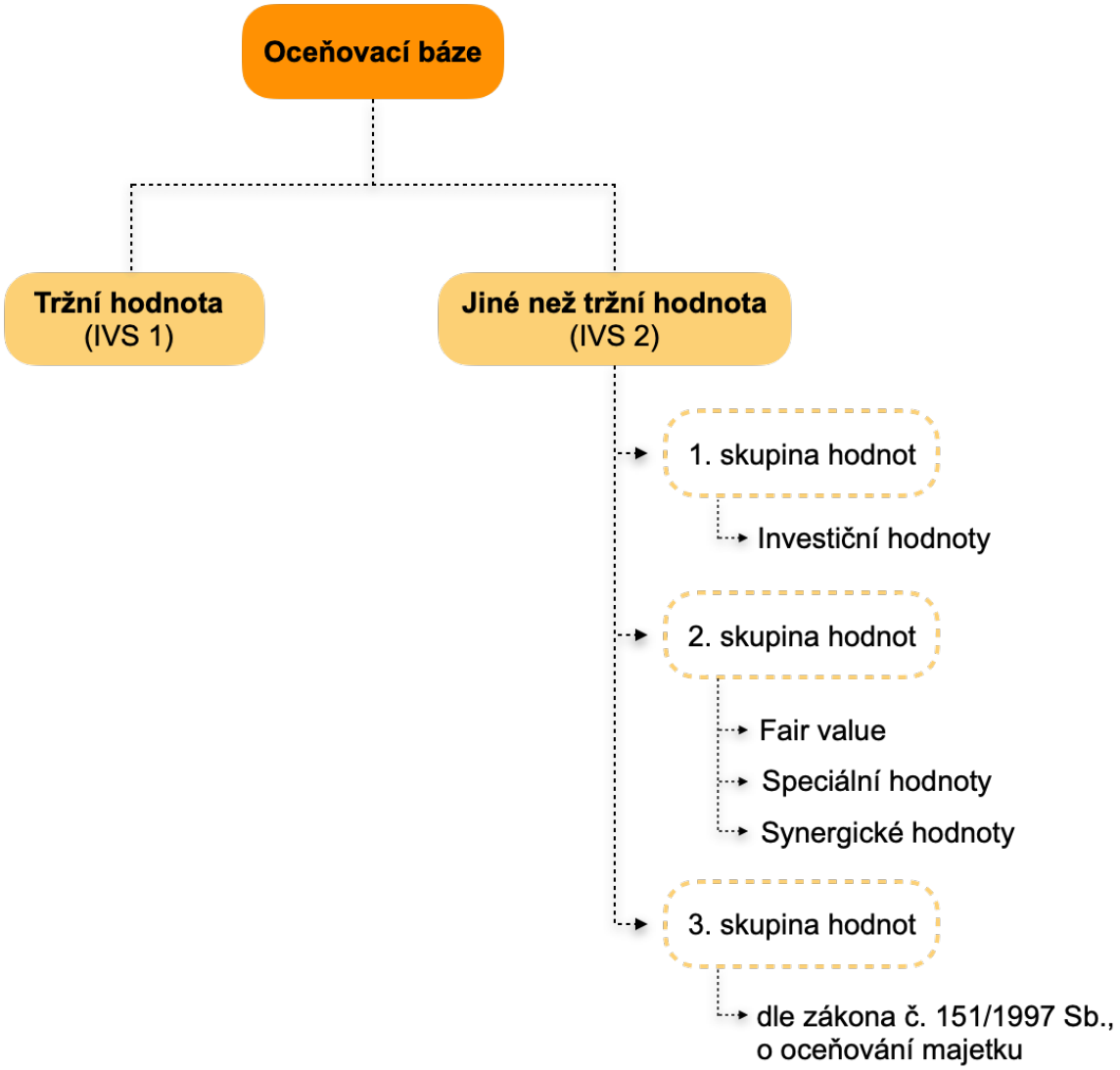
- Směrnice 4 – Oceňování nehmotných aktiv,
- Standard 1 – Oceňování tržní hodnotou,
- Standard 2 – Oceňování netržní hodnotou,
- Standard 3 – Oceňovací zpráva.

2.2 Východiska stanovení hodnoty nehmotných aktiv

Cílem práce oceňovatele je určit hodnotu posuzovaného aktiva na základě hlubší znalosti specifik daného typu nehmotného aktiva a jednoznačného určení zvolené oceňovací báze, které Mezinárodní oceňovací standardy (IVS) dělí do dvou kategorií.

[93] Tvůrcem Mezinárodních oceňovacích standardů je Výbor pro mezinárodní oceňovací standardy (IVSC, International Valuation Standards Council).

Obr. 11 Oceňovací báze dle IVS

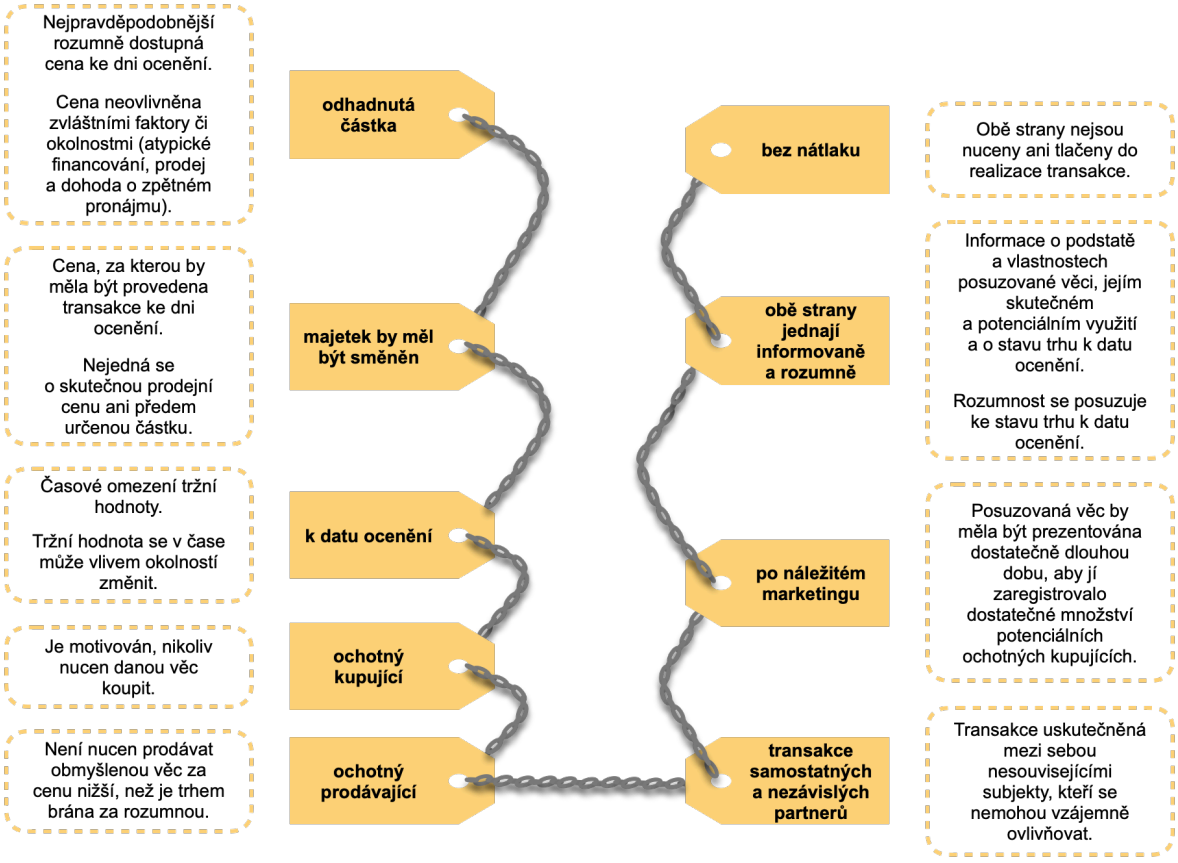


Zdroj: autorka

Tržní hodnota odpovídá na otázku: *„Kolik by za posuzovanou věc zaplatil průměrný poptávající na trhu?“*. Podle IVS 1 je tržní hodnota *„odhadnutá částka, za kterou by měl být majetek směněn k datu ocenění mezi ochotným kupujícím a ochotným prodávajícím při transakci mezi samostatnými a nezávislými partnery po náležitém marketingu, ve které by obě strany jednaly informovaně, rozumně a bez nátlaku.“* Je důležité podotknout, že tržní hodnota neodráží běžné ceny udávané trhem jen na základě nabídky a poptávky, neboť řada těchto transakcí se uskutečňuje pod nátlakem, mezi spřízněnými osobami či pod vlivem dalších okolností, které mohou hodnotu posuzovaného aktiva tržně zvyšovat. Oceňovatel tedy musí k posuzovanému aktivu přistupovat podle legislativně platných definic, které vymezují pomyslné mantinely, v rámci kterých by měl nalézt tvůrčí prostor

pro stanovení reálné tržní hodnoty. Jsou to právě definice a v nich obsažené závazné hypotézy, které zamezují oceňovatelům nadhodnocovat posuzovaná aktiva ve prospěch jedné strany.

Obr. 12 Řetězec hypotéz vyplývajících z definice tržní hodnoty dle IVS 1



Zdroj: autorka

Spolu s definicí tržní hodnoty uplatňují IVS také princip nejlepšího možného využití aktiva, který je definován jako *„nejpravděpodobnější použití majetku, které je fyzicky možné, odpovídajícím způsobem oprávněné, právně přípustné, finančně proveditelné a které má za následek nejvyšší hodnotu oceňovaného aktiva.“*

Tržní hodnota tedy poukazuje na bodový odhad, který splňuje všechny výše uvedené podmínky definice. V praxi ale existují aktiva, která nejsou natolik likvidní, aby bylo možné snadno určit pravděpodobnou rovnovážnou cenu, která vzniká střetnutím nabídky ochotného prodávajícího a poptávky ochotného kupujícího. Jedná se například o softwarové projekty, databáze a jiné ICT řešení, která jsou mnohdy dodávána podnikům na míru.

Druhou kategorií oceňovací báze jsou jiné než tržní kategorie, které vymezuje a podrobně definuje IVS 2 – Oceňování netržní hodnotou. Jedná se o oceňovací báze, které jsou založené na vnímání hodnoty aktiva pohledem konkrétního zájemce s konkrétními požadavky a v konkrétních podmínkách. Při oceňování nehmotných aktiv je pochopení všech specifik klíčové pro stanovení správné netržní hodnoty.

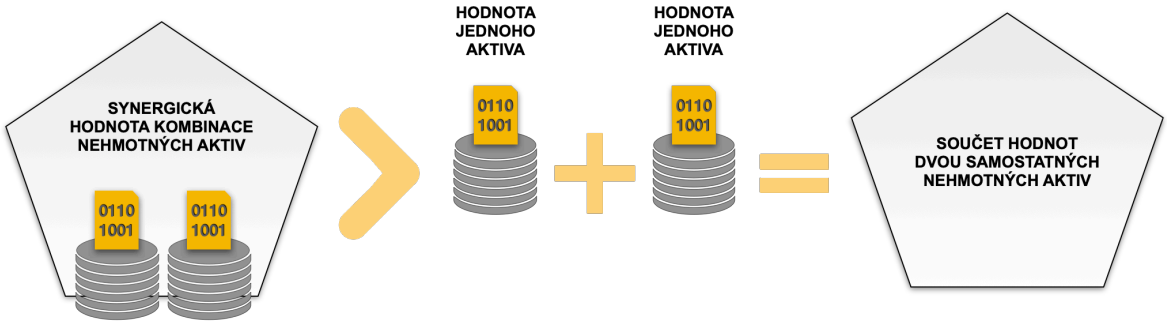
Investiční hodnota představuje hodnotu, která plyne z užívání nehmotného aktiva a zrcadlí hodnotu jeho přínosů pro společnost. Je vysoce pravděpodobné, že investiční hodnota bude vyšší než tržní.

Reálná hodnota dle definic *fair value* je považována za širší koncept, než je tomu u tržní hodnoty, a proto se využívá zejména při oceňování jedinečných nehmotných aktiv s cílem promítnout v oceněné hodnotě užitek. Jako příklad lze uvést akvizici a jiné podnikové kombinace dle IFRS 3[94], kdy se nehmotné aktivum k tomuto datu oceňuje hodnotou *fair value* s odkazem na aktivní trh.

Speciální hodnota odráží skutečné vlastnosti a užitky daného aktiva, které jsou cenné pro konkrétního speciálního zájemce. Pro jinou osobu by mohlo být takové aktivum nevýznamné, nulové hodnoty.

Synergická hodnota vzniká hodnotou užitků plynoucích z kombinací dvou a více nehmotných aktiv, přičemž společně tato nehmotná aktiva působí na oceňovanou hodnotu více než součet jednotlivých hodnot užitků.

Obr. 13 Synergická hodnota



Zdroj: autorka

[94] IFRS 3 je *Mezinárodní účetní standard* pojednávající o podnikových kombinacích a jejich důsledcích, o kterých musí účetní jednotky ve svém účetním výkaznictví sestaveném dle IFRS informovat.

Hodnotu synergie lze vyjádřit jako přírůstek hodnoty z kombinací nehmotných aktiv.

Vzorec 1 Hodnota synergie

$$Hodnota\ synergie = H_{ab} - (H_a + H_b)\,, \tag{1}$$

kde proměnná

H_{ab} hodnota celku,

H_a, H_b hodnota jednotlivých částí.

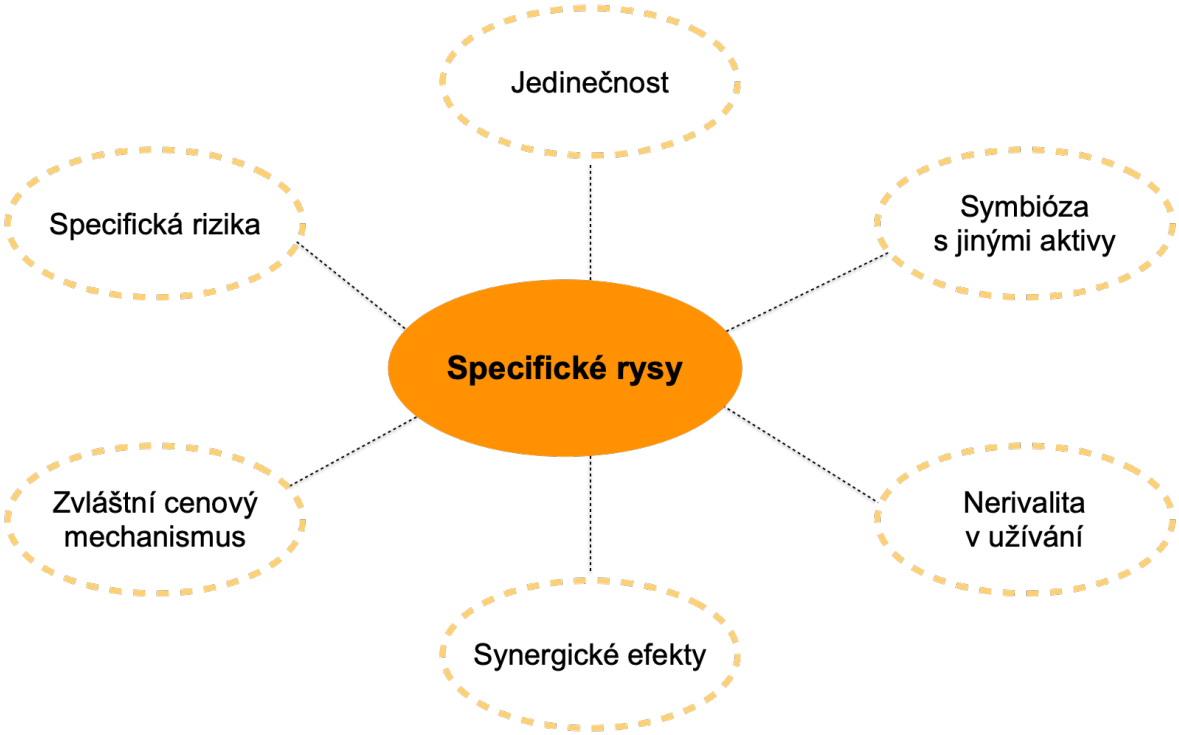
Jako modelový příklad lze uvést problematiku internetového e-shopu. Podnikající fyzická osoba se zabývá výrobou speciálních světel na kolo s následným prodejem koncovým zákazníkům přes internetový obchod, který funguje díky následujícím položkám: doménová adresa včetně webhostingu propagační webové stránky má hodnotu 2 000 Kč, textový obsah byl vyčíslen na 30 000 Kč, doménová adresa včetně webhostingu e-shopu má hodnotu 5 000 Kč, textový obsah e-shopu byl vyčíslen na 10 000 Kč. Databáze zákazníků a zboží má hodnotu 100 000 Kč. Celková hodnota e-shopu byla znalcem vyčíslena na 300 000 Kč. Výsledná hodnota synergie je 153 000 Kč[95].

Hodnota synergie při nesprávné kombinaci dvou a více nehmotných aktiv může nabývat záporných hodnot.

Nehmotná aktiva jsou tvořena lidskou myslí, a proto je vysoce pravděpodobné, že se na světě nenajdou dvě naprosto identická know-how, zdrojový kód softwaru, databáze, či soubor obchodních smluv. Úkolem oceňovatele je tedy nalézt cestu, jak jednotlivá specifika nehmotných aktiv zohlednit při ocenění a přiřadit jim váhu v rámci stanovení hodnoty posuzovaného předmětu tak, aby nedošlo k podhodnocení ani nadhodnocení ohodnocovaného aktiva.

[95] Hodnota synergie = 300 000 – (2 000 + 30 000 + 5 000 + 10 000 + 100 000)

Obr. 14 Specifika nehmotných aktiv

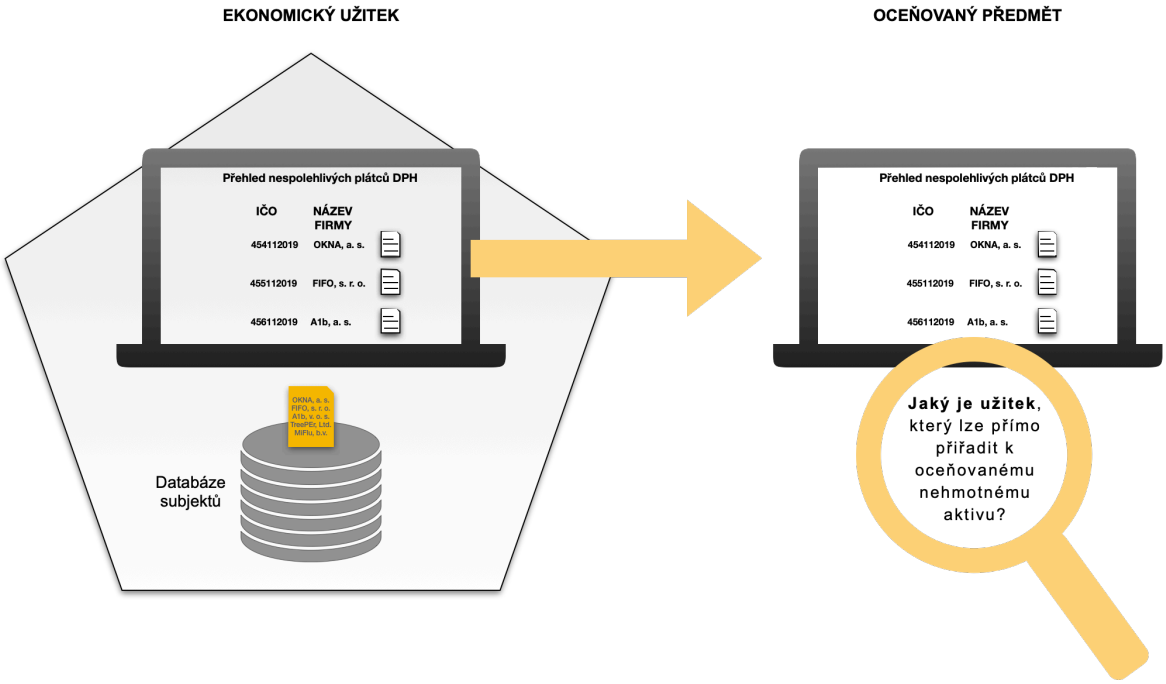


Zdroj: autorka

Jedinečnost lze vyjádřit jako novost, vynalézavost, inovativnost, rozlišovací způsobilost či individuální vyjádření určitého homogenního aktiva. Jedinečnost může být zhmotněna názvem, funkcionalitami, vizuální podobou, jednoduchostí a jinými charakteristikami.

Symbiózou s jinými aktivy se rozumí *„nutnost spojit nehmotné aktivum vždy s jinými aktivy, aby mohlo vydávat ekonomický užitek“.* (SVAČINA, P. 2010) Příkladem může být software, který hlídá plátcovství DPH. Bez napojení na interní odběratelsko-dodavatelskou databázi nemůže software zkoumat subjekty v řetězci a vyhodnocovat, který subjekt je pro podnik rizikem. SW by v takovém případě zobrazil všechny nespolehlivé plátce DPH v zemi, což jsou pro podnik irelevantní a naprosto zbytečné informace. Pro ocenění je nezbytně důležité, aby oceňovatel dokázal vyjádřit užitek plynoucí pouze ze zkoumaného nehmotného aktiva.

Obr. 15 Vynětí oceňovaného nehmotného aktiva z celku



Zdroj: autorka

Nerivalita v užívání představuje použití jednoho předmětu více subjekty za účelem maximalizace investičních příležitostí, aniž by došlo k rozdělení předmětu. Příkladem nerivality může být malá softwarová společnost, která vytvoří software a poskytne jej velké nadnárodní společnosti prostřednictvím licence. Tento kontrakt zajistí výnosy z rozsahu a zároveň mnohonásobně zhodnotí software. Dalším příkladem může být poskytnutí SW na prověřování nespolehlivých plátců DPH prostřednictvím licence dalším podnikatelským subjektům, kterým z jeho užívání budou plynout užitky. Majitel softwaru se práv k jeho užití nevzdává, pouze v transakci množí svůj kapitál při zanedbatelné oběti.

Synergické efekty vznikají spojováním aktiv, z čehož mohou plynout synergie v řádech stovek i tisíců procent[96]. Důvodem je nerivalita v užívání, čímž se zvyšují výnosy z rozsahu a zároveň s téměř nulovými transakčními náklady. Příkladem je software, jehož náklady na výzkum a vývoj byly vysoké, ovšem náklady na jeho licencování jsou spojeny pouze s transakcemi. *„V odborné praxi začíná převažovat názor, že pokud jsou synergie typickou vlastností aktiva a zároveň jsou dostupné*

[96] Na rozdíl od synergií hmotných aktiv, které se pohybují v řádech jednotek a desítek procent.

běžnému tržnímu zájemci, měly by se určitým způsobem do tržní hodnoty započítat.“ (BENEŠ, R. *Metodologie ocenění software*)

Zvláštní cenový mechanismus má *„významný vliv na přirozenou tvorbu tržní ceny nehmotného aktiva v cenovém mechanismu, což je právě proces, který simuluje oceňovatel svou prací“.* (SVAČINA, P. 2010) Při hledání hodnoty by se měl oceňovatel soustředit na přínosy na straně zájemce.

Specifická rizika[97] jsou rizika, která vznikají nad rámec těch běžných. U nehmotných aktiv se specifická rizika dělí do třech skupin:

- právní,
- tržní,
- technická.

Za právní rizika lze považovat patentové přihlášky bez náležitého průzkumu, obchodní tajemství, ztráty průmyslově-právní ochrany na základě žádostí třetích stran, napodobování v zemích s nízkou vymahatelností práv duševního vlastnictví. Za tržní rizika lze považovat komplikovaný odhad ekonomické využitelnosti a riziko rychlého morálního zastarání zejména u technických řešení. Za technická rizika lze považovat selhání v rané fázi výzkumu a vývoje softwarového projektu či nezdařilá implementace tohoto projektu v podnikovém procesu.

2.3 Metody oceňování

Přístupy a metody ocenění nehmotných aktiv se řadu let v různých variantách vyvíjí. I přes tuto skutečnost nebyl dosud předložen závazný postup pro stanovení hodnoty nehmotných aktiv, resp. softwarových projektů. Důvody vysokého počtu metod ocenění jsou následující (SVAČINA, P. 2010):

- každá metoda se svým vlastním způsobem snaží zachytit různorodost nehmotných aktiv a jejich specifik,
- metody jsou částečně inovovány o poznatky z nově se rozvíjejících trhů nehmotných aktiv.

[97] Rizika jsou investory vnímána v souvislosti se třemi stránkami – výnos, riziko a likvidita. Platí, že čím je výnos vyšší, tím vyšší je riziko nebo tím nižší je likvidita (schopnost směnit předmět na peníze).

Všechny metody oceňování jsou založeny na třech pilířích (SVAČINA, P. 2010):

- srovnávacím (*Market*);
- nákladovém (*Cost*);
- výnosovém (*Income*).

„Současná účetní praxe při určování hodnoty softwaru je založena na nákladech vynaložených na vývoj softwaru. Tento přístup nezohledňuje efektivitu, s jakou byl software vytvořen, ani kvalitu produktu.“[98] (J. de GROOT a kol. 2012) Skutečně vynaložené náklady nezohledňují veškeré přínosy, které SW bezpochyby subjektu přináší, a proto je důležité k ohodnocení SW projektu přistupovat komplexněji. Na druhou stranu jsou známy i četné problematické případy investic, kdy SW řešení nebylo realizováno v odpovídající kvalitě a negenerovalo očekávané přínosy. Tato skutečnost vyplývá nejen z podstaty nehmotného aktiva, jehož skutečnou hodnotu není možné jednoznačně sledovat a určit, ale je umocněna i komplikovaností procesu vývoje softwaru. Z těchto důvodu se může stát, že skutečná hodnota SW projektu může být podhodnocená, anebo nadhodnocená.

Srovnávací přístup

Srovnávací přístup je založen na myšlence srovnání dvou a více nehmotných aktiv konkurenčního trhu, které jsou srovnatelné svou hodnotou a užitečností. Tento oceňovací přístup odpovídá na otázku *„Jaká je hodnota obdobného softwarového projektu, který je na trhu již uveden?“* Jako příklad lze uvést databázi odběratelsko-dodavatelských vztahů, u které jsou definovány stejné či obdobné atributy (hodnoty) a která plní stejný účel.

Srovnávací přístup je založen na metodě násobitelů, jejíž matematický zápis je vyjádřen multiplikátorem uvedeným v příloze 6 (Vzorec 2).

Je důležité podotknout, že srovnávací přístup by měl být využit na shodných a dostatečně podobných nehmotných aktivech v ne příliš vzdáleném čase od data ocenění srovnatelného nehmotného aktiva. Vzhledem k různorodosti, inovativnosti, jedinečnosti či k odlišné finanční náročnosti a jiné fázi hospodářské zralosti

[98] *„Current accounting practice in determining software value is based on the cost spent in software development. This approach fails to account for the efficiency with which software has been produced or the quality of the product.“* (J. de GROOT a kol., 2012)

obdobných SW nelze jednoduše srovnávací přístup použít při stanovení hodnoty softwarových projektů[99].

Metodu lze využít jako podpůrnou či doplňující metodu pro stanovení hodnoty pomocí nákladového či výnosového přístupu.

Nákladový přístup

Nákladový přístup poukazuje na ochotu subjektu zaměnit nehmotné aktivum za jiné ve srovnatelné ceně a užitečnosti. Příkladem může být nabytý software, jehož hodnota je menší nebo rovna vlastním nákladům vynaloženým na výzkum a vývoj obdobného softwaru se stejnou užitečností. Nákladový přístup odpovídá na otázku *„Je subjekt schopen vytvořit vlastní činností obdobný softwarový projekt za nižší nebo stejné náklady?“*.

Nákladový přístup rozlišuje oceňovací metody náklady reprodukce a náklady nahrazení. Oba vzorce jsou uvedeny v příloze 6[100].

Oceňování softwarových projektů nákladovým přístupem je užitečný zejména u nových softwarových řešení stavěných na zelené louce[101] či u projektů, které se vyskytují v rané fázi svého životního cyklu a které jsou například chráněny patenty. Obecně lze nákladové ocenění použít u projektů *„s nízkými náklady obětované příležitosti a vysokou pravděpodobností úspěšné substituce ze strany zájemce“.* (SVAČINA, P. 2010) I přes dokladovou doložitelnost[102] nákladů se oceňovatel může setkat s úskalími, která mohou značně zkomplikovat odpovídající stanovení hodnoty SW projektů. Obvykle se jedná o nepoměr[103] mezi skutečně vynaloženými

[99] Srovnávací přístup lze použít například při stanovení hodnoty doménových jmen, patentů, přihlášek, označení a jiných technických řešení.

[100] Metoda nákladů reprodukce (Vzorec 3).

Metoda nákladů nahrazení (Vzorec 4).

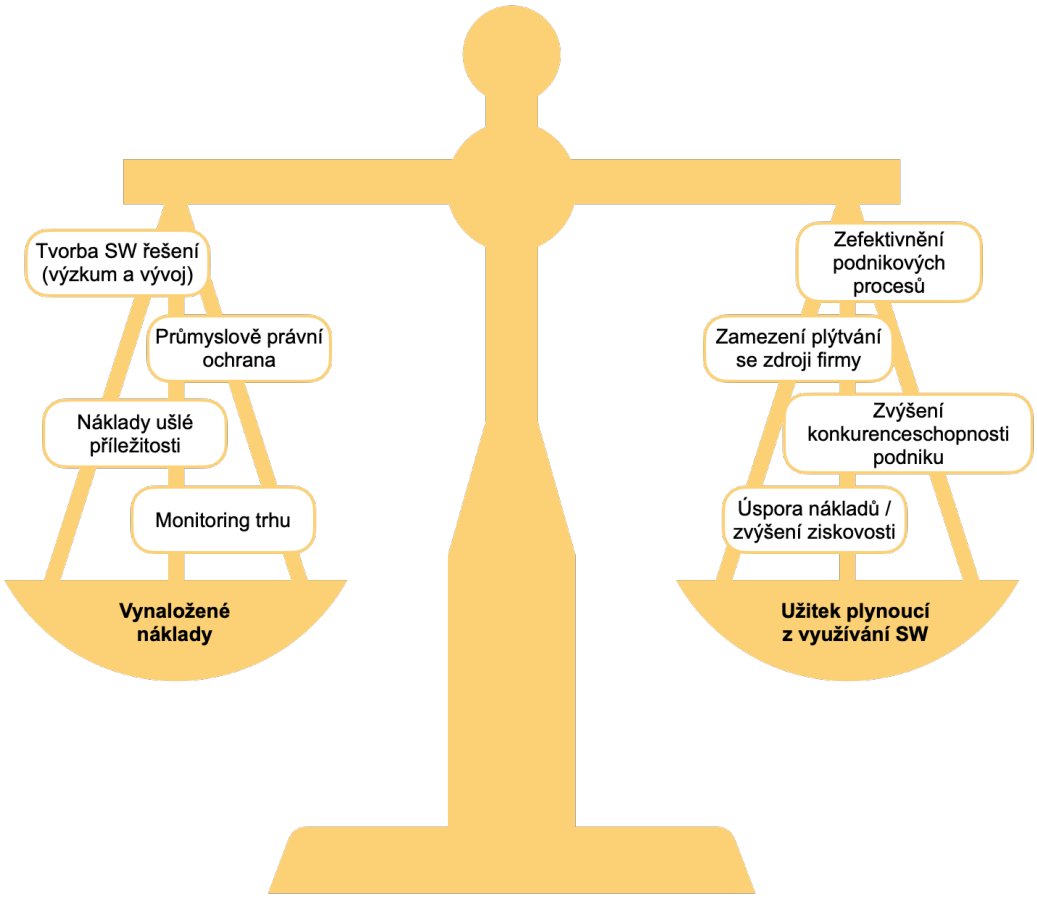
[101] Termín pro zcela nová inovativní řešení, které na trh doposud nikdo neuvedl.

[102] Účetní doklady a jiné interní účetní informace.

[103] Nákladový přístup nemusí být vhodnou metodou při ohodnocování SW projektu, neboť může neefektivní softwarový proces vést k vyššímu odhadu hodnoty SW projektu. Z toho vyplývá, že nemusí platit úměra „čím vyšší náklady, tím hodnotnější SW“. Do nákladů na vývoj SW mohou vstupovat položky, které nutně nepřinášejí hodnotu. Jako například náklady na vývoj funkcionalit, které nebudou obsaženy ve finálním řešení, náklady na experimentování s technologiemi či přístupy nebo ostatní režijní výdaje (jako např. cestovné a stravné externích poradců), které zásadním způsobem nepřispívají k vyšší hodnotě SW projektu.

náklady na výzkum a vývoj SW řešení a skutečným užitkem, který SW řešení podniku přináší.

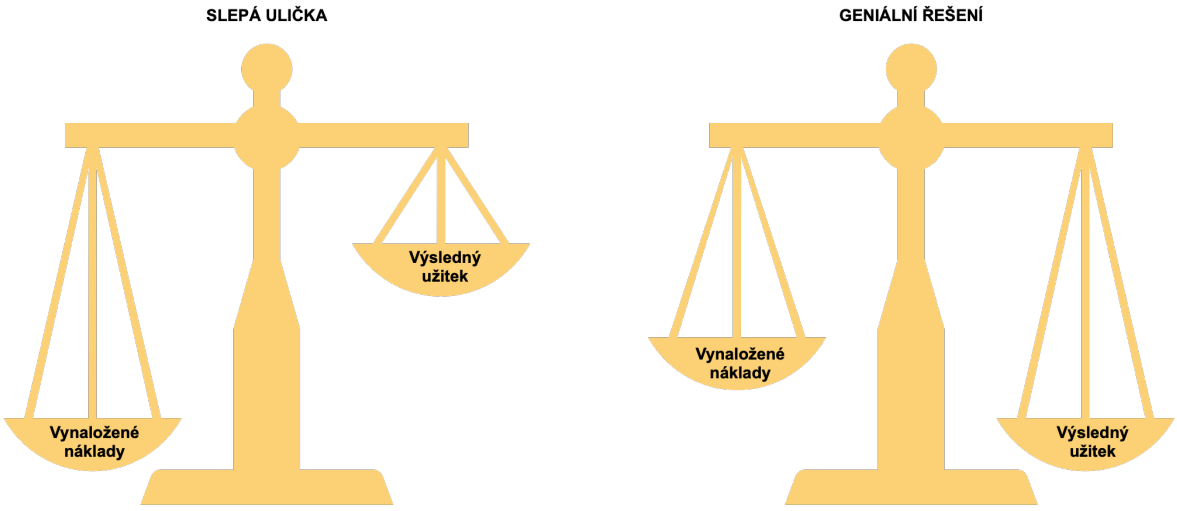
Obr. 16 Úskalí nákladového přístupu věrného a poctivého oceňování



Zdroj: autorka

Náklady ušlých příležitostí mohou u softwarových projektů, zejména těch dlouhodobých, nabývat významných hodnot. Za takové náklady ušlých příležitostí lze považovat například potenciální úspory nebo vysoké zisky v situacích, kdy výzkum a vývoj trvá delší čas, než bylo původně předpokládáno. Příkladem může být situace firmy, která investuje do rozvoje podnikového ICT na úkor rozvoje výrobní linky. Náklady na monitoring trhu souvisejí s rešerší již vytvořených softwarů a jejich patentových registrací. Pokud monitoring trhu potvrdí jedinečnost SW projektu, je vhodné uvažovat o získání průmyslové právní ochrany, s čím souvisí náklady na patentového zástupce a správní poplatky spojené s patentovou přihláškou a její registrací. Náklady na výzkum a vývoj SW řešení mohou podléhat nepatrnému zkreslení, neboť jej nelze standardizovat. V praxi mohou nastat dvě situace.

Obr. 17 Možný pohled na softwarové projekty



Zdroj: autorka

První z nich je výzkum a vývoj SW řešení, který na první pohled může působit přínosně, ovšem v pozdějších fázích se projekt dostane do slepé uličky s vidinou minimálních užitků. I přes neúměrně vynaložené náklady na jeho výzkum a vývoj je lepší SW projekt opustit. Druhou situací je tvorba geniálního řešení, u kterého jsou vysoké náklady brány jako zanedbatelné položky, neboť přínosy z nasazení softwarového řešení do podnikových procesů značně, mnohdy i několikanásobně, převyšují vyčíslené náklady.

Výnosový přístup

Výnosový přístup je založen na očekávaných příjmech. Příkladem může být software sloužící ke kontrole plátcovství DPH, který subjektu pohlídá spolehlivé plnění daňových závazků obchodních partnerů vůči státu. Přínosem může být včasná detekce nespolehlivého plátce DPH, a tím pádem zamezení nedobrovolného převzetí daňových povinností za obchodního partnera. Výnosový přístup odpovídá na otázku *„Jaký užitek subjekt očekává využitím softwarového projektu?“*. Pochopení podstaty a různorodosti přínosů je u výnosových metod oceňování klíčové.

„Hodnotu nehmotného aktiva ve výnosovém přístupu můžeme teoreticky vnímat jako rozdíl mezi hodnotou podniku, který dané aktivum užívá, a hodnotou podniku (projektu), které takové či srovnatelné aktivum neužívá.“ (SVAČINA, P. 2010)

Oceňovatel může posuzovat změny na úrovni zisků či ztrát, toků peněžních prostředků nebo úrovní rizika dosažení kýžených výsledků. Příkladů ekonomických očekávání, které mají kladný dopad na hodnotu softwarového projektu je celá řada:

- růst marže;
- vyšší objemy prodaných výrobků a služeb;
- úspora přímých či nepřímých nákladů (snížení potřeby lidského kapitálu, zefektivnění lidské práce, úspora materiálu, efektivní využití energetických a jiných zdrojů);
- rozdílnou investiční náročností;
- úsporou času a nákladů obětované příležitosti;
- získání nových závazných příležitostí (zvýšení konkurenceschopnosti, získání nových odběratelů);
- snížení dopadu rizika podnikání či jiných vnitřních i vnějších vlivů, které by mohly ohrozit řádné fungování podniku.

Jako příklad lze uvést situaci, kdy podnik na účetní oddělení nasadí RPA[104] robota, jehož úkolem bude zpracování rutinních záležitostí. Podnik může s přesností určit, kolik času věnovala jedna účetní zpracování dané faktury a jaká byla pravděpodobnost neúmyslné chyby[105], za kterou by podnik zaplatil[106]. Tyto výsledky lze přímo porovnat s výsledky RPA robota a vyvodit konkrétní závěry. Díky této technologické modernizaci může dojít k ideální symbióze účetní s RPA robotem, jejichž výsledkem bude rychlé vedení podvojného účetnictví bez chyb a prostor pro řízení firmy na základě účetních dat v reálném čase.

Přínosy z nehmotného aktiva mohou přicházet z různých zdrojů v různých podobách, proto je nezbytně nutné, aby oceňovatel v rámci výnosových metod vyjádřil prospěch stanoveným modelovým způsobem.

---

[104] Robotic Process Automation je technologie stavových automatů (založených na podmínkách „když-tak“ nebo na strojovém učení s prvky umělé inteligence a schopností rozhodovat se dle algoritmů), které vykonávají autonomní práci softwarových robotů a pomáhají tak automatizovat rutinní podnikové procesy.

[105] Nesprávně přepsaný variabilní symbol, chybně stanovené datum splatnosti, špatná fakturační částka, nesprávně určená měna aj. chyby způsobené selháním lidského faktoru.

[106] Formou úroků, penále či jinými nepeněžními formami (poškození dobrého jména firmy aj.).

Všechny výnosové metody ocenění jsou založeny na odhadu zbývající doby životnosti. Životnost lze vymezit různými způsoby (SVAČINA, P. 2010):

- právní;
- smluvní;
- firemní;
- fyzická;
- analytická;
- ekonomická.

Právní životnost je dána zbývající délkou trvání ochrany práv duševního vlastnictví. U softwaru je právní životnost vázaná na život jeho tvůrce s přesahem dalších 70 let po dnu jeho úmrtí. Naopak u patentů je životnost omezená na zákonem stanovenou lhůtu 20 let. Smluvní životnost se vztahuje ke konkrétní délce smluvního vztahu. Jedná se například o kontrakt uzavřený mezi dvěma spolupracujícími subjekty v dlouhodobém časovém horizontu. Firemní životnost je vyjádřena pravděpodobnou dobou stálosti podniku, který i nadále hodlá pokračovat ve své činnosti. Ukazatelem je například stabilita podniku a jeho vnějšího okolí, jeho finanční zdraví, síla firemní identity. Fyzická životnost[107] nehmotného aktiva je limitovaná životností hmotného nosiče a jeho případné obtížné nahraditelnosti. Analytická životnost je založena na základě statistických hodnot. Při pravděpodobnostních výpočtech se zohledňují úbytky vyplývající ze statistických dat. Ekonomická životnost souvisí se ziskovostí produktu a délkou inovačního cyklu odvětví. Odpovídá na otázku *„Dosahuje oceňovaný produkt alespoň běžné tržní úrovně ziskovosti?“*. U softwaru se předpokládá, že životní cyklus bude velmi svižný.

Obecně je nutné výsledný odhad zbývající životnosti určit na úrovni té nejkratší z výše uvedených životností. V praxi konec životnosti odpovídá neschopnosti dosahovat s daným aktivem ekonomických užitků z důvodu morálního zastarání či objevení modernějších a přínosnějších řešení na trhu.

[107] Jako hypotetický příklad lze uvést software, který byl vyvíjen pro určitý typ operačního systému fungujícím pouze na konkrétním zařízení. V případě nefunkčnosti zařízení bude software bezcenný.

Grafické znázornění i vzorce odhadu průměrné doby životnosti jsou uvedeny v příloze 6[108].

S odhadem zbývající doby životnosti souvisí i otázka morálního zastarání, které může i u softwarových řešení nabývat významného charakteru. Opět ale platí, že morální zastarání by mělo být posuzováno individuálně, neboť některá SW řešení mohou mít nadčasový přesah. Příkladem takového řešení může být bankovní software na správu pohledávek, který byl tvořen individuálně na zakázku pro interní potřeby banky.

Výnosový přístup kvalifikuje hned několik metod:

- licenční analogie;
- podíl na zisku;
- metoda prémie;
- čistá současná hodnota;
- metoda diskontní míry pro výnosové ocenění nehmotných aktiv;
- nadměrné zisky;
- vyhlášková metoda a jiné[109].

Metoda licenční analogie je považována za jednu z nejrozšířenějších a nejhojněji[110] používaných metod výnosového přístupu. *„Licenční analogie je založena na úvaze, že ohodnocovaný nehmotný majetek by byl prodán prostřednictvím úplatné smlouvy třetí osobě, případně by bylo poskytnuto právo výlučného užívání takového řešení. Za to by byly placeny poplatky po dobu platnosti smlouvy…“* (ČADA, K. 2010) Příkladem vhodného použití této metody může být ocenění globálně použitelného softwaru kancelářských aplikací.

Matematické vyjádření metody licenční analogie je uvedeno v příloze 6 (Vzorec 7).

---

[108] Grafické znázornění (viz Obr. 24).
Odhad průměrné zbývající doby životnosti nového nehmotného aktiva (Vzorec 5).
Odhad průměrné zbývající doby životnosti užívaného nehmotného aktiva (Vzorec 6).

[109] Mezi další oceňovací metody založené na výnosovém přístupu lze zařadit také metodu reálné opce, technology-factor či brand-equity. Dalšími metodami se diplomová práce vzhledem k omezenému rozsahu práce nezabývá.

[110] Využívá se zejména při ocenění práv duševního vlastnictví, která jsou obchodována formou licenčních a jiných obdobných smluv.

„Metoda licenční analogie v sobě kombinuje jak prvek výnosového přístupu (diskontování), tak prvek srovnávacího přístupu (výše licenčního poplatku).“ (SVAČINA, P. 2010) V rámci ocenění je vhodně užít metodu licenční analogie v kombinaci s další metodou výnosového přístupu.

Výnosová metoda podílu na zisku je založena na odhadu hodnoty nehmotného aktiva, které se díky užití při obchodních transakcích podílí na současné hodnotě podílu na zisku. O matematickém vyjádření metody podílu na zisku pojednává vzorec 8 uvedený v příloze 6.

„Byť je tato metoda ve srovnání s licenční analogií méně precizní v použitých tržních datech, přináší velmi cennou informaci o vztahu zisku a hypotetického licenčního poplatku.“ (SVAČINA, P. 2010)

Metoda prémie je založena na odhadu hodnoty nehmotného aktiva vyjádřením hodnotové prémie, kterou mu lze přímo přiřadit. Metoda je založena na předpokladu, že tržní zájemce chce za nehmotné aktivu zaplatit maximálně tolik, na kolik vychází dodatečný přínos hodnotové prémie. Prémii, která vychází zejména z podstaty přínosů nehmotných aktiv, lze rozlišovat například:

- cenovou,
- nákladovou,
- ziskovou, nebo
- prémii z výnosnosti kapitálu.

Cenová prémie *„staví na myšlence, že výrobek, ve kterém je užito oceňované nehmotné aktivum, je prodáván za vyšší cenu oproti výrobku stejné funkce, ve kterém takové aktivum buď není, nebo není spotřebiteli jako významné vnímáno.“* (SVAČINA, P. 2010) Příkladem může být produkt značky Apple, který má v prodejní ceně zakalkulovanou hodnotu nehmotného aktiva[111], a technologicky a funkčně stejný produkt neznámého výrobce, který bude z logiky (právě o absenci hodnoty nehmotného aktiva) levnější. Matematické vyjádření cenové prémie je uvedeno v příloze 6 (Vzorec 9).

[111] Hodnotu práv duševního vlastnictví (ochranné známky, patenty, průmyslové vzory, užitné vzory a jiné).

Nákladová prémie je založena na myšlence čisté nákladové úspory průměrných nákladů (AC) při srovnatelných objemech (Q*) a srovnatelných cenách (P*). Příkladem může být SW na podporu plánování a řízení výroby, který na základě měřitelných dat[112] pomáhá vhodně rozvrhovat výrobu s cílem maximalizovat objem dodaných výrobků. Díky nasazení tohoto softwaru výrobní společnost uspoří nemalé náklady vzniklé plýtváním a neefektivním rozvržením výroby. Pokud by v tomto případě byly dobře identifikovány[113] nákladové úspory, lze k metodě nákladové prémie přiřadit vyšší váhu v rámci ohodnocování předmětného SW řešení. O matematickém vyjádření nákladové prémie pojednává vzorec 10 uvedený v příloze 6.

Zisková prémie je obdoba cenové prémie, ovšem za použití ziskové marže. *„Stejně jako u cenové prémie, zdrojem ziskové prémie mohou být relativně vyšší ceny, příp. v kombinaci s nižšími průměrnými náklady.“* (SVAČINA, P. 2010) Matematické vyjádření je uvedeno v příloze 6 (Vzorec 11).

Prémie z výnosnosti kapitálu nabízí alternativní srovnání rentability kapitálu podniku, které využívá nehmotné aktivum s rentabilitou kapitálů srovnatelných společností. Matematické vyjádření prémie z výnosnosti je uvedeno v příloze 6 (Vzorec 12).

Jako příklad lze uvést stejné SW řešení jako u nákladové prémie. Díky využití SW pro plánování a řízení výroby vzroste produktivita výroby a objem dodaných výrobků, čímž dojde k vyšší ziskovosti podniku. Podniky, které takové SW řešení nebudou využívat budou mít nižší objem vyrobených výrobků díky plýtvání a neefektivnímu rozvržení výroby.

Metodu prémie lze považovat obecně za metodu podpůrnou k ostatním výnosovým metodám. Lze jí využít zejména pro ocenění nehmotných aktiv, která jsou pro podnik klíčová a která nesou hlavní soutěžní (konkurenční) výhodu.

[112] Data o stavu objednávek, o tvorbě rezervy skladových zásob a její vhodné využití, vyhrazení místa pro skladování a času pro dokončení rozpracované výroby, zamezení výrobních prostojů, vyhrazení vhodného času pro pravidelnou údržbu a snaze o zamezení neplánovaných odstávek z důvodu poruch, stanovení priorit podle efektivnosti výroby a výhodnosti s ohledem na finanční dopady.

[113] Vhodně argumentačně podepřeny a věcně, na základě dostupných dat, dokázány.

Účelem metody čisté současné hodnoty[114] je zjistit maximálně možnou a pro obě strany únosnou úplatu za oceňované nehmotné aktivum. Proměnnými veličinami jsou peněžní toky, které vhodněji vyjadřují užitek plynoucí zájemci o nehmotné aktivum. Metoda čisté současné hodnoty slouží jako *„podpůrná metoda pro odhad maximální výše relativní úplaty“*. (SVAČINA, P. 2010) O matematickém vyjádření této metody pojednává vzorec 13 v příloze 6.

Oproti ostatním metodám přináší *„do přímé souvislosti výši maximální úplaty, investiční náročnost výroby, peněžní toky a ekonomickou životnost“*. (SVAČINA, P. 2010) Příkladem vhodného užití této metody může být licenční výroba, kdy podnik investuje určitou výši nákladů do výzkumu a vývoje patentu či užitného vzoru a následně jej poskytne formou licenční smlouvy k užití během výrobního procesu jiného podniku.

Metoda diskontní míry pro výnosové[115] ocenění nehmotných aktiv vychází *„z obecně vyšších preferencí spotřeby současného kapitálu před spotřebou budoucí. V budoucnosti získaný (nakoupený) kapitál (peněžní tok) proto podléhá principu obětované příležitosti, která se do výpočtu promítá technikou diskontování.“* (SVAČINA, P. 2010) Výše ušlé příležitosti je závislá na čase[116] a riziku[117]. Diskontní míra nehmotných aktiv se stanovuje pomocí modelu oceňování kapitálových aktiv (CAMP), který lze matematicky vyjádřit pomocí vzorce uvedeného v příloze 6 (Vzorec 14).

Díky metodě diskontní míry lze promítnout do výše ocenění všechna specifická rizika pocházející z právních, technických či ekonomických faktorů. Jako vhodný příklad praktického užití této metody lze uvést specifický a pro medicínu významný výrobní SW, který pomocí algoritmu pomůže s výrobou vhodného složení individuálních léčiv na základě výsledků krevního obrazu pacienta. Jedná se o patentové řešení, které bylo spojeno s vysokými vynaloženými náklady na výzkum a vývoj.

[114] Metoda je podle principu a interpretace podobná metodě prémie. Metoda čisté současné hodnoty, na rozdíl od metody prémie, nevychází z účetních (akruálních) dat, a proto by mělo být vyjádření užitku adekvátnější.

[115] Diskontní míru lze využít i u nákladového přístupu ocenění.

[116] Čas mezi současnou a budoucí spotřebou kapitálu.

[117] Riziko, které musí podnik podstoupit při investování do nehmotného aktiva.

Metoda nadměrných zisků[118] zohledňuje přímé zhodnocení podílu na zisku veškerých užitých aktiv. Metoda *„vychází z potřeby ocenit často velmi specifická nehmotná aktiva, která tvoří významnou soutěžní výhodu, ovšem jsou natolik specifická“* (SVAČINA, P. 2010), že je nelze jednoduše ocenit ostatními výnosovými metodami. Příkladem může být, pro podnik významná a specifická, databáze smluvních odběratelů a obchodních partnerů, díky níž získává podnik silnou konkurenční výhodu. Matematické vyjádření hodnoty nadměrných zisků je uvedeno v příloze 6 (Vzorec 15). Metodu nadměrných zisků je vhodné použít také jako doplňkovou k ostatním oceňovacím metodám.

Vyhlášková metoda vychází ze zákona 151/1997 Sb., o oceňování majetku. Uvedený zákon říká, že majetková práva vytvořená tvůrčí činností vyplývající z průmyslově-právní a autorsko-právní povahy se oceňují výnosovým způsobem uvedeným v příloze 6 (Vzorec 16).

„Z důvodu užití pevně stanovených některých parametrů modelu a vážných nejasností ohledně odhadu čistého výnosu alternativními cestami a ostatních parametrů“ (SVAČINA, P. 2010) se doporučuje vyhláškovou metodu užívat pouze ve vyhrazených případech stanovených výše uvedenými právními předpisy.

2.4 Proces ohodnocování softwarového projektu

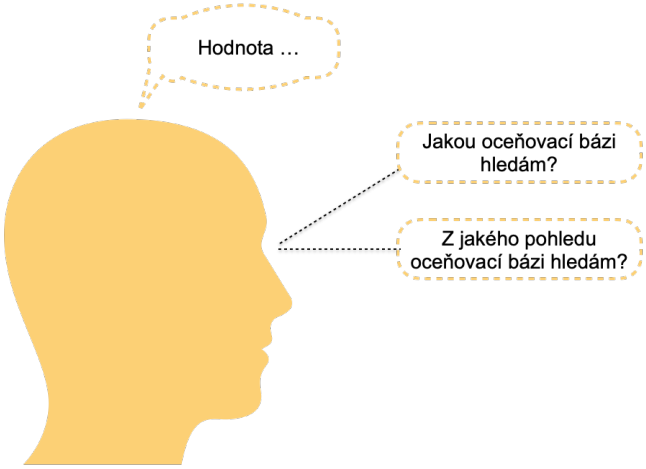
Práce oceňovatele se na první pohled jeví jako pestrá činnost, při které zkoumá různé předměty v odlišných podmínkách s mnohdy turbulentním vývojem tržního prostředí v čase. Ve skutečnosti se oceňovatel setkává s různými úskalími, kterým musí čelit. Za největší úskalí je považována legislativa, která je v různých zemích, i přes sjednocování národních oceňovacích předpisů s Mezinárodními oceňovacími standardy, odlišná.

Jako příklad lze uvést problematiku české oceňovací báze, která je ukotvena do kontextu zákona č. 151/1997 Sb., o oceňování majetku a která se vyznačuje svébytnými rysy. Jelikož se jedná o jedinou závaznou oceňovací bázi v České republice, oceňovatel není nucen konkrétně vymezovat pojem hodnota a při své práci zjišťuje pouze číselné vyjádření hodnoty posuzované věci. Je důležité

[118] Použití metody vyžaduje hlubokou znalost účetní, daňové i oceňovací teorie s hlubším pochopením platných předpisů (IFRS 3 a jiné související legislativy). Praktické použití metody je velmi náročné.

podotknout, že správné a podrobné vymezení oceňovací báze hraje klíčovou roli, neboť jí je vyjádřena optika pohledu na hodnotu posuzovaného předmětu.

Obr. 18 Optika pohledu oceňovatele



Zdroj: autorka

Pokud oceňovatel v posudku blíže nespecifikuje, s jakou oceňovací bází pracoval, je vyjádření hodnoty téměř nepoužitelné. *„Bez pečlivého definování pojmu hodnota nemají závěry obsažené v oceňovacím posudku žádný význam“.* (MAŘÍK, M. 2011)

Dalším úskalím je posuzování hodnoty zkoumaného předmětu. Pro oceňovatele není vůbec snadné určit bodový odhad hodnoty, i přes známé hypotézy tržní hodnoty a teoretický základ vycházející ze závazných legislativních předpisů. I přes tyto komplikace je nutné, aby bodový odhad hodnoty byl stanoven rozumně po pečlivém zvážení všech specifických rysů a zároveň, aby stanovený bodový odhad hodnoty byl vhodně argumentačně podepřen.[119]

V neprospěch stanovení hodnoty nehmotných aktiv hraje i absence netržních hodnot v české legislativě, která může značně komplikovat práci oceňovatele. Z tohoto důvodu je klíčové, aby oceňovatel byl schopen zvážit všechny specifické rysy posuzovaného předmětu.

Na hodnotu posuzovaného předmětu je vhodné nahlížet také subjektivně. Oceňovatel posuzuje, jaký užitek plyne konkrétnímu subjektu (například kupujícímu, prodávajícímu, stávajícímu vlastníkovi anebo konkrétní společnosti s určitým

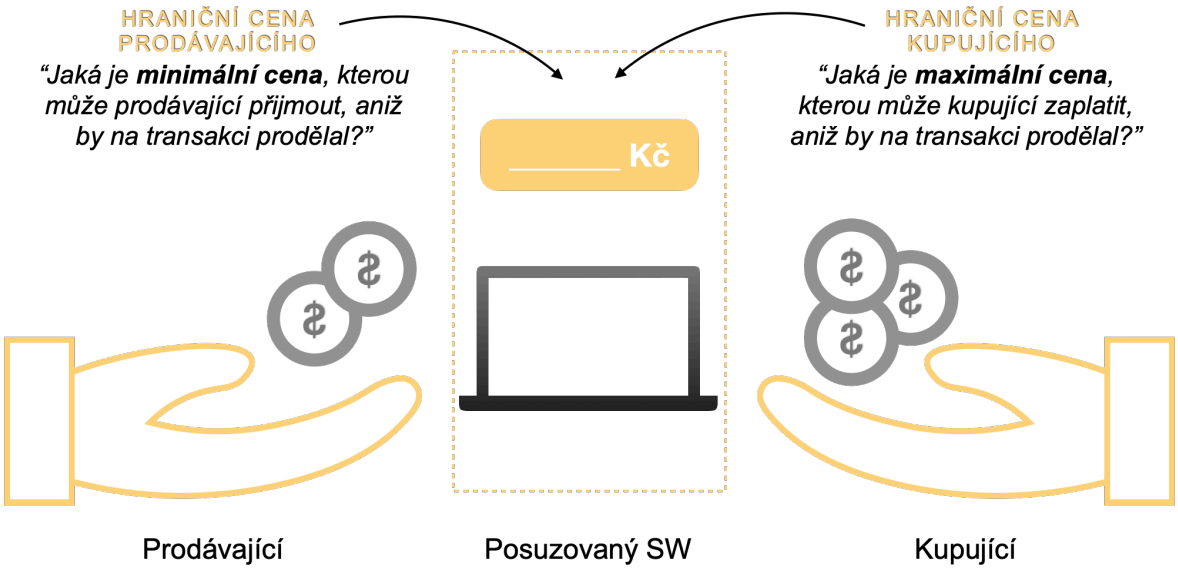
[119] Ve výsledku by *„každé ohodnocení mělo být pravdivé, odpovídající zjištěným podmínkám a okolnostem, bez vztahu k věci a účastníkům. Předpokladem přezkoumatelnosti takové práce je její přesnost a snaha popsat zjištěné okolnosti jasně a úplně.“* (ČADA, K. 2010)

záměrem). *„Na tomto subjektivním postoji je založena tzv. Kolínská škola. Kolínská škola zastává názor, že ocenění nemá smysl modifikovat v závislosti na jednotlivých podnětech, ale na obecných funkcích, které má ocenění pro uživatele jeho výsledků.“* (MAŘÍK, M. 1998) Kolínská škola rozeznává několik základních funkcí, mezi které patří (MAŘÍK, M. 1998):

- funkce poradenská;
- funkce rozhodčí;
- funkce argumentační;
- funkce komunikační;
- funkce daňová.

Funkce poradenská je považována za nejdůležitější, neboť poskytuje informace o maximální a minimální ceně posuzovaného předmětu. Tato funkce hledá hraniční cenu, která vymezuje prostor pro cenová jednání.

Obr. 19 Význam hraničních cen prodávajícího a kupujícího



Zdroj: autorka podle MAŘÍK, M. 1998

Hraniční ceny dávají oběma stranám transakce informace pro vlastní rozhodování. Oceňovatel při stanovení hraničních cen může zohlednit všechna specifika a potenciál posuzovaného SW i veškeré jeho přínosy pro kupujícího.

Funkce rozhodčí ztvárňuje výkon funkce nezávislého oceňovatele, který by měl *„alespoň odhadnout hraniční ceny účastníků transakce, a nalézt spravedlivou cenu v rámci odhadnutého rozpětí.“* (MAŘÍK, M. 1998)

Funkce argumentační poskytuje oběma stranám transakce na základě všech dostupných informací pádné argumenty, které mohou při vyjednávání o ceně zlepšit pozici dané strany.

Funkce komunikační slouží k podávání informací o posuzovaném předmětu externím subjektům, jako například bankám, investorům a jiným zájmovým skupinám.

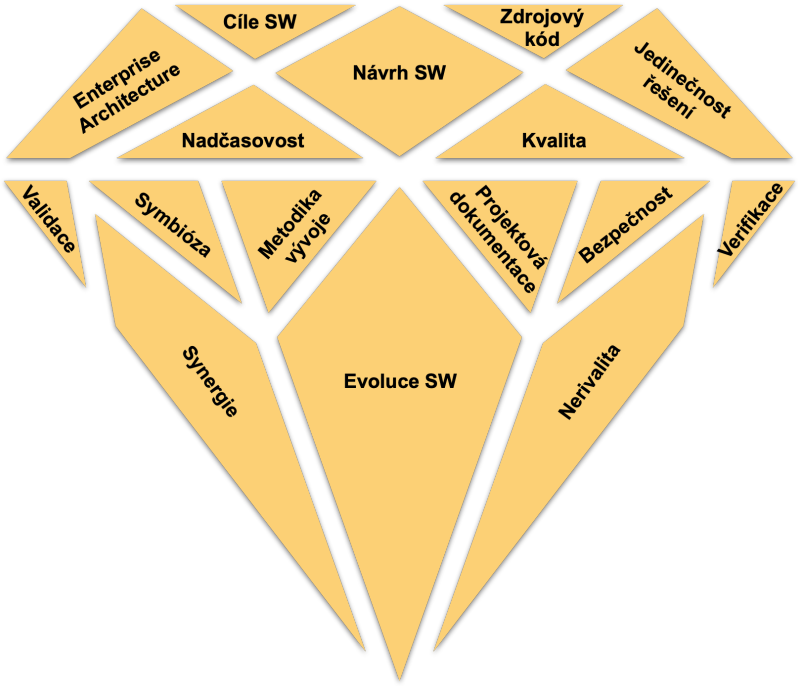
Funkce daňová poskytuje podklady pro daňové záležitosti.

V rámci oceňování softwarových projektů i dalších nehmotných aktiv je možné brát na zřetel myšlenky profesora Krabce (2009), který ve svém díle vnímá oceňování aktiv jako behaviorální vědu v prostředí reálného světa, ve kterém *„není možno spoléhat se na modely dokonalé konkurence, úplných informací a racionality v rozhodování jednotlivce.“* (KRABEC, T. 2009) Otázka tedy zní, zdali by byla snaha oceňovatele, který by zjišťoval onu tržní cenu interpretovanou ve znaleckém posudku jako obecně platnou s pomocí alespoň dvou exaktních výpočtů, správná? Není potřeba na proces stanovení hodnoty SW pohlížet daleko hlouběji, komplexněji a promyšleněji?

Z těchto důvodů lze oceňování softwarů považovat za disciplínu, která uzavírá pomyslný kruh tvůrčích aktivit softwarového procesu. Oceňovatel se na samém konci softwarového procesu stává brusičem, který prostřednictvím základních funkcí[120] objektivně utváří z opracovaného kamene (softwaru) opravdový klenot s nevyčíslitelnou hodnotou.

[120] Výše uvedené základní funkce Kolínské školy.

Obr. 20 Role oceňovatele na konci softwarového procesu



Zdroj: autorka

3 Vývoj specializovaného softwaru

Praktická část práce se zabývá vývojem prototypu dvou specializovaných softwarů. První z nich se zaměřuje na automatizovanou detekci nespolehlivých plátců DPH s generováním dopisů adresovaných všem detekovaným subjektům. Druhý prototyp softwaru ve zjednodušené verzi napomáhá se stanovením hodnoty SW projektu s přihlédnutím ke všem specifikům ze SW projektu vyplývajících.

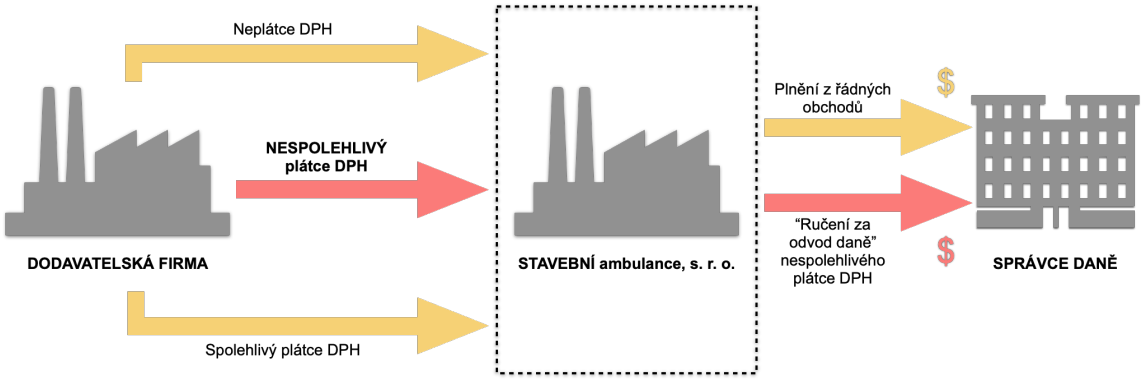
3.1 Detekce nespolehlivých plátců DPH

Společnost STAVEBNÍ ambulance, s. r. o. (IČ 09319559, spolehlivý plátce DPH, bankovní spojení 27220667/0100) od roku 20X1 provozuje svou činnost v oblasti stavebnictví a projekčních služeb. Zaměřuje se na rekonstrukce starých staveb, demolice rozpadlých objektů s následnou recyklací starého stavebního materiálu, projekční práce s důrazem na rekonstrukce či výstavby hospodárných staveb. Svým klientům nabízí komplexní služby od prvotních návrhů až po finální realizace vedoucích ke splnění zákazníkových představ a potřeb.

V rámci své činnosti spolupracuje s desítkami obchodních partnerů. Od roku 20X16 se firma pravidelně zabývá nepříjemnými, a pro firmu velmi ztrátovými, transakcemi s finančním úřadem kvůli odběru služeb a výrobků od nespolehlivých plátců DPH[121]. Stav odvodů za daňové ručení dle § 109, ZDPH byl k 1. 1. 20X23 téměř 7 milionů Kč.

[121] Dle § 106a, zákona o DPH je nespolehlivým plátcem subjekt, který poruší závažným způsobem své povinnosti vztahující se ke správě daně. Dle metodických pokynů GFŘ se za zvlášť závažné porušení povinností považuje například tříměsíční kumulativní nedoplatek na DPH ve výši minimálně 10 mil. Kč bez příslušenství, nebo vyměřená nebo doměřená daň podle pomůcek za porušení stanovených povinností plátce daně v minimální výši 500 tis. Kč. Za nespolehlivého plátce může být i subjekt, který opakovaně neplní lhůty pro podání daňových tvrzení nebo se účastnil na daňovém podvodu.

Obr. 21 Ručení za nespolehlivé plátce DPH



Zdroj: autorka

Společnost STAVEBNÍ ambulance si je vědoma povinností vyplývajících z legislativních předpisů ČR i z povinného[122] ručení za odvod daně nespolehlivého plátce DPH, od kterého odebírala výrobky a služby (přijímala zdanitelná plnění). Do budoucna by však chtěla průtahům se správcem daně předejít a vyhnout se odvodům DPH za nespolehlivé plátce DPH. Rozhodla se, že si vlastní činností vytvoří software na detekci nespolehlivých plátců DPH, který společnost upozorní na nesrovnalosti zveřejněné v Registru ekonomických subjektů[123] (dále jen „*Registr*“) a případně vygeneruje dopis vyzývající dodavatelskou společnost o nápravu zveřejněných údajů či o nápravu své pozice vůči správci daně. Software bude pracovat automaticky bez obsluhy zaměstnance v době pracovního volna. Čas na automatickou obnovu dat bude nastaven na časné ranní hodiny (mezi 02:00 a 04:00). V případě, že se v interním adresáři obchodních partnerů objeví nespolehlivý plátce DPH či zveřejněný bankovní účet, který není totožný s bankovním spojením uvedeným na účetních dokladech[124], bude vygenerován report nespolehlivých subjektů dle Registru[125]. Po přihlášení do systému bude příslušný zaměstnanec upozorněn na vygenerovaný přehled nespolehlivých subjektů a vyzván k řešení situace. Lze definovat následující způsoby řešení:

[122] Ex Lege, tzn. ručení vyplývající ze zákona. Podmínky pro uplatnění daňového ručení jsou upraveny v § 171, odst. 1, daňového řádu.

[123] Pro zjednodušení případové studie je Registr ekonomických subjektů provozovaný Ministerstvem financí pouze simulován. Veškerá data uvedená v případové studii jsou smyšlená.

[124] Faktury, Smlouvy o dílo, Realizační smlouvy a jiné.

[125] Ve skutečnosti jde o Registr nespolehlivých plátců DPH, ve kterém lze nalézt údaje o subjektu DPH (například DIČ, název a sídlo subjektu, zveřejněné bankovní účty či údaje o registraci k DPH). Registr nespolehlivých plátců DPH provozuje Ministerstvo financí ČR.

- V případě nesrovnalostí u bankovního spojení může společnost STAVEBNÍ ambulance písemně vyzvat obchodního partnera k neprodlené nápravě evidovaných nepřesností v *Registru ekonomických subjektů*. Výzvu je nutné zaslat doporučeným dopisem nebo datovou schránkou.
- V případě evidování Nespolehlivého plátce DPH může společnost dle § 109a, ZDPH využít zvláštního způsobu zajištění daně a vyčíslené DPH zaplatit (místo dodavateli) správci daně. Společnost se tak vyhne dvojímu zaplacení DPH z jednoho poskytnutého zdanitelného plnění.
- Písemně[126] vyzvat detekovaného obchodního partnera k objasnění situace[127]. Včasnou detekcí může společnost s dodavatelem rozvázat spolupráci, anebo obchodního partnera vyzvat ke společnému řešení s cílem i nadále za určitých podmínek spolupracovat.

Software bude pracovat se vstupními daty obsaženými v CSV souboru, který bude vygenerován z interního informačního systému. Vstupní data budou obsahovat identifikační číslo (IČ) subjektu, jeho název, ID datové schránky (ID DS) a bankovní spojení vyplývajícího z účetních dokladů.

Výstupními daty jsou detekované subjekty, jejich identifikace, zveřejněné bankovní účty a informace o nespolehlivém plátcovství.

Specifikace požadavků

Cílem SW *Detekce nespolehlivých plátců DPH* je automatická detekce obchodních partnerů evidovaných v *Registru nespolehlivých plátců* (dále jen jako „Registr“) jako plátci nespolehliví, nebo plátci, kteří pro obchodní transakce uvádí jiné bankovní spojení než bankovní spojení zveřejněné v Registru.

Software po zpracování úlohy vygeneruje vyzývací dopisy určené všem detekovaným subjektům.

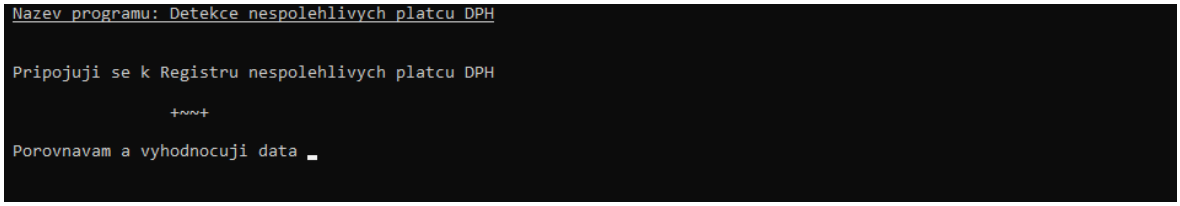
[126] Znění vyzývacího dopisu je uvedeno v příloze 9.

[127] Společnost by měla zjistit, z jakého důvodu je dodavatelská firma na seznamu nespolehlivých plátců DPH a zda je neprodleně schopna a ochotna začít plnit své zákonné závazky vůči státní správě.

Návrhová dokumentace

Hlavní program se automaticky připojí k Registru a následně porovná a vyhodnotí data obsažená v obou databázích.

Náhled softwaru 1 Program po spuštění

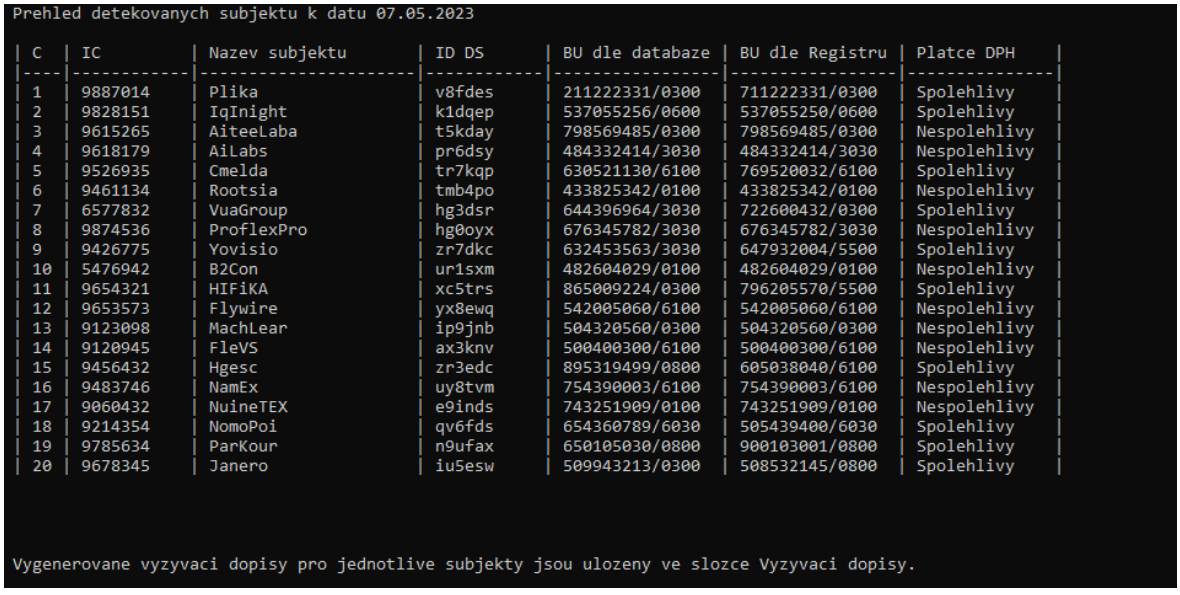


Zdroj: autorka

Aby mohl provést požadovanou úlohu a vygenerovat vyzývací dopisy, je třeba provést následující operace:

- Porovnat, jestli bankovní účet užívaný při obchodních transakcích souhlasí s bankovním účtem zveřejněným v Registru nespolehlivých plátců DPH.
- Vyhodnotit, jestli subjekt není v Registru evidován jako nespolehlivý plátce DPH.

Náhled softwaru 2 Prehled detekovaných subjektů



C	IC	Nazev subjektu	ID DS	BU dle databaze	BU dle Registru	Platce DPH
1	9887014	Plika	v8fdes	211222331/0300	711222331/0300	Spolehlivy
2	9828151	IqInight	k1dqep	537055256/0600	537055250/0600	Spolehlivy
3	9615265	AiteeLaba	t5kday	798569485/0300	798569485/0300	Nespolehlivy
4	9618179	AiLabs	pr6dsy	484332414/3030	484332414/3030	Nespolehlivy
5	9526935	Cmelda	tr7kqp	630521130/6100	769520032/6100	Spolehlivy
6	9461134	Rootsia	tmb4po	433825342/0100	433825342/0100	Nespolehlivy
7	6577832	VuaGroup	hg3dsr	644396964/3030	722600432/0300	Spolehlivy
8	9874536	ProflexPro	hg0oyx	676345782/3030	676345782/3030	Nespolehlivy
9	9426775	Yovisio	zr7dkc	632453563/3030	647932004/5500	Spolehlivy
10	5476942	B2Con	ur1sxm	482604029/0100	482604029/0100	Nespolehlivy
11	9654321	HIFiKA	xc5trs	865009224/0300	796205570/5500	Spolehlivy
12	9653573	Flywire	yx8ewq	542005060/6100	542005060/6100	Nespolehlivy
13	9123098	MachLear	ip9jnb	504320560/0300	504320560/0300	Nespolehlivy
14	9120945	FleVS	ax3knv	500400300/6100	500400300/6100	Nespolehlivy
15	9456432	Hgesc	zr3edc	895319499/0800	605038040/6100	Spolehlivy
16	9483746	NamEx	uy8tvm	754390003/6100	754390003/6100	Nespolehlivy
17	9060432	NuineTEX	e9inds	743251909/0100	743251909/0100	Nespolehlivy
18	9214354	NomoPoi	qv6fds	654360789/6030	505439400/6030	Spolehlivy
19	9785634	ParKour	n9ufax	650105030/0800	900103001/0800	Spolehlivy
20	9678345	Janero	iu5esw	509943213/0300	508532145/0800	Spolehlivy

Zdroj: autorka

Po dokončení úloh program vygeneruje výstupní soubor *Přehled nespolehlivých plátců DPH* a *Vyzývací dopis* pro jednotlivé detekované subjekty. Vyzývací dopisy

jsou uloženy ve složce Vyzývací dopisy, kterou program během zpracování úloh vytvoří.

Součástí návrhové dokumentace je vývojový diagram, který je uveden v příloze 7.

Plán projektu

Projekt vývoje SW byl rozdělen do 12 týdnů, ve kterých dle vodopádového modelu prošel vybranými fázemi softwarového procesu (analýza problému, návrh vývojového diagramu, vývoj programu).

V průběhu vývoje softwaru došlo k nesprávné analýze způsobu zpracování vstupních dat, proto byl softwarový proces jednou navrácen ze systémového návrhu (a částečně spustitelného zdrojového kódu) zpět na definování problému a revidování původních požadavků.

Uživatelská příručka

Software je plně automatizovaný, a proto k jeho ovládání netřeba uživatelského zásahu.

Uživatel vstupuje do procesu detekce nespolehlivých plátců až po vygenerování vyzývacích dopisů, které by měl bez odkladu prostřednictvím datové schránky zaslat detekovaným obchodním partnerům.

Dokumentace kódu

Zdrojový kód softwaru využívá knihovny definované standardem ANSI (<stdio.h>, <math.h>, <stdlib.h>, <unistd.h>, <time.h>). Pracuje se vstupními soubory ve formátu *.cvs* a s výstupními soubory ve formátu *.csv* a *.txt*.

K vyhodnocení dat slouží cyklus *while*, díky kterému jsou procházeny jednotlivé řádky obou vstupních souborů. Pomocí příkazu *if* dochází k podmíněnému vyhodnocení vybraných dat z obou souborů, na jejichž základě dojde k vygenerování vyzývacích dopisů.

Součástí projektové dokumentace kódu je zdrojový kód, který je uveden v příloze 8. V příloze 9 této diplomové práce je ukázka textového výstupu programu.

Závěrečná zpráva

Podnik díky programu získá prostor pro včasnou detekci obchodních partnerů vedených vůči státní správě jako nespolehlivých, díky čemuž může zabránit ze zákona vyplývajícím povinnostem v podobě ručení za nespolehlivého plátce DPH.

3.2 Oceňování SW

Společnost by ráda svůj kontrolní SW *Detekce nespolehlivých plátců DPH* nechala ohodnotit soudním znalcem, aby jej mohla za úplatu poskytnout dalším subjektům či jej vložit jako dílčí část základního kapitálu nově vznikající dceřiné společnosti. Ráda by znala reálnou a věcně podloženou hodnotu vlastní činností vyvinutého SW, která nebude ani podhodnocená, ani nadhodnocená a bude plně respektovat legislativní rámec oceňování majetku.

Soudní znalec si pro tento účel vytvořil postup ohodnocující kroky softwarového procesu a jednoduchý prototyp SW, který by mu pomohl se stanovením hodnoty SW projektu. Základními stavebními kameny oceňovacího (ohodnocujícího) SW jsou:

- matematické oceňovací modely, založené na srovnávacím, nákladovém a výnosovém přístupu, a
- koeficient, který je výsledkem subjektivně přiřazených hodnot (od 0 do 10) k jednotlivým krokům softwarového procesu zohledňujícího specifika nehmotných aktiv a průběh vývoje SW.[128]

V první fázi oceňovacího procesu lze vymezit dobu použitelnosti SW. V případě tohoto softwaru lze dobu použitelnosti stanovit jako neurčitou[129], neboť je vysoká pravděpodobnost dlouhodobého[130] užití SW v podnikové praxi bez jeho zásadních funkčních úprav.

---

[128] K subjektivnímu posouzení může být nápomocná ilustrace *Ohodnocení softwarového procesu*, která je uvedena v příloze 12.

[129] Například IAS 38 rozlišuje dobu použitelnosti nehmotného aktiva na dobu s konečnou použitelností a na dobu s neurčitou použitelností.

Dobu s konečnou použitelností lze uvažovat u softwaru, který podléhá rychlému zastarávání. Mohlo by se jednat například o ekonomický (účetní) software bez pravidelných legislativních aktualizací. Taková doba použitelnosti by mohla být například 2 roky, tzn. doba od poslední k následující novele legislativy.

[130] Institut Nespolehlivého plátce DPH je upraven novelou zákona o dani z přidané hodnoty, která nabyla účinnosti 1. ledna 2013.

Cílem projektové dokumentace je seznámit čtenáře s předloženým SW, jeho architekturou, funkcionalitami a postupem používání. Součástí projektové dokumentace je vývojový diagram, zdrojový kód a ilustrace *Ohodnocení softwarového procesu*.

Specifikace požadavků

Cílem SW je stanovit hodnotu SW na základě získaných dat od objednatele ocenění a na základě subjektivního posouzení jednotlivých kroků softwarového procesu.

SW disponuje výpočtem hodnoty dle srovnávacího, nákladového a výnosového přístupu prostřednictvím 13 různých oceňovacích metod. SW, na základě subjektivního posouzení oceňovatele, umožňuje ve výsledné hodnotě zohlednit jednotlivé kroky softwarového procesu. Toho uživatel docílí prostřednictvím koeficientu, o který bude dílčí výsledek použité oceňovací metody navýšen.

Na SW nejsou kladeny žádné bezpečnostní požadavky, jako například přístupová práva či zabezpečení dat.

SW může sloužit oceňovatelům jako podpůrný nástroj při stanovení hodnoty SW.

Návrhová dokumentace

Hlavní program je členěn do hlavní nabídky (hlavního menu), která se v první úrovni dělí na oceňovací přístupy (srovnávací, nákladový a výnosový), možnost stanovení koeficientu ohodnocujícího kroky softwarového procesu, dále pak na výsledkovou tabulku nebo na volbu ukončení programu. Volba oceňovacího přístupu a nabídka možnosti stanovení koeficientu zohledňující specifika SW a jeho procesu představují hlavní výkonné části hlavního menu i softwaru. Výsledková tabulka má charakter přehledu zvolených metod a jejich výsledků, zohledňujícího koeficientu a výslednou hodnotu SW projektu.

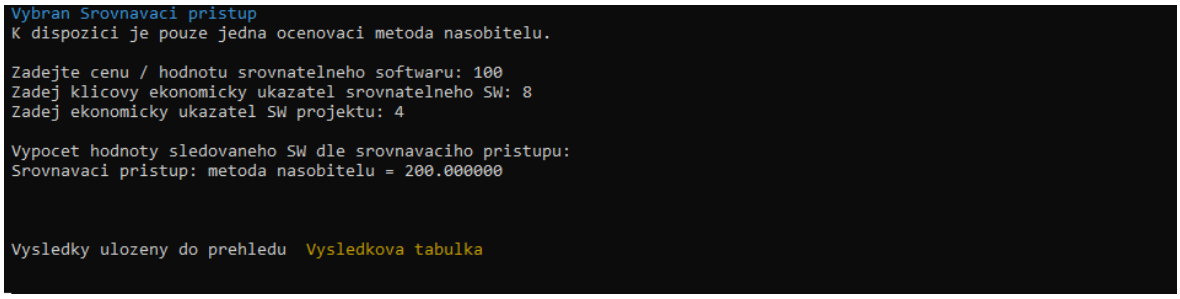
Náhled softwaru 3 Hlavní menu spuštěného programu

Zdroj: autorka

Menu druhé a případně třetí úrovně slouží k volbě konkrétní metody ocenění daného oceňovacího přístupu. Ve zdrojovém kódu jsou tyto metody řešeny formou podprogramu[131], který vykonává úlohu výpočtů jednotlivých metod ocenění. Výkonné tělo představuje vzorec jednotlivých oceňovacích metod, jehož vstupními parametry jsou proměnné vyplývající z podstaty oceňovacího přístupu a metody ocenění. Výstupním parametrem je penězi vyjádřená hodnota SW projektu, která je i návratovou hodnotou podprogramů.

Po zvolení oceňovací metody je uživatel programem vyzván, aby zadal hodnoty jednotlivých proměnných. Po provedení úlohy program zobrazí souhrn, jehož obsahem je rekapitulace zvoleného oceňovacího přístupu, metody ocenění a (dílčí) výsledek hodnoty SW projektu. Uživatel je také informován o uložení výsledků do výsledkové tabulky.

Náhled softwaru 4 Ukázka stanovení hodnoty vybranou metodou ocenění

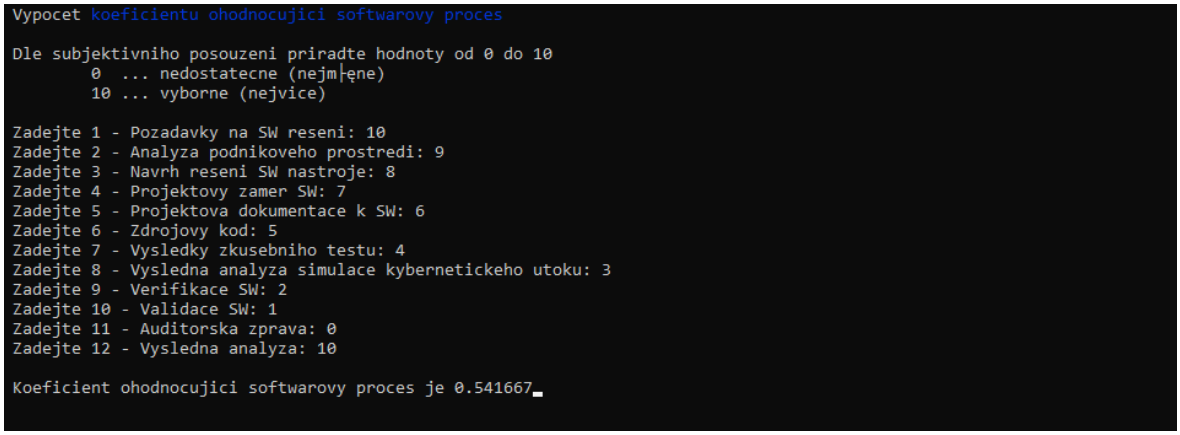


Zdroj: autorka

Nabídka sloužící k výpočtu koeficientu ohodnocujícího softwarový proces slouží k subjektivnímu posouzení a následnému přiřazení hodnot dílčím krokům softwarového procesu, jeho použitým metodikám, postupům a dalším okolnostem, které vedly ke vzniku SW. Uživatel je vyzván, aby k jednotlivým krokům softwarového procesu přiřadil hodnoty od 0 do 10, kde 0 znamená nedostatečné a 10 je přiřazována těm krokům, které lze subjektivně posoudit jako vzorně vykonané. Koeficient je považován za druhou dílčí část oceňovacího procesu, který má vliv na výslednou hodnotu SW.

[131] *„Jestliže potřebujeme provádět určitý úkon na několika různých místech programu, je obvykle nežádoucí programovat jej pokaždé znovu. Tomuto postupu je možné se vyhnout; kód (nazývaný podprogram, subroutine) zapíšeme jen do jednoho místa a doplníme k němu několik málo instrukcí, které po dokončení podprogramu řádně obnoví činnost vnějšího programu.“* (KNUTH, E. D. 2008)

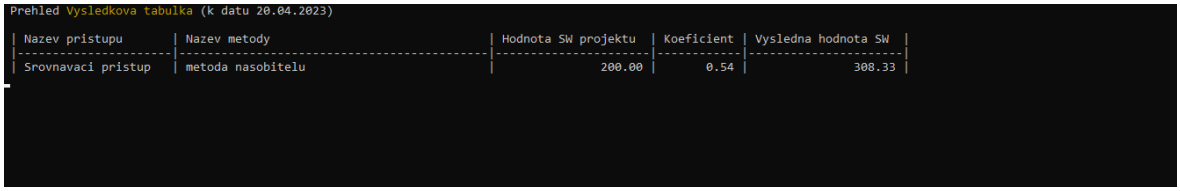
Náhled softwaru 5 Přiřazení hodnot ke krokům softwarového procesu



Zdroj: autorka

Výsledkem je koeficient v desetinném tvaru, který je uložen ke všem zvoleným metodám ve výsledkové tabulce.

Náhled softwaru 6 Výsledková tabulka



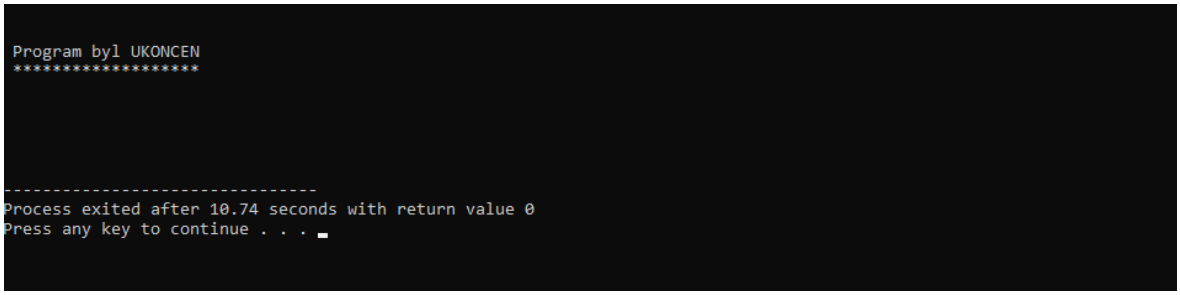
Zdroj: autorka

Výsledková tabulka má strukturu 5 sloupců, které v jednotlivých řádcích obsahují název přístupu, název metody ocenění, hodnotu SW projektu, koeficient dle ilustrace *Ohodnocení softwarového procesu* a výslednou hodnotu SW. Počet řádků výsledkové tabulky závisí na počtu zvolených metod. Výsledková tabulka mimo jiné zobrazuje datum, kdy byly výpočty provedeny.

Po dokončení úloh se program zastaví a čeká na uživatelovu reakci. Jakmile uživatel zobrazené informace přečte, může se automaticky vrátit na hlavní menu stisknutím kterékoliv klávesy.

Po provedení všech zamýšlených operací lze program ukončit stisknutím klávesy 6 na hlavním menu.

Náhled softwaru 7 Ukončení programu



Zdroj: autorka

Součástí návrhové dokumentace je vývojový diagram, který je uveden v příloze 10.

Plán projektu

Projekt vývoje SW je rozdělen do 12 týdnů, které jsou dále rovnoměrně rozděleny dle spirálového modelu do třech oblastí. První čtyři týdny byly vyhrazeny pro určení cílů, přípustných alternativ a rozsahového omezení. Byl sestaven seznam požadavků na funkcionality, návrh jejich řešení a pravděpodobná představa výsledné podoby SW. Výsledkem cyklu byl vývojový diagram. Ve druhém cyklu vývoje vznikalo hlavní funkční tělo SW, podprogramy se všemi funkcionalitami, hlavní nabídka i sub-menu druhé a třetí úrovně a podoba výsledkové tabulky. Výsledkem druhého cyklu byl spustitelný zdrojový kód. Poslední cyklus se zabýval odstraněním funkčních softwarových chyb a vzhledových (kosmetických) nedostatků. Byla otestována funkčnost hlavní funkce i jednotlivých uživatelských funkcí (podprogramů). Výsledkem cyklu je Oceňovací SW.

Uživatelská příručka

Oceňovací SW pomáhá oceňovateli se stanovením hodnoty SW projektu a s ohodnocením jednotlivých kroků softwarového procesu. Spuštění programu probíhá přes spustitelný *.exe* soubor.

Uživatel by měl po spuštění nejprve zvolit některý z oceňovacích přístupů a oceňovacích metod. Volbu lze učinit stisknutím klávesy s požadovaným číslem z nabízeného seznamu a potvrzením této volby klávesou *enter*.

Po volbě je vyzván k zadání jednotlivých proměnných, které program dosadí do platných vzorců oceňovacích metod. Uživatel si může zvolit libovolný počet metod ocenění. Po provedení zvolených úloh se může uživatel navrátit do hlavní nabídky stisknutím libovolné klávesy.

Jestliže má uživatel dostupná data k posouzení jednotlivých kroků softwarového procesu, může přejít k výpočtu koeficientu zohledňujícího softwarový proces. V tomto kroku je uživatel vyzván, aby zadal hodnoty od 0 do 10. Výsledek je zapsán do výsledkové tabulky ke všem zvoleným metodám ocenění. S posouzením kroků SW procesu může pomoci ilustrace *Ohodnocení softwarového procesu*, která je součástí přílohy 12.

Výsledková tabulka slouží jako přehled zvolených metod, jejich dílčích výsledků a výsledného koeficientu. Zároveň provádí výpočet dílčího výsledku metody ocenění a koeficientu, jehož výstupem je hodnota posuzovaného SW.

Dokumentace kódu

Zdrojový kód softwaru využívá následující knihovny definované standardem ANSI:

- <stdio.h>
- <conio.h>
- <math.h>
- <stdlib.h>
- <string.h>
- <unistd.h>
- <time.h>

Knihovna <*stdio.h*> představuje knihovnu pro základní *„funkce, typy a makra pro vstup a výstup“.* (KERNIGHAN, W. B., RITCHIE, M. D. 2006) Slouží pro práci s konzolí, k načtení vstupních dat ze souborů nebo k uložení výstupních dat do souborů v požadovaných formátech nebo pro práci s řetězci.

Knihovna <*conio.h*> slouží k ovládání konzole, ke čtení a zápisu kláves nebo k manipulaci s barevnými grafickými obrazci.

Knihovna <*math.h*> deklaruje matematické funkce a makra, díky nimž lze vykonávat matematické operace. Knihovna je určena pro programování matematických výpočtů nebo strojového učení.

Knihovna <*stdlib.h*> *„deklaruje funkce pro konverze čísel, alokace paměti a podobné úlohy“.* (KERNIGHAN, W. B., RITCHIE, M. D. 2006) Díky této knihovně

lze dynamicky alokovat paměť, což je užitečné pro práci s rozsáhlými řetězci nebo datovými strukturami. Tato knihovna je považována za jednu z nejdůležitějších knihoven jazyka C.

Knihovna <*string.h*> slouží pro zjednodušení práce s řetězci, které představují pole se znaky ukončené nulou (\0).

Knihovna <*time.h*> „deklaruje typy a funkce pro manipulaci s daty a časy. Ve zdrojovém kódu je použita funkce *strftime*, která pomocí definované struktury *%d.%m.%Y*[132] zobrazuje ve Výsledkové tabulce datum.

Pořadí výpočtu běhu programu určují příkazy, které představují řídící konstrukce programu. Ve zdrojovém kódu je možné se shledat s následujícími:

- while;
- if–else;
- else if;
- for.

Cyklus *while* je považován za výraz, který provádí příkaz k vyhodnocení výrazu. Cyklus opakuje určitý blok zdrojového kódu do té doby, dokud platí stanovená podmínka. Jakmile se podmínka stane nepravdivou (nulovou), dojde k ukončení cyklu a k pokračování v následujících operacích programu.

Cyklus *if – else* slouží k podmíněnému rozhodnutí, které je rozděleno na dvě funkční části. První částí je if, které vyhodnotí stanovenou podmínku (pravdivou, nenulovou) a následně provede blok zdrojového kódu. Pokud je podmínka nepravdivá (nulová), provede se blok zdrojového kódu, který je uveden pod *else*.

Konstrukce *else if* slouží jako doplňkové podmíněné formátování ke konstrukci *if-else* pro víceúrovňové větvení.

Ve zdrojovém kódu jsou cykly *while*, *if-else*, *else if* použity pro jednotlivé větve programu, které se provedou v závislosti na uživatelské volbě konkrétního oceňovacího přístupu nebo metody ocenění.

[132] *%d* znamená den v měsíci v číselném rozmezí 01 až 31, *%m* uvádí měsíc v číselném rozmezí 01 až 12 a *%Y* zobrazuje rok ve formátu RRRR.

Cyklická konstrukce *for* dovoluje opakovat určitou část zdrojového kódu v závislosti na splnění podmínky. Tato konstrukce byla ve zdrojovém kódu použita k vytváření výsledkové tabulky, která je závislá na počtu zvolených metod.

Zdrojový kód využívá také příkaz *goto* k návratu do hlavního menu. Jedná se o *„přerušení výpočtu nějaké hluboce zanořené struktury, například dvou a více cyklů naráz.“* (KERNIGHAN, W. B., RITCHIE, M. D. 2006)

V rámci zdrojového kódu jsou jména proměnných nazývaných tak, aby nejlépe vystihovala podstatu proměnné. Jména proměnných obsahují podtržítko (_), které slouží k zajištění čitelnosti názvů dlouhých proměnných.

Zdrojový kód používá u deklarovaných proměnných následující datové typy:

- int,
- char,
- float.

Datový typ *int* představuje celé číslo a je jím deklarován například počet metod. Datový typ *char* je schopný uchovávat znak a ve zdrojovém kódu je použit například pro názvy oceňovacího přístupu či metody ocenění. Datový typ *float* je vymezen pro čísla s pohyblivou řádovou čárkou. Jedná se o veškeré numerické hodnoty vyjádřené desetinným číslem.

Zdrojový kód obsahuje aritmetické operátory (+, -, *, / a %), relační operátory (<, <=, >=, >), logické operátory (==, !=) a operátory inkrementace (++).

Formátovaný vstup je zajištěn pomocí standardní funkce *scanf*, která *„načítá znaky ze standardního vstupu, interpretuje je podle specifikací v řetězci format a ukládá výsledky do zbývajících argumentů“.* (KERNIGHAN, W. B., RITCHIE, M. D. 2006)

Formátovaný výstup zajišťuje standardní funkce *printf*, která převádí programové hodnoty na znaky. Programové hodnoty jsou konverzí zobrazeny pomocí *%* a příslušného znaku. Pro číslo v desítkové soustavě je použit znak *d*, pro znaky z řetězce je použit znak *s* a pro hodnoty datového typu *double* je využit znak *f*.

Součástí projektové dokumentace kódu je zdrojový kód, který je uveden v příloze 11.

Oceňovatel díky SW získá prostor pro tvůrčí zhodnocení všech jedinečností a specifik posuzovaného SW řešení. Výsledné hodnoty SW, které oceňovateli program předloží, mohou sloužit jako dílčí podklad pro stanovení hodnoty SW projektu.

Závěr

Svět se dostává do nové digitální éry, kdy uprostřed veškerého dění stojí člověk se svými tvůrčími schopnostmi a znalostmi a okolo něj se vyskytuje nepředstavitelné množství technologických možností. Možností otevírajících bránu do zcela nového digitálního světa, ve kterém software a hardware hrají nepostradatelnou roli ovlivňující veškeré dění ve společnosti a v podnicích. Základem technologického rozvoje je lidský potenciál, který dokáže vnímat problémy jako příležitosti pro vznik nových, neotřelých či inovativních řešení, detaily jako kouzlo pro tvorbu promyšlených a propracovaných děl a neutuchající smysl pro rozvoj již dokonalých systémů. Spolu s citem pro rozlišení skutečných potřeb, specifikaci požadavků a závazkem psát ten nejlepší kód, jakého je lidská mysl v daný okamžik schopna, se z lidského programátora stává umělec ztvárňující projekt se skutečnou přidanou hodnotou. Všechny tyto předpoklady odlišují programátora od umělé inteligence. Umělá inteligence (prozatím) nedokáže naprogramovat software, který zadavatel skutečně potřebuje, ale zvládne v nesrovnatelné rychlosti naprogramovat nepředstavitelné množství softwarů v různých variacích, které zadavatel chce. A právě pochopení této myšlenky může přivést účastníky softwarového procesu na správnou cestu, jejíž cílem je nalezení odpovídajícího bodového odhadu hodnoty (ocenění, ohodnocení) softwarového projektu.

Diplomová práce si dala za cíl vysvětlit význam vývoje specializovaného softwaru v podmínkách dynamicky rozvíjejícího se světa s důrazem na kvalitu, kybernetickou bezpečnost a skutečnou potřebu řešení, jehož výsledkem je naplňování klíčových procesů a stanovených cílů podniku. Cílem práce je také poukázat na důležitost oceňování softwaru se zohledněním specifických vlastností, které vyplývají z tvůrčí mysli účastníků softwarového procesu.

Předložená diplomová práce se nejprve zabývá softwarovým procesem, který představuje množinu uspořádaných, promyšlených a na sebe navazujících aktivit ovlivňujících vývoj softwaru. Za nezbytný prvek softwarového procesu je nutné považovat jeho účastníky a jejich vzájemnou interakci, jejíž kvalita má významný dopad na výsledek softwarového procesu i na kvalitu výsledného SW projektu. V případě dobré interakce účastníků softwarového procesu vznikne komunikační síť, která je předpokladem vzniku dobrého a zajímavého SW i spokojených uživatelů

na straně zadavatele. Prvním klíčovým výsledkem dobré interakce je rozhodnutí o volbě vhodné metodiky vývoje SW, ze které lze, mimo jiné, odhadnout předpokládané informace o termínu dokončení SW projektu či o jeho předpokládaném rozpočtu. S volbou metodiky souvisí i volba konkrétního modelu vývoje SW, který vývojovému týmu vymezuje praxí prověřené fáze, postupy a úlohy. Jedná se o soubor doporučení, které nejsou pro vývojový tým závazný, nicméně jejich dodržování může vést k úspěšnému dokončení SW projektu ve vymezeném termínu a ve stanoveném finančním rozpočtu. Vývojový tým může na základě svých úvah dostupné modely vývoje SW libovolně modifikovat a kombinovat, čímž může vzniknout životní cyklus obohacený o další aktivity a rozhodovací úlohy. Příkladem takového obohacení mohou být otázky spojené s podnikovou architekturou (EA): *„Jakým směrem podnik směřuje?“*, *„Jaká je IT vyspělost podniku?“* nebo *„Lze předpokládat, že vyvíjený SW nástroj bude v ideální symbióze s podnikovým informačním systémem a SW nástroji?“*. Díky tomuto uvědomění může vzniknout SW řešení, které bude nadčasové a použitelné i v době technologického zrání firmy. Vývojový tým by se měl u SW projektu zaměřit i na jeho kvalitu, které není snadné dostát, ale velmi jednoduše jí lze rozpoznat. Není ani snadné ji zcela přesně vyjádřit a definovat, neboť na kvalitu SW lze nahlížet subjektivně. Je však důležité kvalitu SW posuzovat alespoň z hlediska bezpečnosti, neboť veškeré chyby obsažené v SW nástroji mohou být v temném světě kybernetických útoků velmi zranitelné.

Důležitým úkolem diplomové práce je vymezit metody oceňování nehmotných aktiv, které by mohly být vhodné pro stanovení hodnoty softwarových projektů s důrazem na zohlednění jedinečných specifik a kvalit, kterými se softwarové projekty mohou vyznačovat. I pro zkušené a praxí prověřené oceňovatele může být stanovení bodového odhadu hodnoty softwaru zatěžkávající zkouškou, neboť nebyl doposud stanoven závazný předpis upravující postup ohodnocení nehmotných aktiv. Hodnota softwaru se doposud odvíjela zejména od vypočtené ceny, jejíž proměnné byly v podobě nákladových položek za vyměřený čas vývojářů a dalších účastníků softwarového procesu, za vyčíslení odpisů využitých aktiv a jiných výdajů přímo i nepřímo přiřaditelných k vývoji SW projektu. Ovšem vypočtená cena a hodnota SW není totéž. Obě veličiny mají rozdílnou logiku i cíle, což může způsobit podhodnocení nebo nadhodnocení softwarových projektů. Jelikož laická veřejnost, neznalá strojového jazyka a labyrintu algoritmů, nedokáže ocenit umění krásy

softwarů, hraje v softwarovém procesu oceňovatel klíčovou roli. Znalecký posudek oceňovatele může být například vhodnou přílohou účetního výkaznictví, ve kterém se čtenář, mimo běžné finanční a účetní informace, může dozvědět o důvodech tvorby SW i o všech jeho dopadech na prosperitu firmy. Čtenářem může být například investor, který uvítá nezaujaté informace o tom, proč firma investovala do konkrétního vývoje softwarového řešení, jaké přínosy jí softwarové řešení přinesly a jakou hodnotu software ve vlastnické struktuře firmy má.

Praktická část práce je demonstrována na vývoji dvou specializovaných softwarových prototypů vznikajících z klíčových vybraných fází softwarového procesu. Obě případové studie obsahují analýzu specifik a požadavků na vyvíjené softwarové řešení, návrh algoritmu a zjednodušenou projektovou dokumentaci softwaru. Součástí praktické části je úplné znění spustitelných zdrojových kódů a ilustrace *Ohodnocení softwarového procesu*.

Závěrem lze dodat, že vývoj softwaru zcela jistě ovlivní (obecná) umělá inteligence ve všech jejích fázích vyspělosti. Je možné, že v blízké budoucnosti bude zdrojové kódy psát umělá inteligence, která hravě lidského programátora překoná v rychlosti psaní, v dokonalé znalosti syntaxe programátorských jazyků i ve výsledné efektivitě programu. Dalo by se říci, že takový program bude mít po formální stránce tah na branku. Ovšem klíčovou problematikou bude určení vlastnictví (autorství) vyvíjeného programu umělou inteligencí a stanovení odpovědnosti za nesprávně pracující softwarové řešení. Je důležité si však uvědomit, že umělá inteligence je (prozatím) pouze jazykovým modelem bez svého vědomí, který čerpá z dostupných dat, a za kterým stojí lidští anotátoři koordinující výstupy jazykového modelu.

Seznam literatury

BENEŠ, R. *Metodologie ocenění software. K ocenění a stanovení hodnoty v IT, resp. „ceny“ software, databáze, licence, atp.* [22. února 2022] Dostupné z: https://radek-benes.cz/clanky/cl_oceneni_cena_software.pdf.

BOEHM, B. W. *A Spiral Model of Software Development and Enhancement.* Computer, vol. 21, no. 5, pp. 51–72, May 1988.

BREEDEN, J. II. *Nástroje SAST a DAST: Zajistí vám aplikace bez zranitelností.* SecurityWorld. 2022(03), str. 28–31.

BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů.* Praha: GRADA Publishing, a. s., 2005. ISBN 80-247-1075-7.

BUCHALCEVOVÁ, A. *Zlepšování procesů při budování informačních systémů.* Praha: GRADA Publishing, a. s., 2018. ISBN 978-80-245-2235-7.

BUREŠ, M., RENDA, M., DOLEŽAL, M. a kol. *Efektivní testování softwaru. Klíčové otázky pro efektivitu testovacího procesu.* PRAHA: GRADA Publishing, a. s., 2016. ISBN 978-80-247-5594-6.

BRUCKNER, T., VOŘÍŠEK, J., BUCHALCEVOVÁ, A. a kol. *Tvorba informačních systémů. Principy, metodiky, architektury.* PRAHA: GRADA Publishing, a. s., 2012. ISBN 978-80-247-4153-6.

BŘEZINA, A. *Agilní transformace. Proč bývá tak křehká?* ČESKÉ BUDĚJOVICE: KOPP nakladatelství, 2020. ISBN 978-80-7232-521-4.

CIO BUSINESS WORLD [online]. *2016: Digitální revoluce přinese tlak na zrychlení procesů a kyberbezpečnost.* 2016, Internet Info DG, a. s. [14. března 2022] Dostupné z: https://www.cio.cz/clanky/2016-digitalni-revoluce-prinese-tlak-na-zrychleni-procesu-a-kyberbezpecnost/.

ČADA, K. *Know-how a obchodní tajemství.* PRAHA. Úřad průmyslového vlastnictví, 2010. ISBN 978-80-7282-087-0.

ČADA, K. *CHRÁNIT/NECHRÁNIT, to je otázka. Výsledky výzkumu a vývoje, jejich ochrana a komercializace.* PLZEŇ: alevia, s. r. o., 2014. ISBN 978-80-905538-0-4.

DAVIS, A. M. *Software Requirements: Objects, Functions, and States.* New Jersey: Prentice Hall PTR, 1993. ISBN 978-0138057633.

DOMINGUEZ, J. *The Curious Case of the CHAOS Report 2009*. 2009 (aktualizace 2021), ProjectSmart. [15. června 2022] Dostupné z: https://www.projectsmart.co.uk/it-project-management/the-curious-case-of-the-chaos-report-2009.php.

EELES, P., CRIPPS, P. *Architektura softwaru.* BRNO: Computer Press, a. s., 2011. ISBN 978-80-251-3036-0.

ERDOGMUS, H., FAVARO, J., HALLING, M. *Valuation of Software Initiatives Under Uncertainty: Concepts, Issues, and Techniques.* Value-Based Software Engineering, pp. 39–66, 2006 Springer, Berlin, Heidelberg. ISBN 978-3-540-25993-0.

ERICKSON, J. *Hacking – umění exploitace.* BRNO: ZONER software s. r. o., 2005. ISBN 80-86815-21-8.

FOSTER, J. C. *Buffer Overflow: zneužití, detekce a prevence.* PRAHA: GRADA Publishing, a. s., 2007. ISBN 978-80-247-1480-6.

FEATHERS, C. M. *Údržba kódu převzatých programů.* BRNO: Computer Press, a. s., 2009. ISBN 978-80-251-2127-6.

GRAF, J. *Murphyho obecné počítačové zákony.* BRNO: UNIS Publishing, a. s., 1998. ISBN 80-86097-22-6.

HEROUT, P. *Testování pro programátory.* ČESKÉ BUDĚJOVICE: KOPP, 2016. ISBN 978-80-7232-481-1.

HOLWERDA, T. *WTFs/m.* [25. března 2023]. OSnews, 2008. Dostupné z: https://www.osnews.com/story/19266/wtfsm/

HOWARD, M., LEBLANC, D. *Bezpečný kód.* BRNO: Computer Press, a. s., 2008. ISBN 978-80-251-2050-7.

HULME, G. V., MARTIN, J. A. *Integrujte zabezpečení přímo do vývoje.* SecurityWorld. 2022(03), str. 24–27.

HUSEBY, S. H. *Zranitelný kód.* BRNO: Computer Press, a. s., 2006. ISBN 80-251-1180-6.

CHEUNG, W. K. *Vývojářův kód.* BRNO: Computer Press, a. s., 2013. ISBN 978-80-251-3786-4.

J. de GROOT, NUGROHO, A., BÄCK, T., VISSER, J. *What Is the Value of Your Software?* 2012 Third International Workshop on Managing Technical Debt (MTD), Zurich, Switzerland, 2012, pp. 37–44, doi: 10.1109/MTD.2012.6225998.

JIRÁNKOVÁ, J. *Enterprise Architecture.* Semestrální práce: Vývoj informačních systémů. ŠAVŠ, o. p. s. 2022.

KADLEC, V. *Učíme se programovat v jazyce C.* BRNO: Computer Press, a. s., 2002. ISBN 80-7226-715-9.

KADLEC, V. *Agilní programování. Metodiky efektivního vývoje softwaru.* BRNO: Computer Press, a. s., 2004. ISBN 80-251-0342-0.

KERNIGHAN, W. B. *Jak porozumět digitálnímu světu. Vše, co potřebujete vědět o internetu, bezpečnosti a soukromí.* PRAHA: Dokořán, s. r. o. 2019. ISBN 978-80-7363-903-7.

KERNIGHAN, W. B., RITCHIE, M. D. *Programovací jazyk C.* BRNO: Computer Press, a. s., 2006. ISBN 80-251-0897-X.

KNUTH, E. D. *Umění programování. 1. díl, Základní algoritmy.* BRNO: Computer Press, a. s., 2008. ISBN 978-80-251-2025-5.

KONEČNÝ, J. JANKOVÁ, M. DVOŘÁK, J., ŠULC, V. *Modely systémově vymezených procesů pro kybernetickou bezpečnost.* Soudní inženýrství. 2016, ročník 27, str. 197–201.

KRABEC, T. *Oceňování podniku a standardy hodnoty.* Praha: GRADA Publishing, a. s., 2009. ISBN 978-247-2854-0.

LEFFINGWELL, D. *Calculating the Return on Investment from More Effective Requirements Management.* American Programmer. 10(4), p. 13–16.

LILIENTHAL, C. *Sustainable Software Architecture. Analyze and Reduce Technical Debt.* HEIDELBERG (Deutschland): dpunkt.verlag, 2019. ISBN 978-1-68198-569-5.

MARTIN, R. C. *Čistý kód.* BRNO: Computer Press, a. s., 2009. ISBN 978-80-251-2285-3.

MAŘÍK, M. *Určování hodnoty firem.* Praha: Ekopress, s. r. o., 1998. ISBN 80-86119-09-2.

MAŘÍK, M., MAŘÍKOVÁ, P. *Hodnotové báze pro oceňování podniku – stále otevřený problém.* Odhadce a oceňování podniku č. 3-4/2011, ročník XVII, str. 37–56, ISSN 1213-8223.

MCCLURE, S., SCAMBRAY, J., KURTZ, G. *Hacking bez tajemství.* BRNO: Computer Press, a. s., 2003. ISBN 80-722-6948-8.

MCCLURE, S., SCAMBRAY, J., KURTZ, G. *Hacking bez záhad.* PRAHA: Grada Publishing, a. s., 2007. ISBN 978-80-247-1502-5.

MCCONNELL, S. *Dokonalý kód. Umění programování a techniky tvorby software.* BRNO: Computer Press, a. s., 2005. ISBN 80-251-0849-X.

MLÝNEK, J. *Zabezpečení obchodních informací.* BRNO: Computer Press, a. s., 2007. ISBN 978-80-251-1511-4.

NEGROPONTE, N. *Digitální svět. Being Digital.* PRAHA: Management Press, NT Publishing, s. r. o., 2001. ISBN 80-7261-046-5.

POŽÁR, J. *Modelování kybernetických útoků.* Policejní akademie České republiky v Praze (Fakulta bezpečnostního managementu). 2012.

ROUDENSKÝ, P. *Kvalita softwaru.* PROSTĚJOV: Computer Media s. r. o., 2016. ISBN 978-80-7402-294-4.

ROUDENSKÝ, P., HAVLÍČKOVÁ, A. *Řízení kvality softwaru. Průvodce testováním.* BRNO: Computer Press, a. s., 2013. ISBN 978-80-251-3816-8.

ŘEPA, V. *Procesně řízená organizace.* Praha: GRADA Publishing, a. s., 2012. ISBN 978-80-247-4128-4.

SOMMERVILLE, I. *Softwarové inženýrství.* Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

STANDISH GROUP. *CHAOS report 2015.* [23. února 2023] Dostupné z: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf.

SVAČINA, P. *Oceňování nehmotných aktiv.* Praha: Ekopress, s. r. o., 2010. ISBN 978-80-86929-62-0.

SZOR, P. *Počítačové viry: analýza útoku a obrana.* BRNO: ZONER software, s. r. o., 2006. ISBN 80-86815-04-8.

TÖPFER, P. *Algoritmy a programovací techniky.* PRAHA: PROMETHEUS, s. r. o., 2007. ISBN 978-80-7196-350-9.

WIDEMAN, R. M.: *A literature review – The Role of the Project Life Cycle (Life Span) in Project Management.* AEW Services, Vancouver, BC ©. 2004(02).

WIEGERS, K. E. *Požadavky na software.* BRNO: Computer Press, a. s., 2008. ISBN 978-80-251-1877-1.

Seznam právních předpisů

Český účetní standard pro podnikatele č. 013 – Dlouhodobý nehmotný a hmotný majetek.

Evropské oceňovací standardy (EVS).

IFRS 3 – Podnikové kombinace.

IFRS účetní standard IAS 38 – Nehmotná aktiva.

Mezinárodní oceňovací standardy IVS (Směrnice 1 až 4).

Vyhláška č. 500/2002 Sb., kterou se provádějí některá ustanovení zákona o účetnictví, pro účetní jednotky, které jsou podnikateli účtujícími v soustavě podvojného účetnictví.

Zákon č. 89/2012 Sb., občanský zákoník.

Zákon č. 235/2004 Sb., o dani z přidané hodnoty.

Zákon č. 237/2020 Sb., *o oceňování majetku a o změně některých zákonů (zákon č. 151/1997 Sb., o oceňování majetku).*

Zákon č. 527/1990 Sb., *o vynálezech, průmyslových vzorech a zlepšovacích návrzích.*

Zákon č. 563/1991 Sb., o účetnictví.

Zákon č. 586/1992 Sb., o daních z příjmů.

Seznam obrázků

Seznam náhledů SW

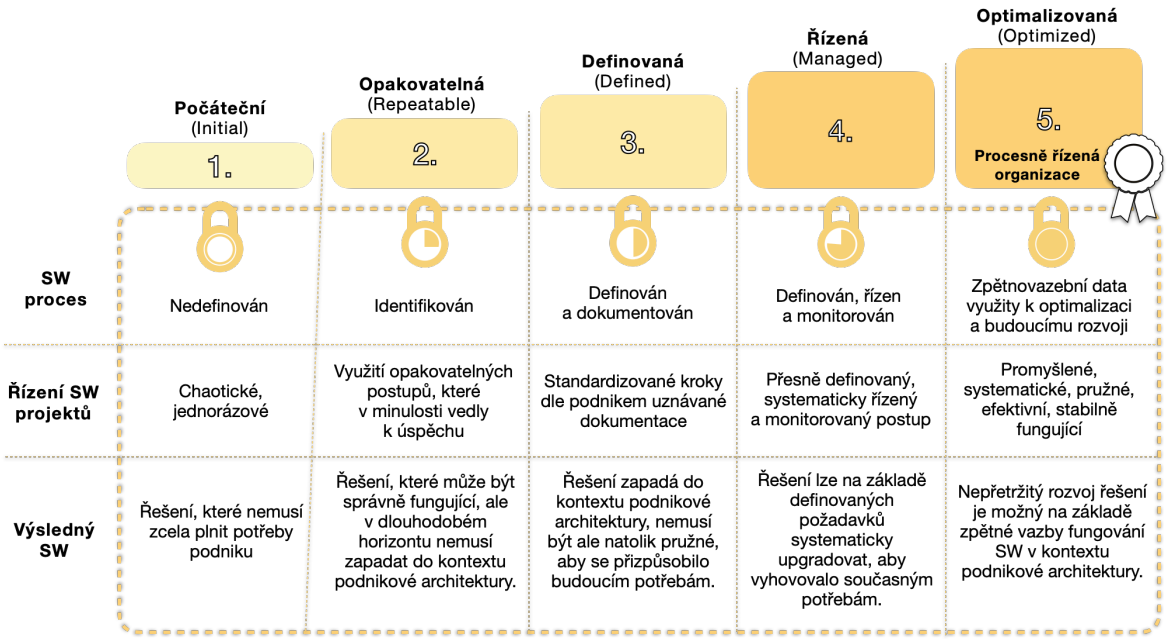
Seznam vzorců

Seznam příloh

Příloha 1 Úroveň digitální vyspělosti a možného využití procesů

Úroveň možného využití procesů ve společnosti může definovat model zralosti CMM[133], který lze považovat za zrcadlo pomáhající odpovědět na otázku *„Jak na tom firma z hlediska využití procesů vlastně je?“*. Díky tomuto uvědomění lze zjistit, jestli je firma na konkrétní typ technologie či softwarového nástroje vůbec připravená.

Obr. 22 Hodnocení ICT vyspělosti organizace



Zdroj: autorka

První, počáteční či výchozí, stupeň je zaměřen na jednorázové řešení bez vnímání širší perspektivy. V kontextu potřeb zadavatele dochází k přehlížení příležitostí, které aktuální prostředí nabízí. Dá se říci, že zadavatel nedokáže vyjádřit jeho potřeby a požadavky, což se odráží na výsledném SW, který vypadá jako by byl tvořen náhodně, bez rozmyšlení, nesystematicky a bez kontextu podnikové architektury. Druhý, opakovatelný, stupeň se vyznačuje snahou o nalezení dlouhodobého způsobu řízení opakováním postupů, které v minulosti vedly k úspěchu. V každém dalším SW projektu lze evidované postupy, požadavky a plány již zopakovat s vidinou úspěšného dokončení projektu. Třetí, definovaný, stupeň lze považovat za systematicky dokumentovatelný, čímž lze snadněji

[133] CMM (*Capability Maturity Model*) vznikl pro hodnocení zralosti vývoje SW v Software Engineering Institute na Carnegie Mellon University. CMM model je součástí metodiky COBIT, která je považována za základ pro posuzování kvality IS.

definovat řídící i výkonné aktivity a následně je prostřednictvím dokumentace uznat jako všeobecně platné. Nově vznikající SW projekty v takto technologicky vyspělé společnosti budou vyvíjeny dle podnikově platných standardů. Je vysoce pravděpodobné, že SW projekty budou úspěšně zapadat do kontextu podnikového ICT systému. Čtvrtý, řízený, stupeň umožňuje na základě předem daných identifikátorů měřit průběh, vlastnosti, funkčnosti a dosažené výsledky procesu. Dochází ke shromažďování a vyhodnocování veškerých dat, které by mohly dopomoci ke zvýšení kvality procesů i SW projektů. Výsledný SW zapadne do již zavedeného podnikového ICT systému, aniž by významně narušoval funkčnost podnikového IS. Nejvyšší, optimalizovaný stupeň, se vyznačuje promyšlenými, systematickými, efektivními, pružnými a stabilně fungujícími postupy. *„Tato úroveň je charakterizována nepřetržitým zlepšováním výsledků na základě zpětné vazby nasazení procesu a testováním nových myšlenek a technologií.“* (ŘEPA, V. 2012) Organizace na této úrovni vyspělosti bude k vývoji SW řešení přistupovat inovativně a pragmaticky. Nebude se bát riskovat zkoušet nové myšlenky, postupy a technologie či novým a kreativním způsobem přetvářet již fungující věci tak, aby díky technologiím přinášely větší přidanou hodnotu. *„Řečeno slovy M. Hammera, taková organizace systematicky přemýšlí, jak „dělat věci jinak“.“* (ŘEPA, V. 2012)

Dalším, mnohdy doplňujícím, modelem je model CMMI[134], který vychází z modelu CMM. CMMI je vhodný pro řízení vývoje složitých a rozsáhlých informačních systémů a softwarových nástrojů, u kterých je nutné kombinovat různé technologie, metodologie či výstupy od různých subjektů. Příkladem může být koordinace zadavatele, vývojového týmu a výzkumného institutu, kdy všechny zúčastněné subjekty pracují na výzkumu a vývoji jednoho SW. Model je, na rozdíl od CMM[135] modelu, zaměřen na neustálé zlepšování jednoho nebo více procesů či podprocesů, díky čemuž softwarový proces a jeho výsledný SW nabývají přidaných hodnot. Za pomyslné přidané hodnoty lze považovat kvalitu výsledného SW, dodržení

[134] CMMI (Capability Maturity Model Integration) je integrační model zralosti, který je považován za standard procesů budování IS. *„CMMI zahrnuje jak referenční model softwarových procesů, tak postup posuzování úrovně způsobilosti procesů a zralosti organizace.“* (BUCHALCEVOVÁ, A. 2018)

[135] Naopak CMM model se v tomto srovnání zaměřuje na systematické (postupné) zlepšování procesů.

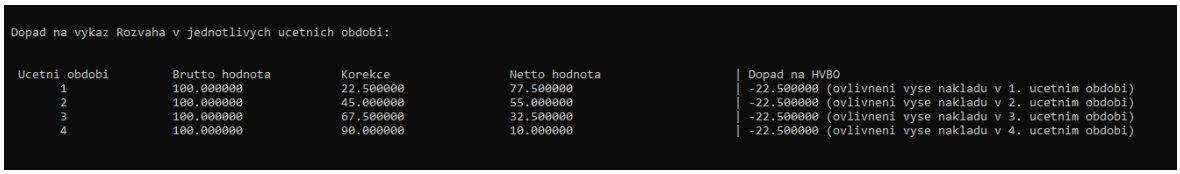
rozpočtu či termínu dodání, zefektivnění softwarového procesu či životního cyklu vývoje SW a mnoho dalších pozitivních dopadů.

Cílem podniků by měla být snaha o přiblížení se alespoň ke čtvrté úrovni zralosti organizace dle modelu CMM, která dává prostor pro technologický rozvoj firmy a pro následný zrod plně procesně řízené organizace.

Příloha 2 Audit softwaru

Účetní jednotka má v majetku firmy, mimo jiné, i stroj za 100 tis. Kč, jehož zbytková hodnota byla vyčíslena na 10 tis. Kč. Předpokládaná životnost stroje je stanovena na 3 roky. Účetní jednotka využívá pro výpočet odpisů majetku metodu lineárního odpisování. Jaké jsou odpisy v jednotlivých letech a jaká je zůstatková cena majetku v aktivech?

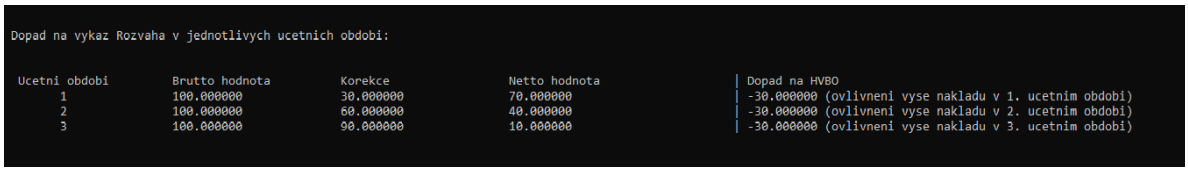
Náhled softwaru 8 Manipulativní softwarová chyba v účetním programu



Zdroj: autorka

Účetní software, který obsahuje manipulativní softwarovou úpravu, vypočetl roční odpis na 22 500 Kč. Netto hodnota stroje v majetku firmy je v prvním účetním období 77 500 Kč.

Náhled softwaru 9 Korektní výpočet účetního programu



Zdroj: autorka

Správný odpis dle zákonné úpravy je 30 000 Kč. Účetní jednotka úpravou jednoho řádku ve zdrojovém kódu prodloužila dobu odpisování o jeden rok, čímž snížila dopad na výsledek hospodaření běžného období (v programu zvaném jako Dopad na HVBO) o 7 500 Kč. Zároveň ovlivnila výši aktiv, čímž může čtenář účetní závěrky nabýt zkreslené představy o hodnotě majetku firmy.

Příloha 3 Náklady na kvalitu

Ve 40. letech 20. století Armand Feigenbaum definoval čtyři kategorie nákladů (ROUDENSKÝ, P. 2016) modelu PAF[136]:

- náklady na prevenci;
- náklady na hodnocení;
- náklady na vnitřní vady;
- náklady na vnější vady.

Náklady na prevenci jsou uvažované a v rozpočtu plánované výdaje na předcházení vzniku softwarových chyb. Za takové výdaje lze považovat náklady na školení, certifikace, řízení kvality a s tím související metodické konzultace se specialisty, audity a jiné prověrky kvality SW. Náklady na hodnocení by měly být také zahrnuty v plánovaném rozpočtu. Jedná se například o veškeré verifikační práce, revize dokumentace, inspekce či nákup a provoz kontrolních zařízení. Náklady na vnitřní vady jsou považovány za neplánované. Vývojový tým je vynakládá na softwarové chyby zjištěné před předáním softwarového projektu zadavateli. Jako příklad lze uvést náklady na korekci zdrojového kódu, konfirmační testy či náklady na analýzu a odstranění příčin. Náklady na vnitřní vady mohou mít vliv na rozpočet SW projektu i na jeho včasné dodání zadavateli. Náklady na vnější vady jsou neplánované výdaje na odstranění softwarových chyb zjištěných zadavatelem po implementaci do provozu. Patří sem nejen náklady na řízení procesu odstraňování SW chyb, ale i náklady související s odpovědnostní smluvní doložkou, ztrátu reputace či ušlý zisk.

Za další významný model vyjadřující náklady na kvalitu lze považovat koncept COPQ[137], který vyjadřuje náklady, které vznikly jako důsledek nedostatečného vývoje SW produktu. Z této logiky lze vyvodit dvě dílčí vrstvy nákladů na kvalitu[138] (ROUDENSKÝ, P. 2016):

- Náklady na soulad s požadavky (POC), které se skládají z preventivních nákladů a nákladů na hodnocení.

[136] PAF (Prevention – Appraisal – Failure, česky Prevence – Hodnocení – Vady) je model Armanda Feigenbauma, který je považován za nejstarší a nejrozšířenější.

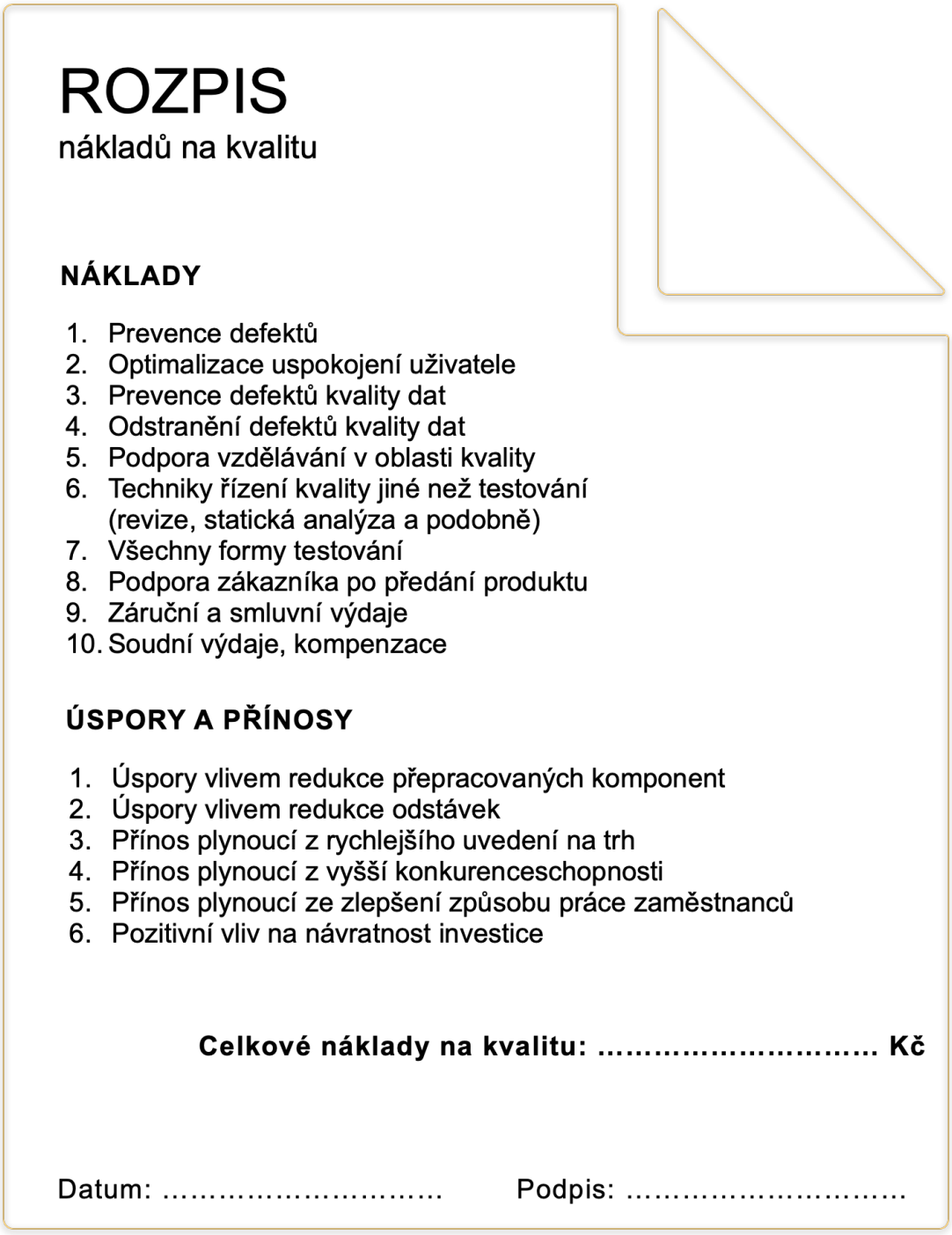
[137] COPQ (Cost of Poor Quality – Náklady na nekvalitu) je model Philipa Crosbyho, který *„náklady na kvalitu považoval za nejvhodnější způsob jejího měření.“* (ROUDENSKÝ, P. 2016)

[138] Celkové náklady na kvalitu (COQ) = POC + PONC

- Náklady nesouladu (PONC), které se skládají ze součtů interních a externích nákladů.

Uvedené položky nákladů je vhodné rozložit do vystihujících měr a obohatit je o přínosy. Výstupem je rozpis položek, který může být významným podkladem pro oceňovatele.

Obr. 23 Rozpis nákladů na kvalitu



Zdroj: autorka podle ROUDENSKÝ, P. 2016

Příloha 4 Softwarová chyba jako možná příčina kybernetického útoku

Jako příklady softwarových chyb, které mohou vést k útoku na zdrojový kód lze uvést následující příčiny (MCCLURE, S. a kol. 2007):

- přetečení zásobníku,
- přetečení haldy,
- chyby ve formátovacích řetězcích,
- chyby typu „off-by-one“,
- chyby při převodu do kanonického tvaru,
- útoky na databáze.

Přetečení zásobníku představuje celkové riziko útoku na stupni[139] 9, přičemž jde o nejjednodušší a nejnebezpečnější druh přetečení. Zásobník je jedna z částí paměti počítače, kterou počítač využívá při volání funkcí. *„Cílem hackera je pomocí přeplnění této paměti dosáhnout provedení vlastního kódu. Na zásobník se ukládají lokální proměnné funkce, parametry funkce a především návratová adresa, na kterou má program pokračovat po dokončení funkce. Pokud funkce A volá funkci B, proces musí vědět, kam se vrátit po skončení funkce B.“* (MCCLURE, S. a kol. 2007) Ochranou proti přetečení zásobníku je opatrná práce se vstupními daty získanými od uživatelů[140]. Programátor by při vývoji softwaru měl ověřovat kvalitu i kvantitu dat. Přetečení zásobníku lze předejít dodržování pravidel bezpečného programování, důkladnou revizí kódu se zaměřením na hledání nebezpečných funkcí[141] či používat nástroje, které umí na přetečení zásobníku upozornit vývojáře již během testování softwaru.

Přetečení haldy je považováno za složitější a méně častý útok, i přes to že těmto útokům podlehla celá řada softwarů. Halda ukládá dynamicky alokované proměnné bez existence návratových adres. *„Cílem útoku může být například proměnná obsahující přístupová práva, díky které se útočník dostane k citlivým souborům.*

[139] Stupnice od 0 do 10. Celkové riziko ovlivňují dílčí kritéria, mezi které patří oblíbenost, jednoduchost a nebezpečnost.

[140] Při uplatnění profesní skepse lze považovat uživatele i za potenciální útočníky.

[141] Jako například strcpy, sprintf, vsprintf, gets, strcat.

Na haldě bývají uložené i ukazatele na funkce, takže přepsáním vhodné části haldy může útočník nahradit některou ze standardních funkcí.“ (MCCLURE, S. a kol. 2007) Přetečení haldy lze zabránit pečlivou kontrolou vstupních dat, které musí být ve správném formátu a s vhodnou velikostí.

„Jediná skutečná obrana před útoky na zásobník i haldu jsou kvalitně napsané programy.“ (MCCLURE, S. a kol. 2007) Všechna dostupná opatření jsou jen pomocné nástroje, na kterých by vývojový tým neměl ochranu softwaru primárně stavět.

Chyby ve formátovacích řetězcích představují stupeň celkového rizika 7, přičemž nebezpečnost dosahuje stupně 9. Útok se snaží přepsat část paměti, *„…aby mohl hacker procesoru podstrčit své vlastní instrukce.“* (MCCLURE, S. a kol. 2007) Příčinou je například špatné užití funkce printf, sprintf. Této hrozbě lze předejít správným užitím funkcí a důslednou kontrolou vstupních dat.

Útoky díky chybám typu „off-by-one“ dosahují hodnoty rizika 7, i přes to, že jde o jeden z nejjednodušších způsobů. Jedná se o chyby, které *„vznikly přehlednutím nějaké možnosti v podmíněných výrazech“* (MCCLURE, S. a kol. 2007) a lze jim předejít pouze soustředěností programátora při vývoji SW.

Chyby při převodu do kanoického[142] tvaru (stupeň rizika 7) mohou hackera pustit až do operačního systému, ze kterého může spustit program a příkazem získat výpis adresáře.

Útoky na databáze nesou celkové riziko 8 a umožňují hackerovi zfalšovat identitu, přetížit software nebo nastavit výchozí hodnotu proměnných.

[142] *„Kanonizace je proces převodu informací do nějakého standardního, kanonického nebo též normálního tvaru. Například znak lomítko (/) lze napsat buď prostým „/“ (ASCII) nebo jako %2f (šestnáctkově).“* (MCCLURE, S. a kol. 2007)

Příloha 5 Pravidla psaní čistého kódu

Robert Cecil Martin (2009) uvádí celou řadu pravidel, které mohou dopomoci k napsání čistého kódu. Jedná se například o:

- výstižné pojmenování a vyjádření informací;
- tvorbu smysluplných rozdílů mezi informacemi;
- využití principů strukturovaného programování;
- pravidlo „Komentáře nezachrání špatný kód“.

Je nutné, aby zdrojové soubory, proměnné, funkce a třídy byly výstižně pojmenované, aby nebylo třeba vysvětlujících poznámek ke srozumitelnému přečtení zdrojového kódu a následnému pochopení významu. Vývojář by měl výstižně vyjádřit informace a neměl by používat zavádějící slova, jejichž běžně užívaný význam může být odlišný od významu, který chce slovem vyjádřit. Jako příklad lze uvést použití názvu pro předponu *hp*, která je mezi programátory známá jako název unixové platformy. Je důležité, aby vývojář dělal smysluplné rozdíly mezi informacemi. Příkladem je název první třídy *Zákazník* a název druhé třídy *Zákazník_Info*. Názvy by měly být rozlišené tak, aby čtenář v kódu zřetelně rozeznal rozdíly a dokázal přesně určit, kterou z funkcí software volá a proč. Řada vývojářů uznává principy strukturovaného programování[143], které je založeno na přiřazení jednoho vstupního a jednoho výstupního bodu v rámci každé funkce a každého bloku. V praxi to vypadá například tak, že k jedné funkci je přiřazen jeden příkaz return. Kód, který potřebuje vysvětlující komentář, aby byl srozumitelný, je špatný kód. Komentáře by měly být použity tehdy, kdy se jedná o informaci právního, informačního charakteru nebo jím chce vývojář sdělit záměr složitého nebo specifického rozhodnutí.

[143] Princip strukturovaného programování může být užitečný zejména při psaní obsáhlejších funkcí. U malých funkcí nepřináší požadovaný užitek.

Příloha 6 Přehled metod oceňování a jejich vzorců

Příloha vymezuje soubor vzorců vhodných pro výpočet hodnoty nehmotných aktiv.

Srovnávací přístup

Vzorec 2 Metoda násobitelů

$$\boldsymbol{H_{NA} = \frac{C_s}{X_s} * X_{NA}}, \tag{2}$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

C_s cena srovnatelného nehmotného aktiva,

X_s klíčová ekonomická charakteristika srovnatelného nehmotného aktiva,

X_{NA} klíčová ekonomická charakteristika sledovaného nehmotného aktiva, jehož hodnota je stanovována. Za klíčovou ekonomickou charakteristiku lze považovat například ukazatele kvantitativní (EBIT[144], EBITDA[145], Tržby nebo Obrat) nebo kvalitativní (míra inovace, posouzení časové úspory výrobního taktu nebo zpracování faktury).

Nákladový přístup

Vzorec 3 Metoda nákladů reprodukce

$$\boldsymbol{H_{NA} = \sum_{i=1}^{n} \sum_{t=0}^{T} [N_i * (1 + I_{CPI})^t * (1 + i)^t] * (1 - A) + TAB}, \tag{3}$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva;

[144] EBIT (*Earnings Before Interest and Taxes*) znamená zisk před zdaněním a před odečtením úroků. Matematicky lze EBIT vyjádřit jako součet hrubého zisku (EBT) a nákladových úroků či jako rozdíl mezi tržbami z prodeje a provozními náklady. Jedná se o druh výsledku hospodaření, který je užívaný v anglosaských zemích a díky kterému lze porovnávat výsledky hospodaření u společností s různou kapitálovou strukturou, daňovou jurisdikcí (zejména mírou zdanění) a dalšími odlišnostmi typickými pro platnou národní legislativu vedení účetnictví a účetního výkaznictví. Zdroje pro výpočet ukazatele ziskovosti podniku EBIT jsou obsažené ve *Výkazu zisku a ztráty*. Ukazatel EBIT odpovídá na otázku *„Jaká je schopnost podniku generovat (provozní) zisk ze své provozní činnosti?“*

[145] EBITDA (*Earnings Before Interest, Taxes, Depreciation and Amortization*) je zisk před úroky, daněmi, odpisy a amortizací. Vyjadřuje hrubý zisk bez režijních nákladů.

N_i hodnota nákladové položky[146], která byla vynaložena na vytvoření nehmotného aktiva;

I_{CPI} míra změny (růst či pokles) cen[147] nákladových položek mezi obdobím vynaložení nákladů (období t) a okamžikem ocenění nehmotného aktiva (období T);

t příslušný rok, ve kterém jsou měřeny aktivní položky souboru;

i náklady ušlé příležitosti;

A míra snížení užitečnosti nákladových položek (amortizace) k datu ocenění (v procentním vyjádření), pokud k zastarání došlo;

TAB přínos z daňové odepisovatelnosti aktiva[148].

Vzorec 4 Metoda nákladů nahrazení

$$H_{NA} = \sum_{i=1}^{n} N_i * (1+i)^t + TAB\ , \qquad (4)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

N_i hodnota nákladové položky[149], která byla vynaložena na vytvoření nehmotného aktiva;

i náklady ušlé příležitosti;

t příslušný rok, ve kterém jsou měřeny aktivní položky souboru;

TAB přínos z daňové odepisovatelnosti aktiva.

---

[146] Počet nákladových položek je od jedné do n. Za nákladové položky lze považovat například osobní (mzdové) náklady vynaložené během tvorby řešení i během samotného vývoje, náklady na opotřebení hmotných aktiv, průmyslovou a právní ochranu či náklady ušlých příležitostí, které mohou nabývat významných částek.

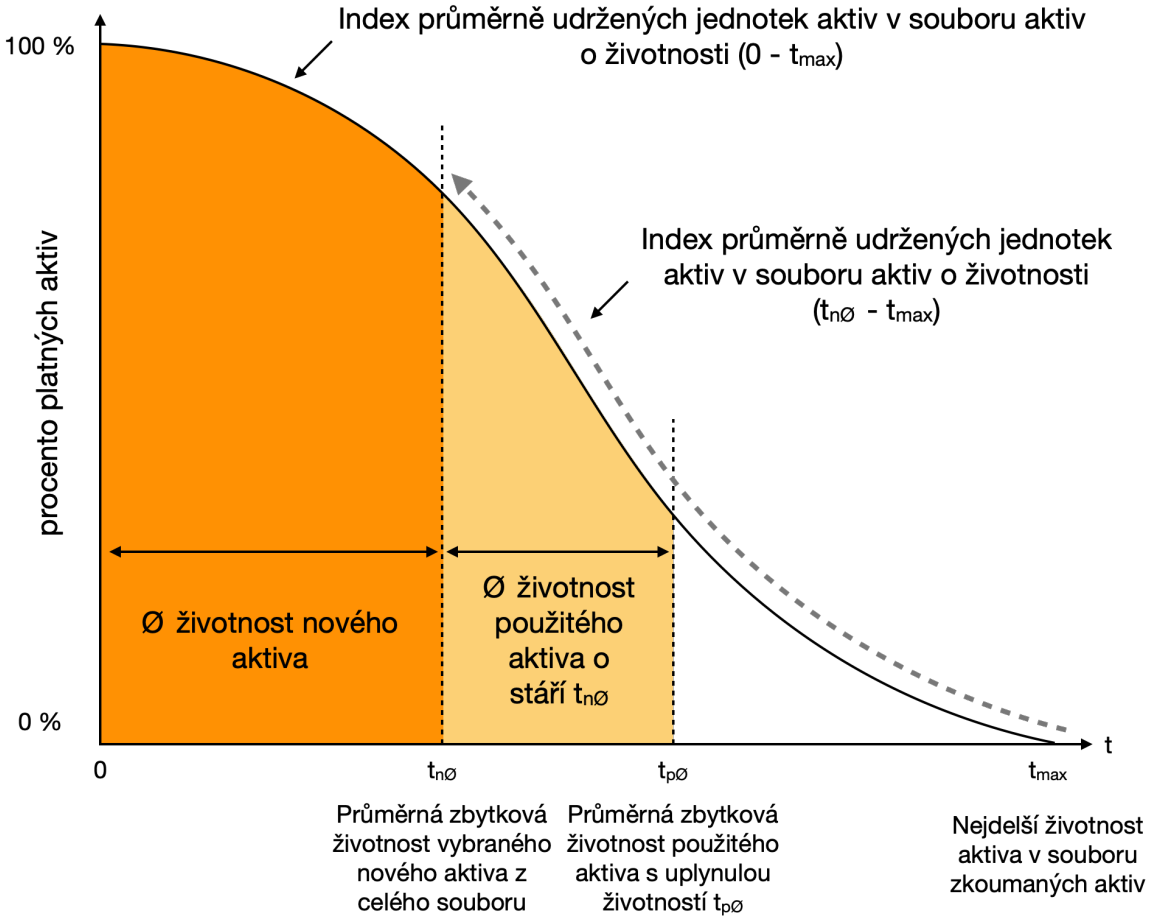
[147] Ceny nákladových položek by měly být stanovené na základě cenových indexů, mezi které patří například CPI (*Consumer Price Index*, index spotřebitelských cen) nebo PPI (*Producer Price Index*, index cen výrobců). Lze použít i jinou vhodnou veličinu, jestliže v meziobdobí došlo k významné změně cen vstupů nákladových položek a zároveň není k datu ocenění známa hodnota těchto položek.

[148] TAB – Tax Amortisation Benefit

[149] Počet nákladových položek je od jedné do n. Za nákladové položky lze považovat například osobní (mzdové) náklady vynaložené během tvorby řešení i během samotného vývoje, náklady na opotřebení hmotných aktiv, průmyslovou a právní ochranu či náklady ušlých příležitostí, které mohou nabývat významných částek.

Odhad zbývající doby použitelnosti

Obr. 24 Odhad zbývající doby životnosti



Zdroj: SVAČINA, P. 2010 (vlastní úpravy)

Vzorec 5 Odhad zbývající doby životnosti nového nehmotného aktiva

$$Ø\ t_{nové} = \frac{\int_0^{max} f\ (t)\ dt}{100} \sim \frac{\sum_{t=0}^{t_{max}} y_t}{100}, \tag{5}$$

kde proměnná

$Ø\ t_{nové}$ průměrná životnost nového nehmotného aktiva;

t příslušný rok, ve kterém jsou měřeny aktivní položky souboru;

t_{max} maximální životnost položky v souboru;

y_t počet aktivních položek v čase t.

Vzorec 6 Odhad zbývající doby životnosti užívaného nehmotného aktiva

$$Ø\ t_{použité} = \frac{\int_x^{max} f\ (t)\ dt}{y_x} \sim \frac{\sum_{t=x}^{t_{max}} y_t}{y_x}, \quad (6)$$

kde proměnná

$Ø\ t_{použité}$ průměrná životnost užívaného nehmotného aktiva;

x doba uběhlé životnosti zkoumaného aktiva;

y_x počet aktivních položek v čase x.

Výnosový přístup

Vzorec 7 Licenční analogie

$$H_{NA} = \sum_{t=1}^{n} \frac{T_t * PM * LP * K_t * (1-d)}{(1+i)^{\ t}} + TAB, \quad (7)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

T_t plán objemu prodeje výrobku obsahujícího oceňované nehmotné aktivum (čisté tržby[150]),

PM podíl nehmotného aktiva[151] (majetku) na objemu prodeje výrobku obsahujícího oceňované nehmotné aktivum (u technických řešení a designu),

LP sazba licenčního poplatku[152] (v procentech),

[150] *„Čistá prodejní cena násobená počtem prodaných jednotek výrobku obsahující nehmotné aktivum.“* (SVAČINA, P. 2010)

[151] *„Tento parametr slouží k úpravě odhadu čistých tržeb na reálnou základnu, na kterou je možné aplikovat sazbu licenčního poplatku v souladu s logikou licenčního obchodu.“* (SVAČINA, P. 2010) Pokud by k této úpravě odhadu nedošlo, pravděpodobně by bylo oceňované řešení nadhodnoceno. Příkladem může být ocenění jednoho modulu antivirového softwaru, který má 40 % podíl na prodejní ceně (celková prodejní cena za všechny moduly činí 5 000 Kč / 1 rok). Tržní licenční poplatek je stanoven na 5 % z celkové prodejní ceny. Licenční poplatek (LP) činí (výpočet: 5 000 Kč * 40 % * 5 %) 100 Kč. Bez zohlednění modulového podílu na celkovém SW řešení by byl licenční poplatek 250 Kč, tzn. došlo by k nadhodnocení o 150 Kč.

[152] Na výši licenčního poplatku má vliv celá řada cenotvorných faktorů, mezi odborníky také známé jako Georgia-Pacific faktory („G-P faktory“, celkem 15 faktorů, které mohou ovlivnit výši licenčního poplatku). Patří mezi ně například zisková marže, podíl na zisku, synergie s jinými produkty, názory expertů či výsledky řádného obchodního vyjednávání a jiné. G-P faktory jsou pojmenovány po soudním sporu Georgie Pacific vs. United States Plywood Corp. z roku 1970, ve kterém byly formalizovány.

K_t index zastarání (u technických řešení a designu),

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva,

d sazba daně z příjmů právnických osob[153],

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB).

Vzorec 8 Metoda podílu na zisku

$$H_{NA} = \sum_{t=1}^{n} \frac{T_t * ZM * PM \ * \ LP^{ZM} * K_t * (1-d)}{(1+i)^{\,t}} + TAB, \qquad (8)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

T_t plán objemu prodeje výrobku obsahujícího oceňované nehmotné aktivum (čisté tržby[154]),

ZM zisková marže[155] z prodeje výrobku obsahujícího užití nehmotného aktiva,

PM podíl nehmotného aktiva[156] (majetku) na objemu prodeje výrobku obsahujícího oceňované nehmotné aktivum (u technických řešení a designu),

---

[153] Licenční poplatky by měly být očistěny o faktor zdanění (1 – d), neboť *„teorie oceňování pracuje ve výnosových metodách obvykle s peněžním tokem po zdanění alespoň na úrovni podnikatelské jednotky, neboť hodnotu by měly tvořit jen příjmy skutečně náležející vlastníku oceňovaného aktiva, nikoliv příjmy státu, kterými jsou daně.“* (SVAČINA, P. 2010)

[154] *„Čistá prodejní cena násobená počtem prodaných jednotek výrobku obsahující nehmotné aktivum.“* (SVAČINA, P. 2010)

[155] Zisková marže by se měla vztahovat na výsledek hospodaření pouze za prodané výrobky a služby obsahující posuzované nehmotné aktivum. Příkladem může být prodej ICT techniky známé značky, která vystupuje pod zapsanou ochrannou známkou.

Vstupní data o ziskové marži z účetnictví nemusejí být vhodná, neboť poskytují provozní výsledek hospodaření podniku jako celku. Taková data mohou obsahovat například zisk (případně ztrátu) z prodeje hmotného majetku (například automobilů, strojů a jiného již nepotřebného zařízení), který by mohl zkreslit vstupní proměnnou oceňovací metody.

[156] *„Tento parametr slouží k úpravě odhadu čistých tržeb na reálnou základnu, na kterou je možné aplikovat sazbu licenčního poplatku v souladu s logikou licenčního obchodu.“* (SVAČINA, P. 2010) Pokud by k této úpravě odhadu nedošlo, pravděpodobně by bylo oceňované řešení nadhodnoceno. Příkladem může být ocenění jednoho modulu antivirového softwaru, který má 40 % podíl na prodejní ceně (celková prodejní cena za všechny moduly činí 5 000 Kč / 1 rok). Tržní licenční poplatek je stanoven na 5 % z celkové prodejní ceny. Licenční poplatek (LP) činí (výpočet: 5 000 Kč * 40 % * 5 %) 100 Kč. Bez zohlednění modulového podílu na celkovém SW řešení by byl licenční poplatek 250 Kč, tzn. došlo by k nadhodnocení o 150 Kč.

LP^{ZM} sazba licenčního poplatku vyjádřená ze zisku (v procentech),

K_t index zastarání (u technických řešení a designu),

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva,

d sazba daně z příjmů právnických osob[157],

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB).

Vzorec 9 Cenová metoda prémie

$$H_{NA} = \sum_{t=1}^{n} \frac{Q_t * \left(P_{t_s} - P_{t_{BEZ}}\right) * K_t * (1-d)}{(1+i)^t} + TAB, \quad (9)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

Q_t objem prodané produkce v měrných jednotkách,

P_{t_s} cena výrobku obsahující oceňované nehmotné aktivum,

$P_{t_{BEZ}}$ cena výrobku bez oceňovaného/srovnatelného nehmotného aktiva,

K_t index zastarání (u technických řešení a designu),

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva,

d sazba daně z příjmů právnických osob[158],

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB),

n životnost nehmotného aktiva.

[157] Licenční poplatky by měly být očistěny o faktor zdanění (1 – d), neboť *„teorie oceňování pracuje ve výnosových metodách obvykle s peněžním tokem po zdanění alespoň na úrovni podnikatelské jednotky, neboť hodnotu by měly tvořit jen příjmy skutečně náležející vlastníku oceňovaného aktiva, nikoliv příjmy státu, kterými jsou daně.“* (SVAČINA, P. 2010)

[158] Licenční poplatky by měly být očistěny o faktor zdanění (1 – d), neboť *„teorie oceňování pracuje ve výnosových metodách obvykle s peněžním tokem po zdanění alespoň na úrovni podnikatelské jednotky, neboť hodnotu by měly tvořit jen příjmy skutečně náležející vlastníku oceňovaného aktiva, nikoliv příjmy státu, kterými jsou daně.“* (SVAČINA, P. 2010)

Vzorec 10 Nákladová metoda prémie

$$H_{NA} = \sum_{t=1}^{n} \frac{Q_t * \left(N_{t_{BEZ}} - N_{t_S}\right) * K_t * (1-d)}{(1+i)^t} + TAB, \qquad (10)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

Q_t objem prodané produkce v měrných jednotkách,

$N_{t_{BEZ}}$ průměrné provozní náklady výrobku bez oceňovaného/srovnatelného nehmotného aktiva,

N_{t_S} průměrné provozní náklady výrobku obsahující oceňované nehmotné aktivum,

K_t index zastarání (u technických řešení a designu),

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva,

d sazba daně z příjmů právnických osob[159],

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB),

n životnost nehmotného aktiva.

Vzorec 11 Zisková metoda prémie

$$H_{NA} = \sum_{t=1}^{n} \frac{T_t * \left(ZM_{t_S} - ZM_{t_{BEZ}}\right) * K_t * (1-d)}{(1+i)^t} + TAB, \qquad (11)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

ZM_{t_S} zisková marže výroby obsahující oceňované nehmotné aktivum (na úrovni provozního VH (EBIT) před zdaněním),

[159] Licenční poplatky by měly být očistěny o faktor zdanění (1 – d), neboť *„teorie oceňování pracuje ve výnosových metodách obvykle s peněžním tokem po zdanění alespoň na úrovni podnikatelské jednotky, neboť hodnotu by měly tvořit jen příjmy skutečně náležející vlastníku oceňovaného aktiva, nikoliv příjmy státu, kterými jsou daně.“* (SVAČINA, P. 2010)

$ZM_{t_{BEZ}}$ zisková marže výroby bez oceňovaného/srovnatelného nehmotného aktiva (na úrovni provozního VH (EBIT) před zdaněním),

T_t tržby za výrobky obsahující oceňované nehmotné aktivum,

K_t index zastarání (u technických řešení a designu),

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva,

d sazba daně z příjmů právnických osob[160],

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB[161]).

Vzorec 12 Prémie z výnosnosti kapitálu

$$H_{NA} = \sum_{t=1}^{n} \frac{A_t * \left(ROA_{t_S} - ROA_{t_{BEZ}}\right) * K_t * (1-d)}{(1+i)^t} + TAB, \qquad (12)$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

A_t provozně potřebná aktiva podniku užívající nehmotné aktivum,

ROA_{t_S} rentabilita aktiv podniku (měřená provozním výsledkem hospodaření/EBIT před zdaněním) obsahující oceňované aktivum,

$ROA_{t_{BEZ}}$ rentabilita aktiv podniku bez oceňovaného/srovnatelného nehmotného aktiva,

K_t index zastarání (u technických řešení a designu),

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva,

d sazba daně z příjmů právnických osob[162],

[160] Licenční poplatky by měly být očistěny o faktor zdanění (1 – d), neboť *„teorie oceňování pracuje ve výnosových metodách obvykle s peněžním tokem po zdanění alespoň na úrovni podnikatelské jednotky, neboť hodnotu by měly tvořit jen příjmy skutečně náležející vlastníku oceňovaného aktiva, nikoliv příjmy státu, kterými jsou daně.“* (SVAČINA, P. 2010)

[161] TAB – Tax Amortisation Benefit

[162] Licenční poplatky by měly být očistěny o faktor zdanění (1 – d), neboť *„teorie oceňování pracuje ve výnosových metodách obvykle s peněžním tokem po zdanění alespoň na úrovni podnikatelské*

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB).

Vzorec 13 Metoda čisté současné hodnoty

$$\text{ČS}H = -I + \sum_{t=1}^{n} \frac{CF_t}{(1+i)^t}, \tag{13}$$

kde proměnná

I investiční výdaj spojený se spuštěním výroby,

CF kladné peněžní toky plynoucí z prodeje výrobků,

i náklady ušlé příležitosti (% p. a.),

t zbývající doba životnosti nehmotného aktiva.

Vzorec 14 Metoda diskontní míry pro výnosové ocenění nehmotných aktiv

$$r_{NA} = r_f + \beta * \left(r_M - r_f\right) + RPZ + \Delta\text{CPI} + r_i\,, \tag{14}$$

kde proměnná

r_f bezriziková výnosnost (přístup založení na užití výnosu do doby splatnosti určité bezrizikové investice, např. státních cenných papírů, s dobou splatnosti srovnatelnou s očekávanou zbytkovou životností oceňovaného nehmotného aktiva),

β koeficient beta, měřící úroveň systematického rizika na základě srovnání výnosů tržního portfolia a individuální akcie,

$r_M - r_f$ riziková prémie akciového trhu USA,

RPZ riziková prémie konkrétní země,

ΔCPI rozdíl v dlouhodobé inflaci mezi USA a inflací konkrétní země,

r_i ostatní přirážky k základnímu modelu (malý podnik, specifické riziko, likvidnost aktiva).

jednotky, neboť hodnotu by měly tvořit jen příjmy skutečně náležející vlastníku oceňovaného aktiva, nikoliv příjmy státu, kterými jsou daně.“ (SVAČINA, P. 2010)

Vzorec 15 Metoda nadměrných zisků

$$H_{NA} = \sum_{t=1}^{n}\sum_{i=1}^{n} \frac{[PVH_t * (1-d) - (H_i - r_i)_t}{(1+i)^t} + \text{TAB}\,, \tag{15}$$

kde proměnná

H_{NA} hodnota sledovaného nehmotného aktiva,

PVH_t provozní výsledek hospodaření před zdaněním v plánovaném roce t,

H_i hodnota i-tého aktiva užitého ve výrobě spolu s oceňovaným nehmotným aktivem,

r_i tržní nájemné po zdanění požadované za užité aktivum,

d sazba daně z příjmů právnické osoby,

i požadovaná výnosnost pro oceňované aktivum,

t zbývající doba životnosti nehmotného aktiva,

TAB přínos z daňové odepisovatelnosti nehmotného aktiva (TAB).

Vzorec 16 Vyhlášková metoda

$$C = \sum_{j=1}^{n} \frac{z_j}{(1+\frac{p}{100})}, \tag{16}$$

kde proměnná

z_j roční čistý výnos užívání práva v letech, po které bude právo užíváno,

p míra kapitalizace,

j pořadové číslo roku, ve kterém bude právo užíváno,

n počet let, ve kterých bude právo užíváno.

Příloha 7 Vývojový diagram – Detekce nespolehlivých plátců DPH

Příloha předkládá vývojový diagram softwaru sloužícího pro detekci nespolehlivých plátců DPH. Graficky popisuje, jakým způsobem je program strukturovaný, znázorňuje jednotlivé kroky softwaru a poukazuje na fáze, ve kterých jsou načítány vstupní soubory (fiktivní databáze) a generovány výstupní soubory (Přehled subjektů, Vyzývací dopisy).

Jednotlivé grafické znaky jsou spojeny spojovací čárou.

Příloha 8 Zdrojový kód – Detekce nespolehlivých plátců DPH

Příloha předkládá zdrojový kód softwaru *Detekce nespolehlivých plátců DPH*. Zdrojový kód čtenáře nejprve seznámí se základními informacemi o programu a následně s funkcionalitami hlavní funkce. Program po spuštění pracuje zcela automaticky bez ručního zásahu uživatele.

Zdrojový kód čtenáře, mimo jiné, seznamuje s přesným zněním vyzývacího dopisu pro detekované subjekty. Mimo zdrojový kód lze vyzývací dopisy v čitelné (standardní) formě vidět v příloze 9 této diplomové práce.

Vyzývací dopis má dvě znění. Jedno znění je zaměřené na subjekty, které jsou v Registru evidovány jako nespolehlivé. Druhé znění dopisu je určeno pro subjekty, jejichž bankovní spojení používané při obchodních transakcích neodpovídá zveřejněnému bankovnímu účtu v Registru.

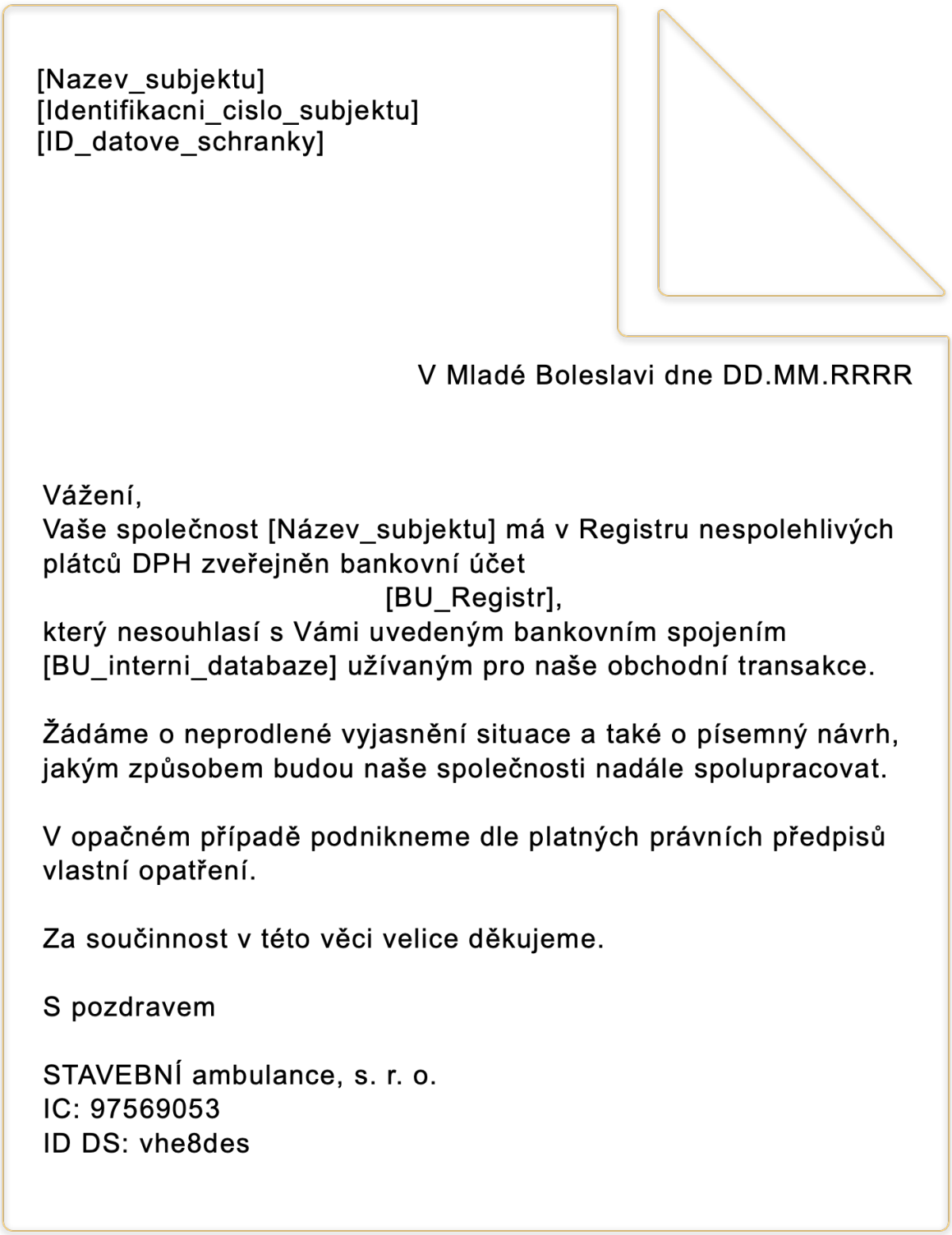
Zdrojový kód má 189 řádků.

Spustitelný prototyp SW lze nalézt spolu s vývojovým diagramem i všemi vstupními soubory na přiloženém CD.

Příloha 9 Náhled dopisu vyzývajícího detekované subjekty k objasnění situace

Příloha předkládá náhled vyzývacích dopisů. V hranatých závorkách ([ ]) jsou názvy proměnných užívaných ve zdrojovém kódu.

Obr. 25 Náhled vyzývacího dopisu pro plátce s odlišným bankovním účtem



[Nazev_subjektu]
[Identifikacni_cislo_subjektu]
[ID_datove_schranky]

V Mladé Boleslavi dne DD.MM.RRRR

Vážení,
Vaše společnost [Název_subjektu] má v Registru nespolehlivých plátců DPH zveřejněn bankovní účet
[BU_Registr],
který nesouhlasí s Vámi uvedeným bankovním spojením [BU_interni_databaze] užívaným pro naše obchodní transakce.

Žádáme o neprodlené vyjasnění situace a také o písemný návrh, jakým způsobem budou naše společnosti nadále spolupracovat.

V opačném případě podnikneme dle platných právních předpisů vlastní opatření.

Za součinnost v této věci velice děkujeme.

S pozdravem

STAVEBNÍ ambulance, s. r. o.
IC: 97569053
ID DS: vhe8des

Zdroj: autorka

Obr. 26 Náhled vyzývacího dopisu pro nespolehlivého plátce

[Nazev_subjektu]
[Identifikacni_cislo_subjektu]
[ID_datove_schranky]

V Mladé Boleslavi dne DD.MM.RRRR

Vážení,
Vaše společnost [Název_subjektu] je zveřejněna v Registru nespolehlivých plátců DPH jako [Platcovstvi] plátce.

Žádáme o neprodlené vyjasnění situace a také o písemný návrh, jakým způsobem budou naše společnosti nadále spolupracovat.

V opačném případě podnikneme dle platných právních předpisů vlastní opatření.

Za součinnost v této věci velice děkujeme.

S pozdravem

STAVEBNÍ ambulance, s. r. o.
IC: 97569053
ID DS: vhe8des

Zdroj: autorka

Příloha 10 Vývojový diagram – Oceňování SW

Příloha předkládá vývojový diagram softwaru sloužícího pro oceňování softwarů a softwarových projektů. Graficky znázorňuje jednotlivé kroky softwaru a popisuje, jakým způsobem je program strukturovaný.

Vývojový diagram obsahuje tři několikanásobné alternativy (větvení). Uživatel je vyzván, aby zadal číslo z hlavního menu a z menu druhé a třetí úrovně. Po zadání čísla se provede algoritmus skrytý pod konkrétním číslem. Pro několikanásobné alternativy byla použita grafická značka podmíněného výrazu.

Pro zjednodušení rozsahu vývojového diagramu byla použita množina grafické značky pro ruční vstup uživatele (Vložení uživatelských hodnot). V softwaru je uživatel vyzván, aby zadal jednotlivé proměnné. Počet vstupních proměnných se různí vzhledem k vybrané metodě oceňování.

Výpočet jednotlivých úloh provádí uživatelská funkce, která je ve vývojovém diagramu znázorněna pomocí grafické značky podprogramu. Výsledek výpočtu je zobrazen uživateli po provedení úlohy pomocí grafické značky zobrazení výstupu.

Každý výstup je pomocí běžného příkazu zapsán do výsledkové tabulky.

Program obsahuje návratovou funkci, která umožňuje uživateli vrátit se na hlavní menu po stisknutí libovolné klávesy. Ve vývojovém diagramu je tato funkce znázorněna pomocí spojovací značky.

Jednotlivé grafické znaky jsou spojeny spojovací čárou.

Příloha 11 Zdrojový kód – Oceňování SW

Příloha předkládá zdrojový kód programu Oceňování SW. Zdrojový kód čtenáře nejprve seznámí se základními informacemi o programu a s uživatelskými funkcemi (podprogramy), následuje hlavní funkce. Hlavní funkce obsahuje strukturu několikanásobné alternativy (větvení) hlavního menu a menu druhé a třetí úrovně. V jednotlivých větvích jsou prováděny ruční vstupy uživatele (zadání uživatelských hodnot proměnných) a úlohy uživatelských funkcí. Po provedení úkonu se uživatel z větve dostává na hlavní menu.

Hlavní funkce obsahuje také strukturu výsledkové tabulky, ve které jsou prováděny propočty u všech, uživatelem vybraných, oceňovacích metod.

Zdrojový kód, mimo jiné, obsahuje formátování textu (například *\033[36m* pro tyrkysově modrou, *\033[35m* pro fialovou nebo *\033[32m* pro zelenou *a \033[0m* pro ukončení veškerého předchozího formátování.

Zdrojový kód obsahuje 1 247 řádků.

Spustitelný prototyp SW lze nalézt spolu s vývojovým diagramem na přiloženém CD.

Příloha 12 Ilustrace Ohodnocení softwarového procesu

Ilustrace *Ohodnocení softwarového procesu* se skládá z 12 schémat, které na sebe navazují a které tvoří souvislý softwarový proces (viz kapitola Softwarový proces a Životní cyklus vývoje). Každé z 12 schémat je věnováno právě jedné fázi softwarového procesu.

Každá dílčí fáze obsahuje zjednodušenou grafickou vizualizaci aktivit vývojového týmu, zadavatele a dalších specializovaných osob (pětihranný obrazec), jejich písemný výstup (dokument) a příklady otázek, díky kterým může oceňovatel dojít k mnoha úvahám o posuzovaném SW a následně i k ohodnocení dílčího kroku softwarového procesu. Ohodnocení, které nabývá hodnot 0–10, oceňovatel zadává v programu Oceňování SW. Výstupem ohodnocení je koeficient, který ovlivňuje výslednou výši ocenění SW.

ANOTAČNÍ ZÁZNAM

AUTOR	**Ing. Jana Jiránková**		
STUDIJNÍ PROGRAM/OBOR/SPECIALIZACE	**Podniková ekonomika a manažerská informatika**		
NÁZEV PRÁCE	**Vývoj a oceňování specializovaného softwaru**		
VEDOUCÍ PRÁCE	**Ing. Vladimír Beneš, Ph.D.**		
KATEDRA	**KI - Katedra informatiky**	**ROK ODEVZDÁNÍ**	**2023**

POČET STRAN	**84**
POČET OBRÁZKŮ	**26**
POČET TABULEK	**0**
POČET PŘÍLOH	**12**

STRUČNÝ POPIS	Diplomová práce je zaměřena na problematiku vývoje specializovaného softwaru jako nástroje pro dosažení podnikatelských cílů v digitální a informační éře. Hlavní pozornost je soustředěna také na ocenění softwarového projektu se zohledněním specifických vlastností softwaru. Cílem diplomové práce je vysvětlit význam vývoje softwaru, volby vhodné metodiky a modelu vývoje, životního cyklu i metody ocenění finálního softwarového projektu. Praktická část se věnuje vývoji dvou specializovaných softwarových prototypů, jejich analýze při definování požadavků a na vizualizaci návrhu SW prostřednictvím vývojových diagramů. Součástí praktické části jsou zdrojové kódy obou softwarových prototypů.
KLÍČOVÁ SLOVA	Vývoj softwaru, Softwarový proces, Kvalita softwaru, Životní cyklus, Oceňování softwaru, Hodnota softwaru, Umělá inteligence, IT Audit, Projektové řízení, Metodiky vývoje softwaru, Umění čistého zdrojového kódu, Kybernetická bezpečnost, Kybernetický útok

ANNOTATION

AUTHOR	**Ing. Jana Jiránková**		
FIELD	**Business Informatics**		
THESIS TITLE	**Development and Valuation of specialized Software**		
SUPERVISOR	**Ing. Vladimír Beneš, Ph.D.**		
DEPARTMENT	**KI - Department of Informatics**	**YEAR**	**2023**

NUMBER OF PAGES	**84**
NUMBER OF PICTURES	**26**
NUMBER OF TABLES	**0**
NUMBER OF APPENDICES	**12**

SUMMARY	The Diploma thesis is focused on the Development of specialized software as a tool for achieving Business goals in the digital and information era. The main attention is also focused on the evaluation of the software project considering to the account of the specific features of the Software. The aim of the thesis is to explain the importance of software development, the choice of a suitable Methodology and Development model, the Life Cycle of Software development and Methods of evaluating of the final software project. The practical part is dedicated to the development of two specialized Software prototypes, their analysis while defining requirements and the visualization of the SW design through flowcharts. The practical part includes the Source codes of both Software prototypes.
KEY WORDS	Software development, Software process, Software quality, Life Cycle of software development, Valuation of Software, Software value, Artificial Intelligence, IT Audit, Project management, Software development methodologies, The Art of Clean Source Code, Cyber Security, Cyber Attack