Univerzita Pardubice
Fakulta ekonomicko-správní
Ústav systémového inženýrství a informatiky

Analýza komplexity v informačních systémech

Bc. Vlastimil Pařízek

Diplomová práce
2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Vlastimil Pařízek**
Osobní číslo:	**E16678**
Studijní program:	**N6209 Systémové inženýrství a informatika**
Studijní obor:	**Informatika ve veřejné správě**
Název tématu:	**Analýza komplexity v informačních systémech**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování:

Cílem práce je sumarizovat příčiny komplexity včetně jejích dopadů na informační systém ve všech fázích a dimenzích životního cyklu a nastínit využití vybraných metrik pro měření komplexity během vývoje a provozu informačního systému.

Osnova práce:
- Pojem informační systém a jeho druhy.
- Komplexita informačních systémů.
- Dopady vysoké a nízké komplexity dle úrovně abstrakce a fáze životního cyklu.
- Aplikace vybraných metrik komplexity na reálném informačním systému.
- Zhodnocení získaných výsledků.

Rozsah grafických prací:

Rozsah pracovní zprávy: **cca. 60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

BASL, Josef. Podnikové informační systémy: podnik v informační společnosti. 2., výrazně přeprac. a rozš. vyd. Praha: Grada, 2008. Management v informační společnosti. ISBN 8024722798.
BRUCKNER, Tomáš. Tvorba informačních systémů: principy, metodiky, architektury. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.
Xia, W., & Lee, G. (2003). Complexity of Information Systems Development Projects: Conceptualization and Measurement Development.
J. Manage. Inf. Syst., 22(1), 45-83.
LEMBERGER, Pirmin a Médéric MOREL. Managing complexity of information systems: the value of simplicity. Hoboken, NJ: Wiley, 2011, s. 17-145. ISBN 978-1-84821-341-8.

Vedoucí diplomové práce: **Ing. Martin Ibl, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání diplomové práce: **1. září 2017**

Termín odevzdání diplomové práce: **30. dubna 2018**

doc. Ing. Romana Provazníková, Ph.D.
děkanka

L.S.

doc. Ing. Pavel Petr, Ph.D.
vedoucí ústavu

V Pardubicích dne 1. září 2017

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako Školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47 b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 30. 4. 2018 Bc. Vlastimil Pařízek

PODĚKOVÁNÍ:

Tímto bych rád poděkoval svému vedoucímu práce Ing. Martinu Iblovi, Ph.D za jeho odbornou pomoc, cenné rady a poskytnuté materiály, které mi pomohly při zpracování diplomové práce. Dále děkuji své rodině za podporu, které se mi dostávalo po celou dobu studia.

ANOTACE

Diplomová práce se věnuje analýze komplexity v informačních systémech, její příčině a důsledkům pro uživatele a organizace. V práci je vymezen pojem informační systém a jeho životní cyklus. Dále jsou uvedeny metriky pro měření komplexity během vývoje a provozu informačního systému. Metriky jsou následně aplikovány na reálném systému. Na závěr jsou zhodnoceny získané výsledky i výstupy z celé práce.

KLÍČOVÁ SLOVA

komplexita, informační systém, životní cyklus, analýza komplexity, měření komplexity

TITLE

Analysis of complexity in information systems

ANNOTATION

This diploma thesis deals with the complexity analysis of information systems, its causes and consequences for users and organizations. The concept of information system and its life cycle is defined in the thesis. In addition, metrics are used to measure complexity during the development and operation of the information system. The metrics are then applied to the real system. Finally, the obtained results and outputs from the whole work are evaluated.

KEYWORDS

complexity, information system, life cycle, analysis of complexity, measurement of complexity

OBSAH

SEZNAM TABULEK

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

DC	Database Complexity
DSS	Decision Support Systém
EIS	Executive Information Systém
EPC	Event-driven Proces Chain
ES	Expert System
ERP	Enterprise Resource Planning
ERD	Entity Relationship Diagram
HW	Hardware
ICT	Information and Communication Technologies
IDEF	Integration Definition
IS	Informační Systém
MMDIS	Multidimensional Management and Development of Information Systems
MIS	Management Information Systém
NF	Normální Forma
SŘBD	Systém řízení báze dat
SIS	Strategic Information Systém
SW	Software
RMD	Relational Data Model
TPS	Transaction Processing System
UML	Unified Modeling Language
CFC	Control-flow Complexity
MCC	McCabe Complexity

ÚVOD

Komplexita je pojem dnes často používán v odborných článcích, metodikách a standardech používaných k řízení informatiky. Právě komplexitou v informačních systémech se zabývá tato práce. V rámci této diplomové práce je na komplexitu v informačních systémech pohlíženo jako na vlastnost systému, která ovlivňuje jeho přehlednost, obtížnost implementace, modifikovatelnost, pochopitelnost nebo použitelnost.

Komplexita informačního systému přímo ovlivňuje náklady na jeho pořízení, provoz i údržbu. Minimalizace těchto nákladů je hlavním důvodem, proč je třeba komplexitu sledovat a řídit. V každé fázi životního cyklu je třeba se zaměřit na prvky složitosti systému z jiné úrovně a věnovat maximální úsilí optimalizaci, aby do další fáze nebyla nadbytečná složitost přenesena. Nadbytečná komplexita ztěžuje práci analytikům, programátorům i uživatelům a zároveň snižuje spolehlivost systému.

Cílem této práce je sumarizovat příčiny komplexity včetně jejích dopadů na informační systém ve všech fázích a dimenzích životního cyklu a nastínit využití vybraných metrik pro měření komplexity během vývoje a provozu informačního systému.

Tato práce napomůže čtenáři zorientovat se v problematice posouzení komplexity na různých úrovních abstrakce v rámci životního cyklu IS. Navíc práce nastíní na příkladu aplikovatelnost vybraných metrik řízení komplexity systému v rámci všech jeho fází. S využitím vybraných metrik budou kvantifikovány jednotlivé kroky vývoje. Použití těchto metrik lze využít jako podpora při výběru optimální varianty, a to jak pro vývojáře, tak pro zájemce o nový informační systém.

V práci bude nejprve definován pojem informační systém a popsány jeho druhy dle využití v organizacích. Další kapitola bude věnována pojmu komplexita, a to nejprve v kontextu informační teorie a následně ve vztahu na informační systémy a aplikace. Pro zkoumání komplexity a jejich příčin v jednotlivých fázích životního cyklu a všech obsahových dimenzích bude využito členění dle metodiky MMDIS, popis fází a dimenzí bude obsahovat druhá kapitole této práce. U všech fází i dimenzí bude definován jejich vztah ke komplexitě informačního systému a budou sumarizovány příčiny a dopady vysoké komplexity. Dále budou shrnuty zdroje vzniku komplexity v tak složitých systémech jako jsou podniky. Dopady vysoké komplexity a výhody nízké složitosti v jednotlivých úrovních abstrakce bude sumarizovat třetí kapitola. V práci budou popsány metriky sloužící pro kvantifikaci a následné posuzování

komplexity v daných fázích vývoje informačního systému. Tyto metriky budou použity v poslední kapitole, kde aplikovány na vybraném systému.

1. POJEM INFORMAČNÍ SYSTÉM A JEHO DRUHY

Tato kapitola bude věnována informačním systémům. Nejprve bude vysvětlen pojem informační systém a dále budou popsány základní druhy infomačních systémů dle jejich postavení v systému řízení podniku.

1.1. Informační systém

Jednoznačně definovat pojem informační systém není snadné, a proto neexistuje žádná jednoznačná definice, protože každý uživatel nebo tvůrce informačního systému používá různé terminologie a zdůrazňuje jiné aspekty. Lze však tvrdit, že informační systém lze popsat jako vzájemné propojení uživatelů, hardwaru, softwaru, informací a procesů, které s těmito informacemi pracují. Pod pojmem procesy rozumíme funkce, které zpracovávají informace, které do systému vstupují a transformují je na informace, které ze systému vystupují. Procesy se tedy rozumí funkce zabezpečující sběr, přenos, uložení, zpracování a distribuci informací.

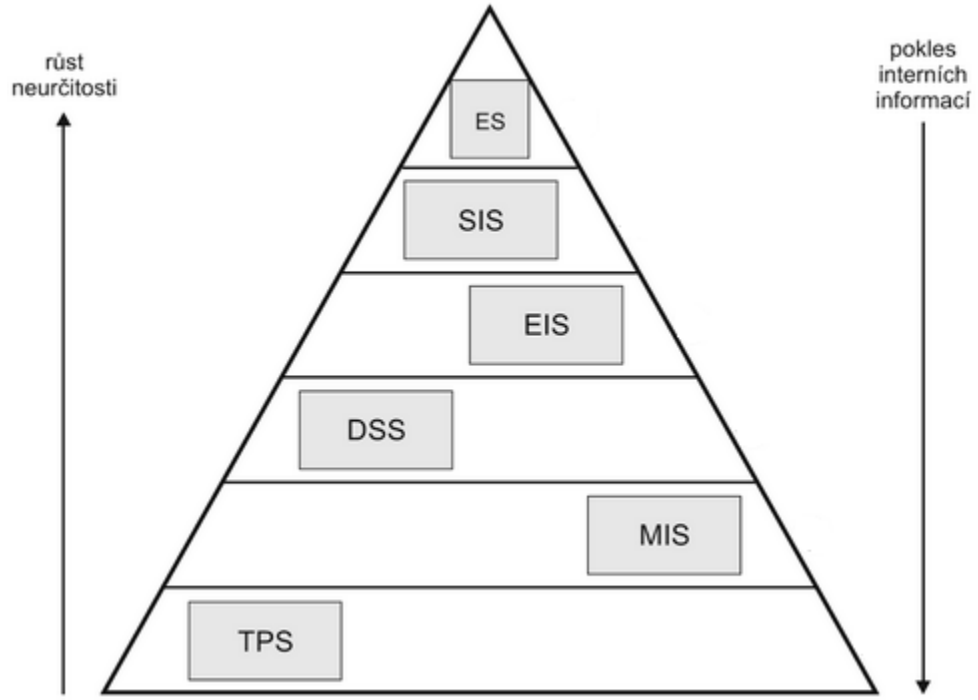
Jako každý systém má i informační své okolí. Okolí informačního systému tvoří veškeré objekty, které nějak ovlivňují samotný systém, a také objekty, které jsou systémem ovlivněny.

Celkově lze říci, že IS je softwarové vybavení, které je schopné na základě zpracovávaných informací řídit procesy. Dále je schopen tyto informace poskytovat řídícím pracovníkům tak, aby jim byl nápomocný zejména při plánování, koordinaci a kontrole veškerých procesů, a to jak v soukromém, tak veřejném sektoru. [28]

1.2. Druhy IS

Informační systémy je možno rozdělit do kategorií dle různých hledisek. Hlavním kritériem je účel využití systému. Dále lze systémy rozlišit dle obsahu, velikosti, strukturální složitosti, počtu a typu uživatelů a územního rozsahu. Pro volbu metod a přístupů k řízení IS je rozhodující především vztah IS k účelu jeho využití, zejména ve vztah k systému řízení podniku. Z hlediska postavení IS v systému řízení rozlišujeme především to, v jakém stupni informační pyramidy se nachází. Informační pyramida je na Obrázek 1.

Čím výše se systém v pyramidě nachází, tím více přibývá neurčitost v požadavcích na informační systém, ale také ubývá objem interních informací. Díky tomuto přibývá potřeba externích informací z podstatného okolí systému. [7]



Obrázek 1: Informační pyramida

Zdroj: upraveno dle [14]

1.2.1. Transakční systémy (TPS)

Transakční systémy mají za cíl mechanizovat typické agendové úlohy, jako jsou mzdy, fakturace nebo skladové hospodářství. Uživatelé transakčních systémů jsou vysoce kvalifikovaní pracovníci schopní provádět samostatná rozhodnutí v zájmu podnikových cílů. Uživatelské rozhraní těchto systému je poměrně komplexní proto, je kladen důraz na kvalitu uživatelů, kteří se systémem denně pracují. [8]

1.2.2. Systémy pro řízení (MIS)

Jedná se o systémy vycházející z účetních a ekonomických systémů. Jejich hlavní funkcí je zpřístupnění různých přehledů nebo tvorba sestav, čímž usnadňují práci řídícím pracovníkům, hlavně při kontrole výkonnosti. Typické jsou detailní přehledy o provozu některých dílen, provozů, závodů i celých podniků. [8]

1.2.3. Systémy pro podporu rozhodování (DSS)

Systémy podpory rozhodování jsou informační systémy, které podporují podnikové nebo organizační rozhodovací činnosti. Tyto systémy mohou být využity na všech úrovních řízení podniku, ale je zpravidla používán středním a vyšším managementem. Pomáhají lidem rozhodovat o problémech, které se mohou rychle měnit a nejsou předem jednoznačně specifikovány.

Pod tyto systémy někteří autoři zahrnují všechny systémy, které by mohly podporovat rozhodování. Správná definice je dle [27] takováto:

- DSS májí tendenci být zaměřeny na méně strukturovaný, nespecifikovaný problém, který obvykle čelí manažeři vyšší úrovně, pokouší se kombinovat použití modelů nebo analytických technik s tradičními datovými funkcemi a funkcemi vyhledávání;
- DSS obsahují funkce, které jsou snadno použitelné i pro uživatele, který nemá počítačové vzdělání;
- DSS se zaměřují na flexibilitu a přizpůsobivost ke změnám v přístupu k rozhodování uživatele. Některé DSS zahrnují také znalostní systémy.

Správně navržený systém DSS je interaktivní softwarový prostředek určený k tomu, aby pomáhal rozhodujícím osobám sestavovat užitečné informace z kombinace prvotních dat, dokumentů a osobních znalostí nebo obchodních modelů, k identifikaci a řešení problémů a přijímání rozhodnutí.

1.2.4. Informační systémy pro vrcholové řízení (EIS)

EIS jsou aplikace účelově orientované na potřeby podpory vedení podniků a institucí. Využívají všech dostupných informačních zdrojů vytvářených na nižších úrovních informačního systému, tj. úlohami transakčního charakteru, úlohami pro taktické a operativní řízení a úlohami pro podporu rozhodování. [29]

1.2.5. Strategické informační systémy (SIS)

SIS se snaží se o zvýšení konkurenceschopnosti podniku. Jsou přímo spjaty s výrobou nebo výrobkem. V průmyslové oblasti čerpají informace třeba z automatizovaných výrobních linek a v obchodu například z elektronické pošty.

1.2.6. Prognostické expertní systémy (ES)

Tyto systémy jsou vytvořeny z nástrojů, které dovolují provádět analýzy, a které odpovídají na otázky typu „CO KDYŽ?“ a umějí vytvářet prognózy. Do této kategorie spadají také expertní systémy (ES). Tyto systémy jsou založeny na množině pravidel, která pomáhají nepříliš zkušenému a znalému pracovníkovi řešit úlohy často diagnostického charakteru. Hlavním cílem zavedení expertního systému je poskytnout znalosti, které vlastní pouze několik expertů v oboru, více pracovníkům. Tyto systémy využívají technologie z oblasti umělé inteligence v podobě soustav pravidel produkčního typu uložených v tzv. bázi znalostí (soustava IF-THEN pravidel).

2. KOMPLEXITA INFORMAČNÍCH SYSTÉMŮ

V následující kapitole bude vysvětlen pojem komplexita. Nejdříve bude komplexita analyzována v kontextu informační teorie a následně v kontextu informačního systému. Komplexita informačního systému bude zkoumána z pohledu různých obsahových dimenzí a fází životního cyklu. Dále bude objasněny zdroje komplexity informačního systému. Pro vybrané dimenze budou uvedeny metody kvantifikace komplexity.

2.1. Komplexita

Komplexitou je možno zjednodušeně nazvat komplikovanost či složitost nějakého systému. Komplexní systém jako celek vykazuje jednu nebo více charakteristických vlastností, které však nejsou jasně viditelné z vlastností jednotlivých částí systému. Teorie komplexity vychází z postulátu „celek je víc než suma jeho částí“. U komplexních systémů se vyskytuje tzv. emergence, což je vlastnost komplexních systémů, kdy v celkovém chování lze odhalit komplexní vzory vznikající z jednodušších pravidel. Pod pojmem vzor je myšleno zejména podobnost v chování. U komplexního systému nejde o počet prvků, ale o složitost jejich propojení, hustotu sítě. Důsledkem rostoucí komplexity je větší složitost systému, která způsobuje například jeho horší čitelnost, srozumitelnost či použitelnost. [18]

Komplexita se vztahuje k implicitní povaze systému, která se dotýká jak vlastností interagujících komponent, tak povahy jejich interakce. Komplexita systému zahrnuje aspekty jako je neurčitost, fluktuace, singularita, vnitřní dynamika, konektivita a další.

Druhy rozeznávané komplexity:

- Deskriptivní komplexita – odpovídá tomu, jak se systém jeví pozorovateli, který stojí mimo něj. Nabízí obvykle statické pohledy na systém.
- Přirozená komplexita – je vztažena k vnitřku systému, vystihuje vnitřní podstatu systému a vystihuje i jeho dynamiku.

Ve výše zmíněných druzích komplexity je zakotvena informace nutná k popsání systému, tak i informace k objasnění neurčitosti. V praxi se obě komplexity dostávají do konfliktu. Jestliže chceme omezit jednu z nich, pak druhá pravděpodobně roste nebo v nejlepším případě zůstává stejná. Tato vzájemná výměna je jedním z nejvýznamnějších metodologických východisek systémové vědy.

Aspekty ovlivňující komplexitu každého systému jsou zejména:

• množství prvků a vazeb mezi nimi;

• rekurzivní vztahy (systém nezávisí na jen na vstupních podnětech, ale i na minulém vývoji);

• paralelní procesy a jejich vzájemná synchronizace;

• interakce (vzájemné ovlivňování procesů a přítomnost efektu motýlích křídel)[1];

• neurčitost a fluktuace (všechny systémy udržují určité vzory). [19]

Předmětem této práce je komplexita v kontextu informačních systému. Informační systémy automatizují podnikové procesy, a proto je třeba se zaměřit na komplexitu procesů, které mají být automatizovány, neboť procesy s velkou komplexitou, které je těžké pochopit nelze dobře zavézt do systému a způsobují velké problémy při návrhu systému. Každý podnikový proces je sled vzájemně na sebe navazujících činností, které probíhají napříč organizačními jednotkami a reagují na různé podněty z vnitřního i vnějšího prostředí. Proces obstarává transformaci vstupů na výstupy za pomocí nějakých zdrojů. [2]

Komplexita podnikových procesů musí být neustále pod kontrolou, aby nedocházelo k jejímu zvyšování v čase, což by mohlo vézt k chybám a ztěžovalo by jejich pochopení. Míru složitosti lze odvodit dle toho, jak obtížné je daný proces pochopit nebo vysvětlit. Komplexita procesů není konstantní. Tak jak se proces v čase vyvíjí, mění se i jeho komplexita.

2.2. Určení míry komplexity

Určení míry komplexity není jednoduché. Nejzákladnější možnost pro posouzení komplexity je hodnotit systém dle jeho velikosti. Tento přístup však dle vlastností komplexity uvedených výše není vyhovující, neboť velikost nemusí být přímo spojena se složitostí. Dalším hlediskem může být entropie, kde však musí být zkoumána nějaká forma zprávy a nejvyšší hodnotu má náhodný systém. Lze také měřit algoritmický informační obsah. Tato metoda souvisí s teorií složitosti a slouží pro klasifikaci výpočetních systémů. Zde se rozlišuje algoritmická komplexita, jako nejkratší délka programu popisující nějaký objekt a efektivní komplexita, kterou pro nějakou entitu definuje Murray Gell–Mann jako délku vysoce komprimovaného popisu její pravidelnosti. V této metodě se jedná o snahu oddělit komplexitu a náhodnost. Tento typ komplexity je také označován jako Kolmogorova-Chaitinova komplexita. Komplexitu lze také měřit pomocí logické hloubky, která určuje, jak je těžké objekt

[1] Efekt motýlích křídel – vyjadřuje citlivou závislost vývoje systému na počátečních podmínkách, jejichž malé změny mohou mít za následek velké variace v delším průběhu

sestrojit a termodynamické hloubky, což je sekvence událostí vedoucí k objektu, včetně zdrojů vyžadovaný ke konstrukci systému. Další metodou je statistická komplexita, která je založena na určení minima informace nutné k předpovědi budoucnosti systému. Dále jsou na měření komplexity používané metody založené na teorii fraktální dimenze či stupně hierarchie.[18] Blíže budou objasněny tři výše zmíněné přístupy ke komplexitě, a to: Shannonova entropie, K - komplexita a logická hloubka dle Charlese H. Bennetta.

Předpokládejme, že každý objekt nebo systém, jehož složitost chceme definovat, je popsán binárními sekvencemi (například 01001101). Shannonova entropie, K-komplexita a logická hloubka mají jedno společné a to, že všechny vyhodnocují složitost systému jako množství informace, které obsahuje jeho popis. Dále předpokládejme, že chceme přiřadit složitost ke kompilovanému kód algoritmu, který řeší určité problémy. Systémem, je zde nějaký software a binární sekvencí je jeho kompilovaný kód. Cílem je dosáhnout při sestavování zdrojového kódu toho, aby fungoval co nejrychleji a byl co nejkratší. Velmi důležitý je také počet problémů, které algoritmus dokáže vyřešit.[16]

2.2.1. Shannonova entropie

Jednoduše si lze tento pojem představit jako míru neurčitosti nebo náhodnosti přítomné v systému. Toto množství náhodnosti má vliv na průměrné množství informací a také na představu o složitosti systému. Východiskem je, že zdrojem informací je text, který se má předat, který si lze představit jako náhodný generátor znaků, který je tvořen náhodnou sekvencí znaků, jejichž pravděpodobnosti jsou známy (znak "a" se objeví častěji než "x"). Cílem je kódování řetězců znaků, jejichž pravděpodobnosti jsou známy tak, aby jejich průměrná zakódovaná délka byla co nejkratší, z čehož vyplývá, že často se vyskytující znaky mají být zakódovány krátkými řetězci. [16]

Výsledkem je, že průměrná délka kódu je větší, pokud je zdroj náhodnější a průměrná délka optimálního řešení je závislá na pravděpodobnosti výskytu každého znaku, a nikoliv na přesné znakové sadě nebo abecedě, která se používá.

Vztah Shannonovy entropie k množství informace lze odvodit tak, že v případě, kdy je neurčitost nízká a víme předem, jaký bude výsledek, nezískáme mnoho informací. V případě, že zdroj je vysoce nepředvídatelný, získáme spoustu informací, jakmile dostaneme výsledek, který jsme nemohli předem odhadnout. Důležitou vlastností Shannonovy entropie je, že ji lze spočítat, jakmile známe pravděpodobnosti náhodných událostí. Lze tedy shrnout, že entropie měří formu složitosti související s nepředvídatelností systému.

2.2.2. Kolmogorova složitost

V případě, že chceme přiřadit složitost jednomu objektu nebo systému. Lze využít tzv. Kolmogorovu složitost dále jen K-složitost. K-složitost je jedním možným měřítkem množství informací obsažených v binárním řetězci. Binární řetězec může být například popis objektu, posloupnost znaků knihy nebo soubor schémat UML. Tuto binární sekvenci je třeba komprimovat co nejvíce je to možné a následně je zjištěna K-složitost řetězce jako délka jeho nejvíce komprimované formy. Optimální komprimovaná forma binárního řetězce není nic jiného než nejkratší program, který při spuštění, dává zpátky náš původní řetězec. Pokud má řetězec v sobě velké množství struktur, například pokud existuje vzorec, který se pravidelně opakuje jako 011011011…, potom program, který vygeneruje tento řetězec, může být mnohem kratší než původní řetězec. Na druhou stranu, když je řetězec "náhodný" a nemá v sobě žádnou strukturu, pak pouze explicitní výstupní operací lze vytvořit původní řetězec. Žádná komprese není možná a délka tohoto výstupního operačního programu je delší než délka původního řetězce. Nezávislost velikosti řetězce na skutečné délce programu, tak vypovídá o tom, že K - složitost jako skutečné měřítko složitosti.[16]

Při definovaní K-složitosti objektu nebo systému, jako složitost jeho popisu v binární formě, existuje nejednoznačnost, protože většina fyzických objektů nebo systémů má různé úrovně popisu. Popsat IS z nejvyšší úrovně lze například pomocí případů užití (use case diagram). V nižší úrovni je možno brát v úvahu zdrojový kód aplikací jako další platný popis a v nejnižší úrovni vlastní binární kód na každém čipu. Každý z těchto popisů obsahuje velmi odlišné množství informací, a proto je třeba vybrat jednu úroveň, aby byla odstraněna nejednoznačnost. K-složitost je tedy atribut binárního řetězce a není to atribut fyzického objektu nebo systému. Rozsah popisu je charakterizován úrovní abstrakce systému.

Oproti Shannonově entropii neexistuje explicitní vzorec pro výpočet K-složitosti. K - složitost v porovnáním s jinými typy komplexity poskytuje další poznatky o objektech nebo systému, ale v praxi nelze dle této hodnoty vyvozovat závěry o celkové složitosti systému. Také neexistuje žádný efektivní způsob výpočtu K-složitosti, kde efektivní znamená, že bude vypočítána v předvídatelném čase. To, že neexistuje efektivní výpočet, však nesouvisí s nedostatečnými schopnostmi nebo technologiemi. Vypočítat univerzální míru složitosti je nemožné. Jedinou možností je zaměřit se na určení komplexity v omezeném rozsahu témat. Lze shrnout, že K-složitost formalizuje myšlenku, že složité nebo náhodné objekty mají dlouhé popisy.[16]

2.2.3. Bennettova logická hloubka

Náhodné objekty bez struktury mají vysokou K-složitost, protože je nelze komprimovat. Pro velké složité systémy není K-složitost použitelná, protože pro tyto systémy nelze sestavit jejich krátký komprimovaný popis. Je třeba však rozlišovat systémy bez možnosti komprimace jejich popisu od systémů, které vyžadují mnoho úsilí pro jejich vytvoření (nekomprimovatelný popis systému nemusí vždy znamenat vysoké úsilí pro jeho vytvoření). Tuto myšlenku formalizuje Bennettova logická hloubka. V této metodě je opět složitost měřena jako množství informací s ohledem na daný cíl. U binárního řetězce je tento cíl najít popis (program), který vyžaduje co nejméně výpočtů, aby se reprodukoval daný řetězec. Bennettova logická hloubka je definovaná jako počet kroků programu, které jsou třeba pro vytvoření daného řetězce. Objekt tak má velkou logickou hloubku, pokud zapouzdřuje velké množství výpočtů nebo návrhů nebo je výsledkem dlouhého výběrového procesu. Pro aplikaci této metody, platí to samé, co pro K-složitost. Jetedy obtížně vypočitatelná a je třeba předem stanovit měřítko popisu systému. Bennettova logická hloubka formalizuje myšlenku, že složitost je nakonec spojena s výpočetním úsilím potřebným k vytvoření objektu.[16]

Vztah logické hloubky a K-složitosti v oblasti IT popisuje tabulka níže. Všeobecně však většina IS jako celek mají jak vysokou logickou hloubku, tak vysokou K-složitost.

Tabulka 1: Vztah logické hloubky a K-složitosti

	Nízká K – složitost (krátký popis)	**Vysoká K-složitost (dlouhý popis)**
Nízký logická hloubka	Standardní kód, který se generuje automaticky.	Databáze se spoustou nestrukturovaných informací
Vysoká logická hloubka	Jednoduchá obchodní pravidla, které však musí být kódovány ručně.	Velká a dobře strukturovaná databáze a kód

Zdroj: upraveno dle [16]

K-složitost a logická hloubka mohou pomoci objasnit, jaká je úroveň abstrakce modelu systému. To je zvláště důležité u složitých systémů, jako jsou IS, které potřebují několik úrovní popisu. Úrovně abstrakce v IT jsou obvykle označovány pojmy, které víceméně odpovídají vrstvám IT architektury (procesní, softwarová, aplikační, hardwarová atd.). V kontextu IT si lze představit systém jako obchodní aplikaci, kde jejím modelem je soubor UML diagramů.

Pokud jsou popisovány nějaké obchodní procesy, měly by být modelovány pomocí souboru UML diagramů, které obsahují pouze informace zřejmé z dané úrovně (procesní) a nic jiného z nižší úrovně (například podrobné technické informace). Zároveň model programu by neměl mít vysokou logickou hloubku, protože by to znamenalo, že množství konstrukcí potřebných k přechodu od modelu k dokončenému systému je velmi velké. Aby modely byly užitečné, neměly by vyžadovat příliš mnoho úsilí pro vybudování systému, který popisují. Dobré modely jsou tedy modely ve vhodném měřítku, pro jejichž vytvoření byla vynaložena většina celkového projektového úsilí. Lze tedy shrnout, že rozsah popisu modelu je prvním atributem úrovně abstrakce a druhým je úsilí potřebné k přechodu od modelu k reálnému systému.[16]

Tyto tři typy složitosti mají společné to, že formalizují myšlenku, že složitost souvisí s množstvím nebo hodnotou informací obsažených v objektu, systému nebo jeho popisu.

Další metody měření složitosti a jejich rozdělení

Obecně lze metody měření rozdělit dle toho, jak odpovídají na tyto tři otázky, které plynou ze složitosti nějakého objektu:

- Jak těžké je ho popsat?
- Jak těžké je ho vytvořit?
- Jaký je jeho stupeň organizace?

Níže je seznam některých metrik pro měření složitosti seskupených podle otázky, na kterou se snaží odpovědět (některé metriky jsou pro nejednoznačnost překladu ponechány v angličtině).[17]

Obtížnost popisu:

- Množství informace;
- Algoritmická složitost;
- Fisher Information;
- Renyi Entropy;

- Kódová délka (Huffmanova, Shannon-Fanova, Hammingova);
- Chernoff Information;
- Rozměr;
- Fraktální rozměr;
- Lempel - Ziv Complexity.

Obtížnost vytvoření:

- Výpočetní složitost;
- Časová výpočetní složitost;
- Prostorová výpočetní složitost;
- Složitost založená na informacích;
- Termodynamická hloubka;
- Náklady;
- Crypticity.

Stupeň organizace:

a) popisující obtížnost popisu organizační struktury:

- Metrická entropie; Fraktální rozměr; Nadměrná entropie;
- Stochastická složitost;
- Sofistikovanost;
- Efektivní míra složitost;
- Skutečná míra složitosti;
- Podmíněné informace;
- Podmíněný algoritmický informační obsah;
- Délka schématu;
- Ideální složitost;
- Hierarchická složitost;
- Stromová subgrafní rozmanitost;

- Homogenní složitost;
- Gramatická složitost.

b) popisující množství informací sdílených mezi částmi systému:

- Algoritmická vzájemná informace;
- Kapacita kanálu;
- Korelace;
- Uložené informace;
- Organizace.

2.3. Komplexita v kontextu informačního systému

Informační systém má za úkol podporovat podnikové procesy. Výše bylo uvedeno, že každý podnikový proces je charakterizován jistou mírou komplexity, tak i jeho podpora v rámci informačního systému nese obdobnou složitost. V dnešní době je tento pojem často uváděn v mnoha metodikách a standardech používaných k řízení informatiky. Pojem komplexita ale bývá zřídka kdy jasně definován. Na složitost informačního systému je možno nahlížet z několika pohledů:

- Jak velké množství dat a informací je zpracováno.
- Jak komplikované procesy jsou systémem řešeny.
- Jak složitá je organizační struktura podniku, který využívá IS.
- Jak je přehledné uživatelské rozhraní.

Pro každý podnik je velmi důležité požívat sofistikovaný SW a HW, neboť oboje má dopady na celou organizaci včetně jejích pracovníků, ale i na ekonomické výsledky. Ve vztahu na velikost organizace je intuitivní, že podnik s velkou a složitou organizační strukturou bude potřebovat také velmi obsáhlý informační systém nebo i více různých dílčích systémů. Je třeba, aby informační systém dokázal efektivně pokrýt co možná nejvíce požadavků plynoucích ze struktury podniku, ale neobsahoval komplexitu větší, než je rámec těchto požadavků. Ideální informační systém by měl pokrýt pouze ty požadavky, jejichž pokrytí přinese vyšší užitek, než jsou celkové náklady spojené s jejich pokrytím. [9]

Na růst komplexity informačního systému má velký vliv složitá organizační struktura podniku, která je definovaná bez znalosti technologických aspektů. Toto má zásadní vliv na

nárůsty komplexity v dalších fázích vývoje a implementace informačního systému, protože s sebou přinese nárůst struktury dat a hrozí jejich redundance. Klade také nároky na design, implementaci a údržbu systému oprávnění pro práci se systémem.

Rostoucí komplexita se nepromítá pouze do oblasti softwaru, ale má velké dopad i na související hardware, kde rostoucí složitost přináší zvyšující se náklady nejen na jeho pořízení, ale hlavně náklady spojené s jeho administrací, provozem a údržbou. Důležitým prvkem každého informačního systému je jeho uživatelské rozhraní, protože má přímý vliv na efektivitu práce uživatele. Složitá a zdlouhavá práce se systémem má neblahý dopad na efektivitu zaměstnanců podniku.

Z výše uvedeného plyne, že komplexitu informačních systémů je nutno řídit, protože v opačném případě roste a s ní rostou podniku náklady a snižuje se efektivita. Již vzniklá velká složitost v informačních systémech se obtížně snižuje. Proto je nutné nejen nedopustit její zvyšování již během návrhu a vývoje informačního systému, ale i během jeho provozu a údržby. Výsledná obsažená optimální komplexita informačního systému musí být vždy ta nejmenší možná, kterou lze dosáhnout. [9]

V současné době je složitost informačních a komunikačních technologie čím dál větším problémem. Stále větší množství funkcí, které informační technologie nabízí, klade také velké nároky na uživatele. Informační technologie ovlivňují běžný život každého člověka v několika směrech. Komplexní systémy s sebou nesou obrovské náklady na jejich nákup, implementaci, poradenství a údržbu, ale také představují riziko pro zajištění kontinuity provozu. Pro zajištění optimální komplexity je nutné při návrhu systému důkladně porozumět podnikové a informační architektuře, ale i rozpoznat mechanismy vedoucí k jejímu vzniku. [10]

2.4. Životní cyklus a obsahové dimenze v kontextu komplexity IS

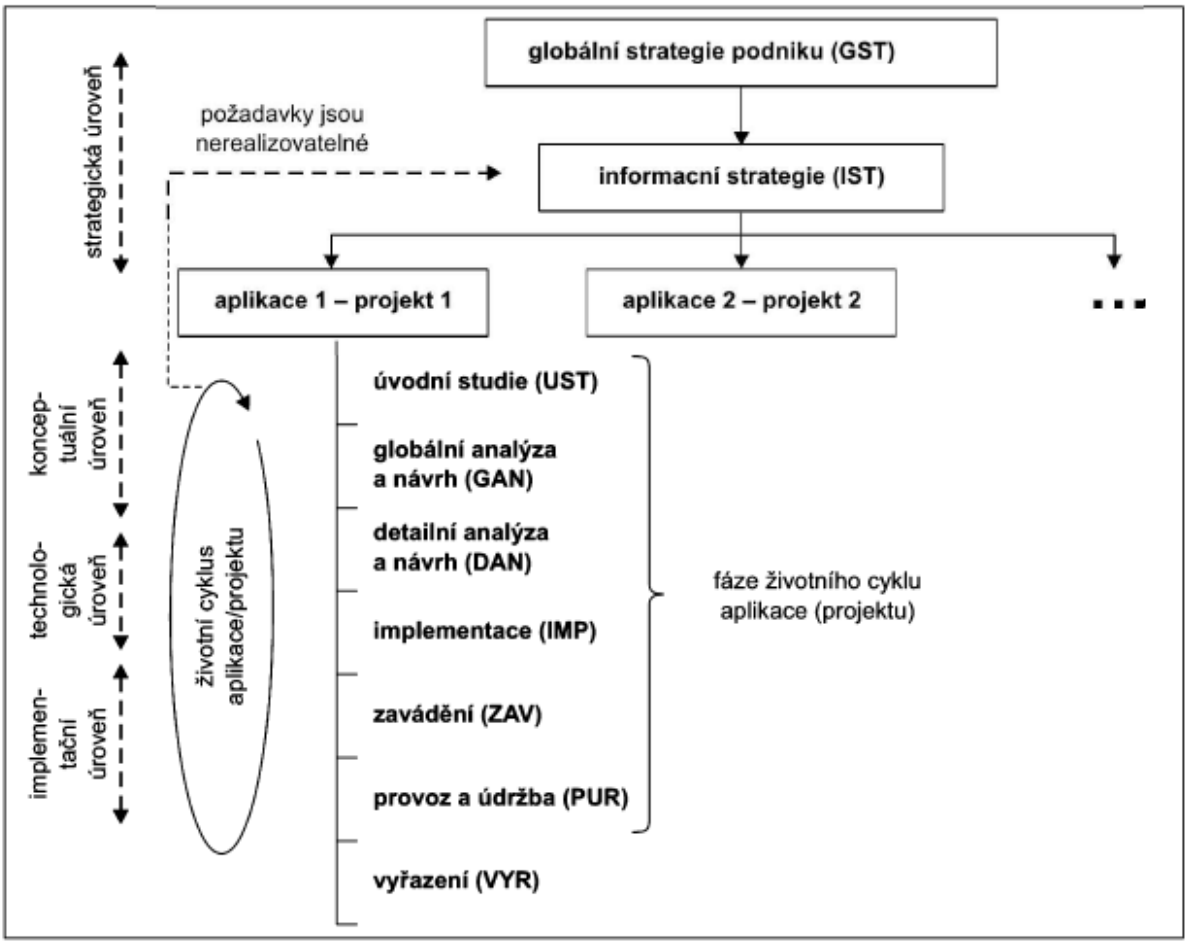
Jedním z cílů této práce je zaměřit se na komplexitu informačního systému v rámci jeho obsahových dimenzí a fází životního cyklu. Jednotlivé dimenze rozlišuje metodika MMDIS (Multidimensional Management and Development of Information Systems). Cílem této metodiky je vývoj, údržba a provoz komplexního a integrovaného informačního systému podniku, který optimálně využívá potenciálu dostupných informačních technologií a informatických služeb k maximální podpoře podnikových cílů, přičemž nemusí být postaven na nejnovějších, nejsofistikovanějších technologiích a službách, ale vybírá z nich ty, které mají ekonomický smysl.

Tato metodika byla vybrána z důvodu toho, že přehledně rozlišuje jednotlivé dimenze, na kterých lze následně zkoumat vliv na komplexitu informačního systému. Pro potřeby této práce bude využito pouze popisu a členění obsahových dimenzí a fází životního cyklu, nikoliv celá metodika.

Metodika zohledňuje všechny faktory (dimenze), které ovlivňují informační systém v jeho životním cyklu a dá se použít jak pro vývoj nových prvků informačního systému, tak i pro rozvoj již existujícího informačního systému.

V metodice MMDIS jsou rozlišovány tyto dimenze[4]:

- Funkce/procesy;
- Data/informace;
- Organizační a legislativní aspekty;
- Pracovní, sociální a etické aspekty, aspekty lidských zdrojů;
- Software;
- Hardware;
- Uživatelské rozhraní;
- Bezpečnost;
- Ekonomické a finanční aspekty.



Obrázek 2: Struktura metodiky MMDIS

Zdroj: [4]

Struktura MMDIS se skládá z globální strategie podniku a informační strategie. Informační strategie obsahuje všechny projekty týkající se IT/ICT. Projekty mají svůj životní cyklus rozdělený na fázi úvodní studie, globální analýzy a návrhu, detailní analýzy a návrhu, implementace, zavádění systému, provozu a údržby, a nakonec vyřazení systému. Struktura metodiky MMDIS znázorňuje Obrázek 2.

2.4.1. Globální strategie

Globální strategie se také nazývá byznys strategií. Není zaměřena přímo na informační systémy, ale dává smysl a cíl všem podnikovým aktivitám. Každý podnik by měl mít formulovanou svoji globální strategii a důsledně ji prosazovat, neboť jinak se podnik vyvíjí neřízeně a může se v důsledku prosazování zájmů jednotlivých řídících pracovníků a jejich útvarů stát, že jsou v rozporu s jinými útvary v organizaci. Existence globální podnikové strategie a její prosazování managementem nejsou automaticky zárukami úspěšnosti podniku,

velmi záleží, jak je strategie formulována a jak je prosazována. Přeje-li si management prosadit nově formulovanou strategii a změnit podnikovou kulturu, měl by věnovat pozornost i informačnímu systému. Globální strategie určuje:

- hlavní předmět podnikání,
- skupiny zákazníků,
- nabízené produkty a služby,
- obchodní model,
- hlavní obchodní partnery,
- zdroje,
- metriky ověření naplnění cílů,
- osoby zodpovědné za plnění cílů.

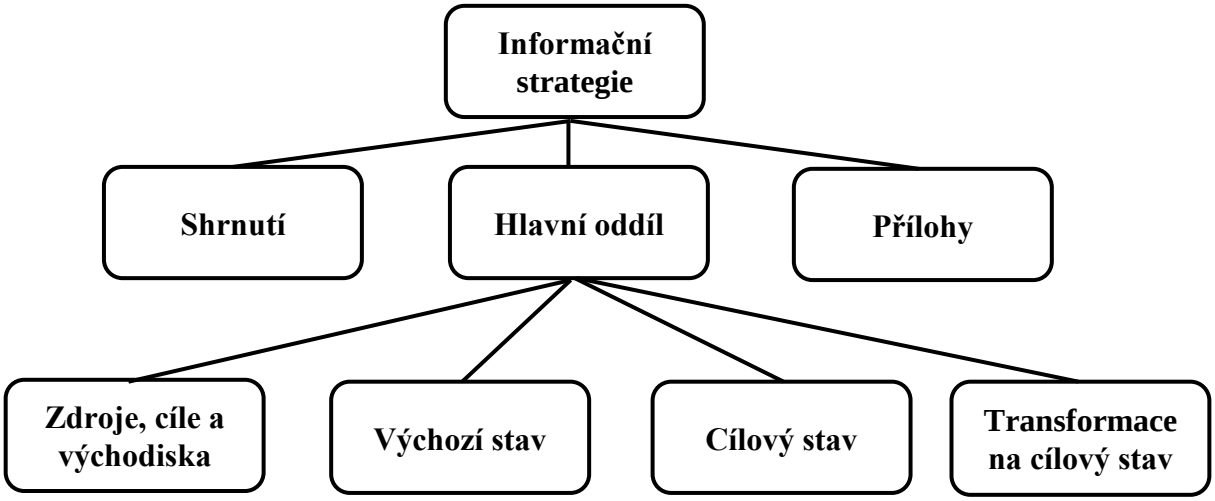
Vlastní tvorba globální strategie je kreativní proces a v různých podnicích se k tvorbě přistupuje odlišně, většinou vychází ze SWOT analýzy, která identifikuje aktuální situaci podniku. Na základě toho se definuje poslání podniku, cíle a jejich priority. Na globální strategii navazují dílčí strategie jako je marketingová, finanční, personální, informační a další. Dílčí strategie rozpracovávají globální a plánují, jak daná oblast podpoří dosažení globálních cílů.[4]

2.4.2. Informační strategie

Jedna z navazujících strategií, jejímž cílem je navrhnout celkovou koncepci IS/ICT podniku tak, aby byly optimálně podpořeny celopodnikové cíle definované v globální strategii a navrhnout jednotlivé projekty, a které budou postupně tuto koncepci realizovat. Tvorba informační strategie se skládá ze tří skupin činností[4]:

- analýza a hodnocení současného stavu IS/ICT, benchmarking, analýza IS/ICT obchodních partnerů, hodnocení SW dostupného na trhu, hodnocení trendů SW a požadavky uživatelů,
- definice cílového stavu IS/ICT, zejména: vizí a cílů informatiky, principů řízení podnikové informatiky, procesů řízení informatiky, architektury a zdroje ICT služeb (jaké služby budou dodávány interně a jaké z externích zdrojů),
- návrh cesty transformace současného stavu do stavu cílového pomocí jednotlivých informačních projektů.

Schématický je struktura informační strategie znázorněna na Obrázek 3.



Obrázek 3: Struktura informační strategie

Zdroj: upraveno dle [4]

Zvolená informační strategie zásadně ovlivňuje veškeré IT projekty v organizaci. Analýza stávajícího a definice cílového stavu obsahuje veškerou budoucí složitost IS.

2.4.3. Životní cyklus aplikace

Tato část zahrnuje vlastní vývoj aplikace od doby rozhodnutí o jejím pořízení až po její vyřazení z provozu. Jednotlivé dílčí fáze jsou níže detailně popsány. Je využito členění dle MMDIS.

- **Úvodní studie**

Tento dokument je zaměřen na detailní posouzení realizovatelnosti byznys požadavků na projekt a na variantní návrh koncepce řešení konkrétní části IS. Závěrem úvodní studie může být, rozhodnutí o realizovatelnosti projektu, o případném rozdělení velkého projektu na více subprojektů nebo v případě, že se dojde k závěru, že požadavky jsou nerealizovatelné například z důvodu vysoké finanční náročnosti nebo v případě malých přínosů z projektu, tak životní cyklus aplikace skončí a je nutné se vrátit zpět k informační strategii, přehodnotit cílový stav, požadavky na projekt a zdroje vyhrazené pro projekt.

Nejdůležitější činnosti a výstupy úvodní studie[4]**:**

- Definice cílů a metrik, jimiž se bude naplnění cílů měřit.
- Popis cílového stavu IS, vymezení rozsahu a okolí části IS řešené projektem.
- Definice typu uživatelů aplikace.
- Návrh koncepce řešení (vývoj nebo nákup).
- Popis aktuálního stavu a dané části IS a potřeby jeho změny.
- Hrubé vymezení funkcionalit.
- Cílové určení zásadních parametrů aplikace (dostupnost, doba odezvy, kontinuita).
- Odhad zdrojů pro řešení, provoz a údržbu (HW, SW, data, lidé).
- Organizace projektu (obsazení projektových rolí).
- Přínosy projektu.
- Rozpočet projektu.
- Rizika projektu.
- Harmonogram projektu.
- V případě vývoje vlastní aplikace je provedeno vymezení podstatných parametrů aplikace, které v budoucnu budou podléhat změnám (nároky na customizaci a personalizaci).
- V případě nákupu typového SW je vytvořen poptávkový dokument, který je zveřejněn. Následně je proveden příjem a vyhodnocení nabídek, vyhodnocení prezentace produktů a služeb uchazečů, výběr nejlepších uchazečů, kteří zpracují úvodní studii zavedení aplikace a na základě těchto studií výběr nejlepšího SW a jeho dodavatele, se kterým je nakonec podepsán kontrakt.

Vztah ke komplexitě IS

Úvodní studie poskytuje hrubé informace o budoucí komplexitě uvažovaného systému a její výstup je na ní přímo závislý. Velké a složité IS jsou velmi nákladné a mohou být rozloženy do více dílčích projektů. V případě, že z důvodu vysoké komplexity nelze odhadnout přínosy z projektu IS nebo jeho náklady, může vysoká komplexita vézt až k zamítnutí realizace projektu.

- **Globální analýza a návrh**

Globální analýza a návrh vychází z úvodní studie a jejím cílem je vymezení požadovaných funkcí a dat řešené aplikace na konceptuální úrovni a formulování požadavků na kvalitu a bezpečnost aplikace. Konceptuální úrovní je myšlena taková úroveň, která řeší obsah a podstatu aplikace pro byznys v pojmech byznysu, tedy nezávisle na implementačním prostředí i na technologické platformě určené pro budoucí provoz aplikace. V rámci tohoto dokumentu je uveden návrh implementační platformy, která bude řešena v detailní analýza a návrhu.

V případě volby řešení pomocí vývoje vlastní aplikace jsou v rámci této fáze prováděny tyto činnosti:

- vytvoření návrhu funkcionalit aplikace,
- vytvoření návrhu automatizace vybraných procesů pomocí workflow,
- sestavení konceptuálního modelu dat,
- upřesnění parametrů aplikace pro customizaci a personalizaci,
- vytvoření grafického manuálu,
- formulování detailních požadavků na kvalitu a bezpečnost aplikace,
- upřesnění implementačního prostředí.

Při řešení projektu pomocí typového SW je v této fázi provedena instalace SW pro účely školení, vlastní školení projektového týmu zákazníka, vytvoření hrubého modelu podnikových procesů a vytvoření prototypu SW, ve kterém se testuje vhodnost funkcí pro podporu vybraných podnikových procesů. [4]

Vztah ke komplexitě IS

Tato fáze ovlivňuje zejména velikost a rozsah IS. Jeho složitost je následně ovlivněna zejména procesy, které bude systém zpracovávat, ale také organizační strukturou podniku, kterou bude systém pokrývat. Hlavním nositelem složitosti jsou dimenze týkající se procesů, dat a organizační struktury. Blíže se vlivem těchto obsahových dimenzí bude zabývat kapitola 2.5.

- **Detailní analýza a návrh**

V rámci této fáze je transformována konceptuální úroveň do technologické, které je závislá na implementační a provozní platformě.

Činnosti a výstupy detailního návrhu jsou[4]:

- návrh způsobu testování,
- případná migrace dat ze starého IS do nového,
- návrh technologické infrastruktury,
- návrh přístupových práv pro jednotlivé typy uživatelů,
- návrh rozhraní na ostatní aplikace,
- návrh programových modulů budoucí aplikace a pro každý modul funkce, které modul zajišťuje,
- interface modulu,
- interní data modulu,
- algoritmus transformace vstupních dat na výstupní,
- návrh fyzické struktury datové základny,
- návrh uživatelského rozhraní,
- v případě typového SW úprava modelů podnikových procesů dle možností SW,
- úprava modelů organizační struktury,
- návrh customizačních parametrů,
- návrh výstupních sestav.

Vztah ke komplexitě IS

Složitost je v této fázi ovlivněna zvoleným technologickým prostředkem. V této fázi se pracuje s materiály, které byly výstupem fáze předchozí. Komplexita detailního technologického návrhu bude tedy minimálně stejná jako ve fázi globální analýzy a návrhu. V praxi je však v složitost navýšena z důvodu omezených možností zvoleného SW řešení.

- **Implementace**

Úkolem fáze implementace je transformace technologické úrovně návrhu IS do implementační úrovně, tedy naprogramování programových modulů ve zvoleném prostředí, realizace fyzického návrhu databáze v konkrétním databázovém prostředí SŘDB, funkční testování, jednotlivých modulů i celé aplikace a kompletace dokumentace (projektové, programové, uživatelské a administrátorské

Činnosti a výstupy implementace[4]:

- vytvoření uživatelské dokumentace a pravidel provozu aplikace,
- aktualizace a kompletace veškeré dokumentace,
- vytvoření databáze v SŘBD,
- nastavení přístupových práv,
- funkční testy a oprava chyb,
- zátěžové testy a optimalizace aplikace,
- nastavení customizačních parametrů,
- úprava obrazovek,
- realizace výstupních sestav.

Vztah ke komplexitě IS

Ve fázi implementace je hlavním aspektem komplexity systému obsahová dimenze týkající se SW, neboť implementace je vlastní tvorba zdrojového kódu aplikace. Komplexitě zdrojového kódu a SW architektuře bude věnována kapitola 2.5.5

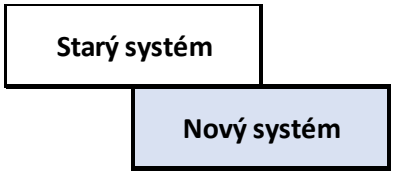
- **Zavádění**

Kompletně otestovaný systém je třeba v dalším kroku nasadit do provozu v daném prostředí, pro které byl vytvořen. Provádí se jeho instalace, transformace původní datové základny a probíhá školení uživatelů. Při školení se zpravidla začíná s vedoucími pracovníky a dále se s programem seznamují zaměstnanci v provozu.

Postupů pro zavádění IS do ostrého provozu je několik. V praxi se nejběžněji užívají následující:

- **Souběžné zavádění**

Informační systém je zaveden souběžně se starým systém. Tento postup je vhodné použít při zavádění jednodušších IS, které jsou jednoduché na ovládání a nejsou proto nutná zdlouhavá školení uživatelů. Uživatelé souběžně obsluhují nový i starý systém po uplynutí dané doby je starý systém odstraněn a uživatel již využívá pouze nový. Princip této varianty zavádění je na Obrázek 4.

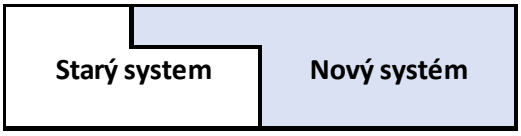


Obrázek 4: Souběžné zavádění

Zdroj: upraveno dle [12]

- **Pilotní zavádění**

Informační systém je zaveden na jednom pracovišti. Na tomto pracovišti se zacvičí i uživatelé z ostatních pracovišť, kam je systém zaveden až po tom, co je pilotní část systému vyzkoušena a uživatelé zaškoleni. Tento způsob je vhodný pro zavádění kvalitativně odlišných IS, které vyžadují rozsáhlé testování v provozních podmínkách. Toto pilotní zavádění umožňuje postupnou transformaci dat z předchozích IS. Schématický je tato varianta znázorněna na Obrázek 5.

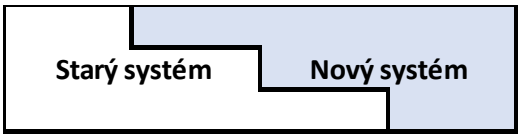


Obrázek 5: Pilotní zavádění

Zdroj: upraveno dle [12]

- **Postupné zavádění**

Využívá se zejména u velice složitých systémů se složitými vnitřními vazbami. Nejprve se zavádějí primární části systému, na kterých ostatní části závisí, po jejich ověření v provozu jsou zavedeny i ostatní části sytému. Obrázek 6 ilustruje postup této metody.

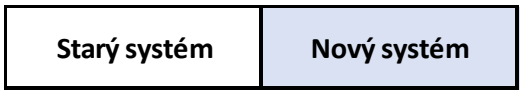


Obrázek 6: Postupné zavádění

Zdroj: upraveno dle [12]

- **Nárazová strategie zavádění**

Na Obrázek 7 je tento způsob zavádění znázorněn. Při tomto způsobu zavádění je starý systém ihned nahrazen novým. Činnost starého systému je ukončena nárazově. Tento postup je velmi riskantní, používá se jen tam, kde není jakýkoliv souběh více systémů možný.



Obrázek 7 :Nárazová strategie

Zdroj: upraveno dle [12]

V praxi však nastává nutnost kombinovat jednotlivé postupy. Nejčastější je kombinace postupu nárazového a postupného.

Výstupem této fáze by měla být instalovaná technologická infrastruktura provozního prostředí, instalovaná aplikace v provozním prostředí, kompletně funkční systém v provozním prostředí, přesunuta veškerá data ze starého systému, vyškolená obsluha, akceptační testy a předávací protokoly.

Vztah ke komplexitě IS

Dle složitosti systému je rozhodnuto o způsobu zavádění. Jak bylo uvedeno výše, tak velké složité IS není vhodné zavádět nárazově a je třeba volit bezpečnější postupné zavádění. Je však třeba věnovat dostatek úsilí také odstranění starého systému, aby nezůstala v IT struktuře podniku duplicita, která by mohla být zdrojem chyb a s nimi spojenými náklady. O důležitosti lidského faktoru a školení bude více napsáno v kapitole 2.5.4.

- **Provoz a údržba**

Jedná se o závěrečnou fázi životního cyklu aplikace. Systém je plně využíván pro potřeby organizace. Do této etapy také spadá údržba systému, tedy zajištění správného provozu, úprava parametrů aplikací nebo změny některých programů tak, aby splňovaly nové požadavky uživatelů. Čím déle vydrží aplikace v této fázi, aniž by bylo nutné výrazně inovovat, tím větší efekty přinesou finanční prostředky a další zdroje vynaložené na její vývoj. Dříve či později, však dojde ke změně požadavků na systém a je třeba provézt tzv. **reengineering**, kdy nastává přehodnocování požadavků na systém. V případě, že nové požadavky není systém schopen splnit pomocí dílčí úpravy, je nutno se opět v rámci životního cyklu vrátit na jeho začátek.

Vztah ke komplexitě IS

V této fázi se promítá komplexita ze všech předchozích. Složitost IS však během jeho provozu roste, a to zejména z důvodu změny požadavků uživatelů. Jedná se zejména o přidávání prvků uživatelského rozhraní, úpravy databáze, úpravy a přidávání výstupních sestav, přidávání nebo úprava komunikačních kanálů nebo rozšíření podnikatelské aktivity podniku, kterou je třeba zahrnout do IS. Jak roste komplexita v závislosti na přidávání prvků bude popsáno v kapitole 2.5.7

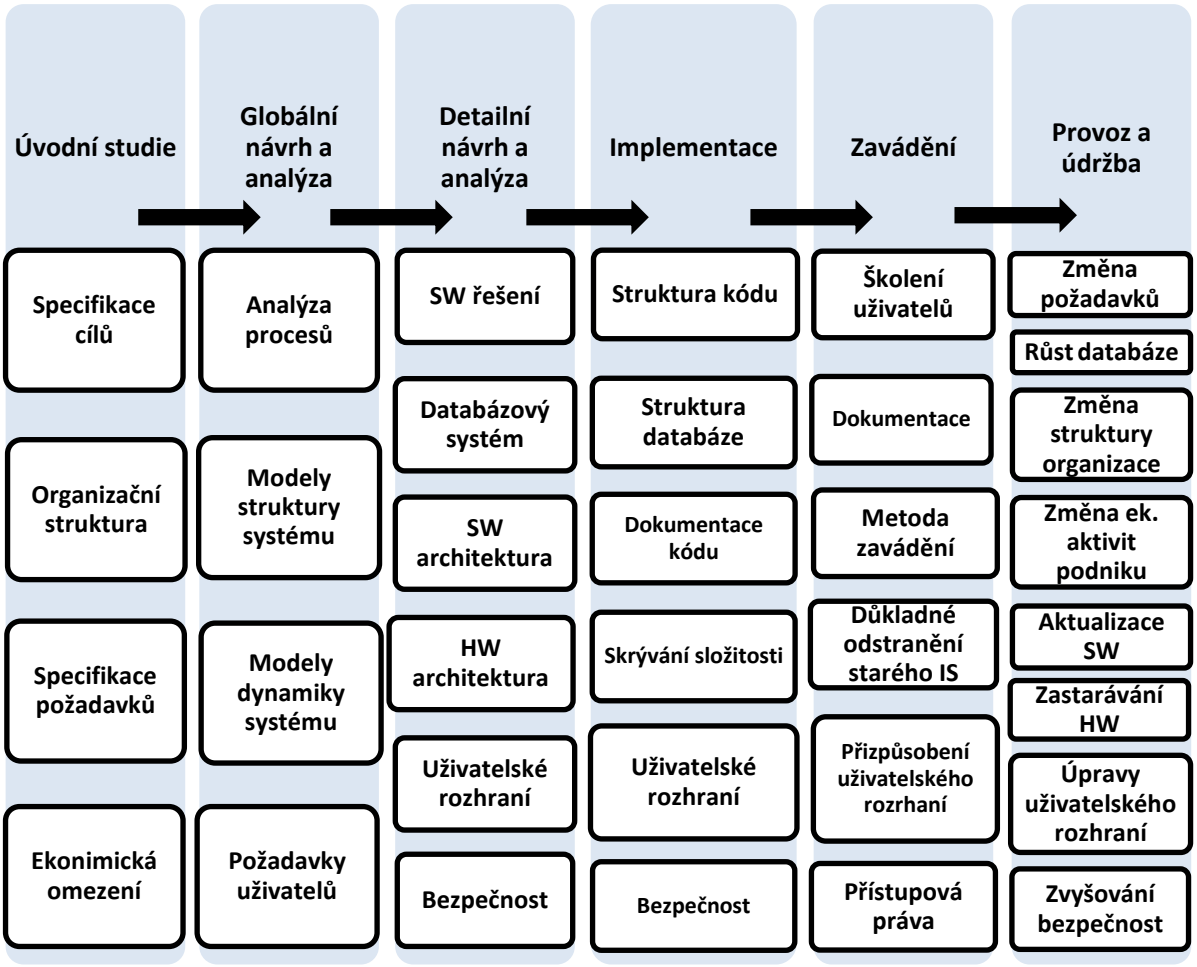
- **Vyřazení**

V případě, že aplikace již není potřebná anebo je její provoz a údržba neefektivní vzhledem k jejím přínosům, dochází k jejímu vyřazení. Vyřazení však nelze provézt pouhým vymazáním programu a dat aplikace z provozní platformy. Je třeba archivovat nezbytná data a současně ošetřit ty vazby aplikací a ty byznys procesy, které byly vyřazením dotčeny.

Vztah ke komplexitě IS

S rostoucí komplexitou IS je i jeho vyřazení z provozu komplikovanější a roste také riziko ztráty dat nebo poškození navazujících aplikací. Jako v každé fázi je i na konci životního cyklu vysoká komplexita spojena s vysokými náklady (práce IT pracovníků, finanční dopad vzniklých škod, prodleva ve výrobě…).

Prvky komplexity v jednotlivých fázích životního cyklu popisuje Obrázek 8. Jsou uvedeny fáze od úvodní studie až po provoz a údržbu. Fáze vyřazení již komplexitu daného IS nezvyšuje, ale jeho komplexita má vliv na průběh a složitost vyřazování.



Obrázek 8: Prvky komplexity IS v jednotlivých fázích životního cyklu

Zdroj: vlastní zpracování

2.4.4. Modely životního cyklu

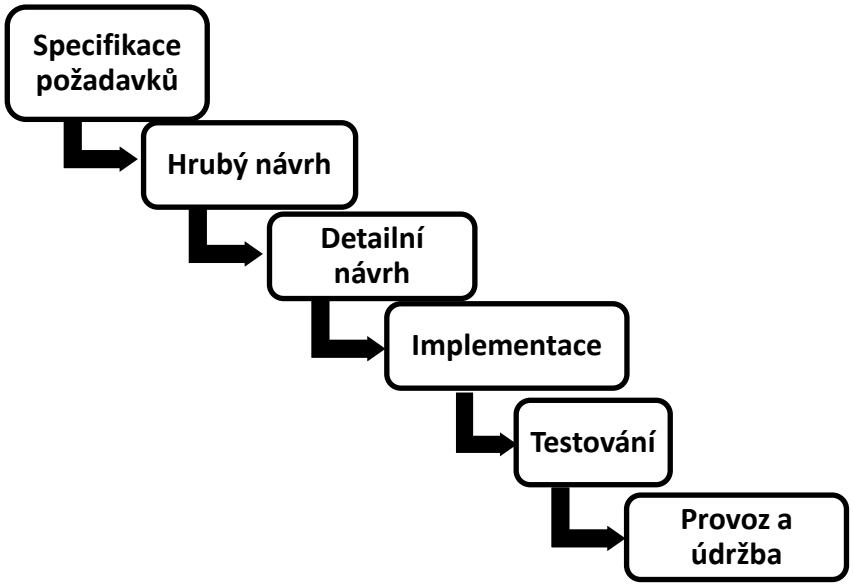
Fáze životního cyklu jsou v praxi realizovány v různých modelech. Ty nejčastější jsou uvedeny v následující podkapitole.

- **Model vodopád**

Jedná se klasický model používaný již v 70. letech 19. století je. Vznikl z důvodu nutnosti zavedení jednotného postupu do vývoje systémů, který umožní řešení komplexnějších problémů díky hierarchické dekompozici a snížení množství chyb precizní kontrolou všech výstupů jednotlivých etap.

Základní charakteristikou tohoto modelu je posloupnost na sebe navazujících etap, které se vzájemně neprotínají. Výstup z jedné etapy je vstupem do etapy následující. Jednotlivé etapy jsou prováděny dle plánu a po skončení etapy se k ní již během cyklu nevrací.

Výhodou tohoto modelu je jeho rychlost a cenová nenáročnost, za předpokladu, že se nevyskytnou problémy. Vhodný pro řešení situace, kdy je znám problém i způsob jeho řešení. Tento typ modelu není příliš vhodný pro řešení složitých systémů, kde není přesně definovaný problém a lze tedy obtížně řešit situaci v krocích definovaných modelem na Obrázek 9.



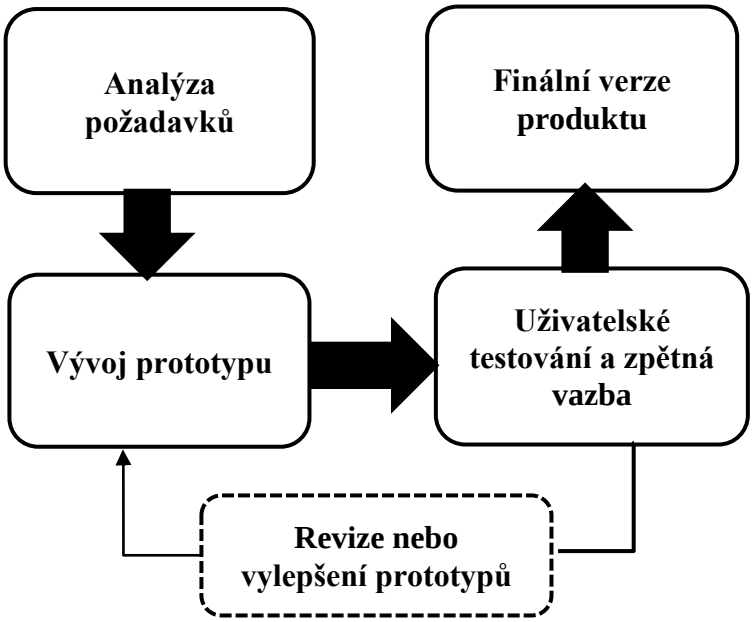
Obrázek 9: Vodopádový model životního cyklu

Zdroj: upraveno dle [13]

Z Obrázek 9 vyplývá, že výsledek projektu zjistíme až po poslední fázi, ve chvíli, kdy je systém předán zákazníkovi a chyby, které jsou objeveny až nyní je velmi složité a nákladné odstranit. V praxi je problém ve velké časové prodlevě mezi objednávkou systému a první verzí předané zákazníkovi, který je již často netrpělivý a další spolupráce s ním je obtížnější. Vodopádový model lze považovat za základní, který je v praxi použitelný hlavně pro malé aplikace, ale i pro velké systémy je jeho použití lepší než náhodné a živelné řešení problému.

- **Prototypový model**

Model vychází z myšlenky, že zákazník zjistí chyby ve specifikaci problému až v době, kdy mu je předložen hotový systém. Hlavním cílem modelu je předložit zákazníkovi první verzi systému, která buď zjednodušeně obsahuje celý řešený prostor, nebo postihuje kompletně pouze část řešeného problému. Prototyp je realizován v co nejkratším čase a v takové funkčnosti, která prezentuje veškerá vnější rozhraní a umožňuje zákazníkovi reagovat na výsledky. Následné změny se provádí na základě připomínek uživatelů, které upřesňují původní požadavky. Postupně je prototyp vylepšován až do doby, dokud není zákazník zcela spokojen, čímž vznikne plnohodnotný požadovaný systém. Model je znázorněn na Obrázek 10.



Obrázek 10: Prototypový model životního cyklu

Zdroj: upraveno dle [13]

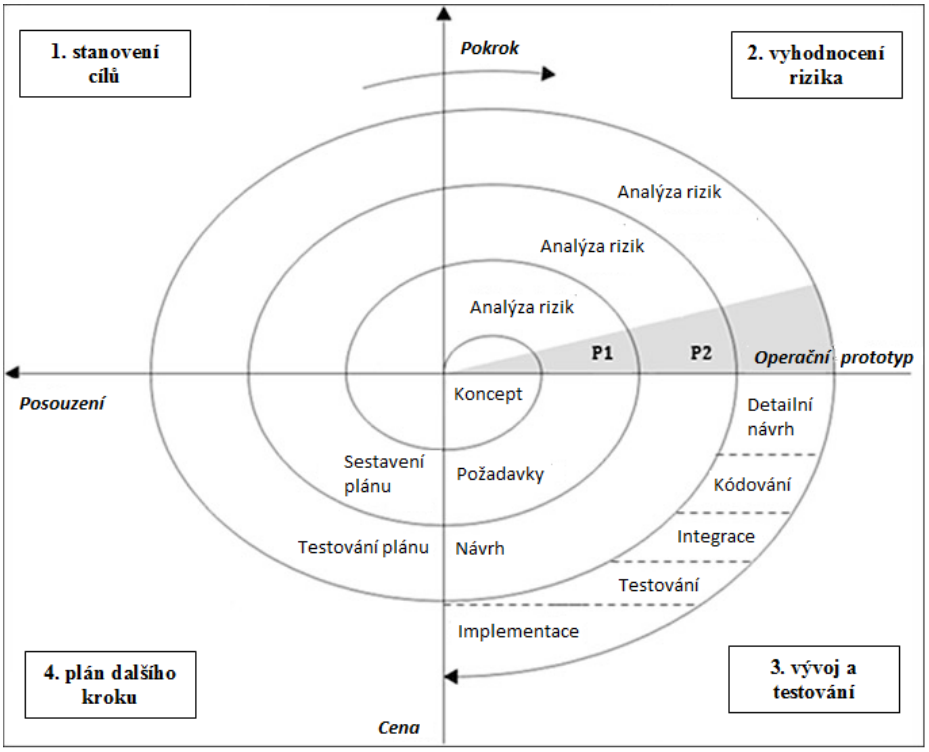
Výhodou tohoto modelu je to, že poskytuje místo pro reakci na změny ze strany uživatelů, na které je možno relativně rychle reagovat. Problém se splněním časového rámce stanoveného ve smlouvě se zákazníkem může nastat v případě, že z důvodu nevystihnutí připomínek uživatelů je prototyp nutno upravovat ve velkém množství cyklů.

- **Model spirála**

Tvůrce tohoto modelu byl v roce 1988 B. W. Boehm. Model kombinuje prototypový přístup a analýzu rizik. Jednotlivé vývojové kroky se neustále opakují, tak že v každém dalším kroku se na již ověřenou část systému přibalují části na vyšší úrovni. Každý krok se skládá z následujících částí:

- Specifikace cílů a určení plánu řešení.
- Vyhodnocení možných řešení a analýza rizik řešení.
- Vývoj prototypu dané úrovně a jeho předvedení a vyhodnocení.
- Revize požadavků a testování správnosti prototypu.
- Ověření výstupu daného kroku je-li v souladu se zjištěnými požadavky.

Model využívá ověřené kroky vývoje a s využitím analýzy rizik předchází možným chybám. Požadavky zákazníků jsou konzultovány v jednotlivých krocích, což umožňuje systém postupně modifikovat dle přání uživatele. Uživatel má možnost ovlivnit již vývoj prvního prototypu. Nevýhodami je zejména časová náročnost spolupráce pro zákazníka, špatně předvídatelné náklady a celková časová náročnost. Je nutné klást velký důraz na správnost analýzy rizika, kdy špatně ohodnocené riziko může mít velký vliv na celý projekt.[13] Schéma modelu vyjadřuje Obrázek 11.



Obrázek 11: Spirálový model životního cyklu

Zdroj: upraveno dle [13]

Zvolení správného modelu životního cyklu má velký dopad na výslednou komplexitu systému. Ovšem při zvolení jakéhokoliv modelu je třeba kompletně a detailně zmapovat dané podnikové procesy.

2.5. Obsahové dimenze informačního systému

Níže budou definovány obsahové dimenze, tak jak jsou rozděleny v metodice MMDIS. Z této metodiky bude využito členění uvedené v kapitole 2.4. U každé dimenze bude detailně objasněn její vztah ke komplexitě IS. Budou také uvedeny metriky pro kvantifikaci komplexity u vybraných obsahových dimenzí.

2.5.1. Funkce/procesy

Dimenze zaměřená na procesy probíhající v podniku a na jejich podporu informačním systémem. Z hlediska výkonnosti je tato dimenze jednou z nejvýznamnějších. Správné modelování procesů má zásadní vliv na komplexitu výsledného systému. Z procesního pohledu je zkoumána dynamika chování podniku při výskytu různých událostí. Úlohou je posoudit, zda jsou podnikové procesy optimální z pohledu doby průběhu procesů, spotřeby zdrojů a kvality výstupů. V další fázi jsou hledána možná zlepšení, které jsou následně implementována. Události lze rozdělit na tři typy: informační (nastává příchodem informace, například příchod objednávky od zákazníka), časovou (souvisí s uplynutím nějaké lhůty, například podání daňového přiznání) a mimořádnou (například požár nebo výpadek proudu). Pro analýzu procesu je nutno přesně identifikovat všechny události. Při definici a modelování procesu v rámci vývoje IS je popsáno, jakými funkcemi IS na tyto události reaguje.

Proces je definují následující vlastnosti [4]:

- cíl procesu,
- událost aktivující proces,
- vlastník procesu,
- kvalitativní a kvantitativní metriky,
- omezení procesu,
- seznam externích vstupů,
- seznam rolí,
- seznam činností,

- doba trvání,
- náklady na proces,
- vstupy a výstupy každé činnosti,
- návaznosti činností,
- přiřazení rolí k činnostem,
- SW podporující proces.

Proces lze graficky znázornit popsaným orientovaným grafem, kdy uzly grafu reprezentují podnikové funkce (činnosti) a hrany reprezentují návaznost těchto funkcí při reakci na danou událost. Při modelování procesů lze využít několik notací jako je UML (diagram aktivit) nebo IDEF. Funkce v procesu mohou být realizované výhradně lidskou činností, lidskou činností s podporou IS nebo zcela automatizovaně.

Návrh procesu prochází 4 kroky [4]:

1. Vytvoření seznamu podnikových procesů a jejich rozlišení na hlavní, řídící a podpůrné.
2. Model návaznosti procesů. Návaznosti mezi procesy jsou vedeny přes události a přes předávaná data. Ke znázornění lze využít EPC diagram.
3. Popis průběhu procesu (před popisem je nutné stanovit zralost procesu a požadovanou úroveň procesu, výsledkem jsou modely průběhů procesů)
4. Průběžná kontrola konzistence s modely v dalších dimenzích zejména v datové.

Vztah ke komplexitě IS

Počet a složitost procesů má zásadní dopad na složitost vlastního IS, do kterého jsou procesy zavedeny. Procesy jsou zpravidla popsány UML diagramy, pro které musí platit, že musí obsahovat jen ty prvky a vazby, které jsou pro systém nutné pro provedení procesu. Každý diagram by měl mít tedy nízkou K-složitost, ale i logickou hloubku (úsilí pro vytvoření reálného systému z modelu by mělo být co nejmenší). Nicméně z praxe vychází, že IS obsahují často více prvků, než je skutečně třeba. Tyto prvky zvyšují složitost a tím i možnou poruchovost a chybovost. [16] Před zavedením vybraných procesů do informačního systému je třeba zaměřit se na jeho komplexitu. Pro určení komplexity procesu lze využít metriku MCC (McCabeova komplexita), která bude vysvětlena níže v kapitole 2.5.5 a CFC metriku.

- **CFC metrika**

Kvantifikuje komplexitu procesu podle jeho workflow grafu. Složitost grafu pak hodnotí podle obsažených uzlů (AND, OR, XOR) a výsledná složitost je pak součtem složitosti všech uzlů v grafu. **CFC** uzlu **AND** je vždy rovna **1.** Pro uzel OR je **CFC = 2^n -1**, kde **n** je počet výstupů z uzlu. Pro uzel **XOR** je **CFC** rovna počtu výstupů z uzlu.[5]

Tyto metriky pomohou analyzovat procesy a případně provézt jejich modifikaci tak, aby byla jejich komplexita snížena. Procesy, o kterých je rozhodnuto, že jsou optimální jsou následně zavedeny do informačního systému. Optimalizace procesů je velmi důležitá z důvodu obtížného snižování složitosti, u již hotového systému.

Snížení složitosti u hotového IS, v podstatě znamená zvýšení jeho jednoduchosti, a to lze provézt jedním zřejmým způsobem, a to je odstranit zbytečné nebo duplicitní funkce. Skutečně zbytečné funkce jsou v IS méně časté. Častěji existují duplicitní komunikační systémy nebo duplikáty úložišť totožnosti (systémy zálohování však nelze považovat za duplikáty). K takové situaci dochází, když nový systém nebo nová funkce nahrazuje starší bez její odstranění. To může znamenat až dvakrát více údržbářských prací. Často se tak stane, že stav, který byl původně dočasný, se stane trvalým. Údržba systémů se pak provádí ručně a ty jsou náchylné k chybám. Takovéto zvýšení složitosti je pak bez jakýchkoli reálných výhod, s výjimkou případů několika lidí, kteří si budou moci zachovat své staré návyky.

Významnější jsou situace, kdy je skutečně snížen počet nebo rozsah funkcí. Zachování pouze základních funkcí umožňuje cílenější práci a zvyšuje šance, že uživatelé budou plně ovládat nástroje, které používají. Mnoho aplikací je navrhováno tak, aby poskytly uživateli velké množství různých úprav a nástrojů, čímž vzrůstá jejich flexibilita. Nevýhodou je, že právě tato flexibilita má za následek horší zvládnutí těchto nástrojů lidmi, kteří je užívají denně. Potřeba přizpůsobení a flexibilita je nadhodnocena, zatímco potřeba plně ovládat IT nástroje zůstává podceňováno.

Flexibilita může v některých případech zvýšit kreativitu pracovníku, ale příliš často vytváří nežádoucí náhodnost systému. Flexibilita by tedy měla být zachována pouze tam, kde je kreativita jasně identifikována jako hlavní faktor pro vytváření hodnoty Přínosem snižování rozsahu a počtu funkcí spočívá v tom, že tyto odstraněné funkce jsou často i ty nejsložitější, které je třeba implementovat a udržovat.

Snížení složitosti redukcí znamená snížení některých složek K-složitosti systému. Popis IS je kratší, a tak je snadnější pochopitelnější. Je také pravděpodobné, že dojde ke snížení entropie, kterou s sebou nesou slabě předvídatelné operace údržby, při odstranění duplicit v systému.

Pro modelování funkčních požadavků lze využít UML diagram užití. Tento diagram zachycuje všechny aktéry a jejich interakce se systémem (případy užití). Pro kvantifikaci komplexity diagramu užití je možno použít tzv. **Merchesiho metriku** a **součet prvků diagramu**.

- **Merchesiho metrika**

Metrika dle M. Merchesiho využívá počet případů užití (N_{CU}), počet aktérů (N_a) a počet vazeb zahrnutí (include) a rozšíření (extends). Z těchto následně Marchesi navrhl metriku **4UC** [21], jejíž rovnice je následující:

$$UC4 = K_1 UC1^2 + UC3 + K_2[smm([C]) - smm([E])] \quad (1)$$

Kde:

- **K1** a **K2** jsou konstanty (méně než jedna) a musí být vypočítány empiricky,
- **UC1** představuje počet případů použití. **UC1=N_{CU},**
- **UC3** je celkové množství vazeb mezi případy užití a aktéry bez redundantních vztahů zahrnutí (include) a rozšíření (extends),
- **[C]** je matice (s rozměrem N_a x N_{CU}) a prvek c_{ik} matice [C] má hodnotu 1, pokud aktér k má vztah s případem užití k, v jiném případě má hodnotu 0,
- **[E]** je matice (N_a x N_{CU}) reprezentující vztahy mezi případy užití po odstranění redundantních vztahů zahrnutí (include) a rozšíření (extends).
- **smm([C]) - smm([E])** je počet vazeb zděděných všemi případy užití, které rozšiřují (extends) nebo používají (uses) jiné případy použití.[21]

- **Součet prvků diagramu**

Pro srovnání lze také využít prostý součet všech prvků v diagramu (užití, aktérů a vazeb). V tomto případě počet případů užití vztahů a aktérů, včetně vztahů rozšíření (extends), užití (uses) a zahrnutí (include). Výsledkem je následující vztah:

$$C_{uc} = \sum aktérů + \sum užití + \sum vazeb \quad (2)$$

Tato metrika však neodráží striktně složitost diagramu, ale spíše jen jeho velikost, pro srovnání s Merchesiho metriku však je možné ji použít.

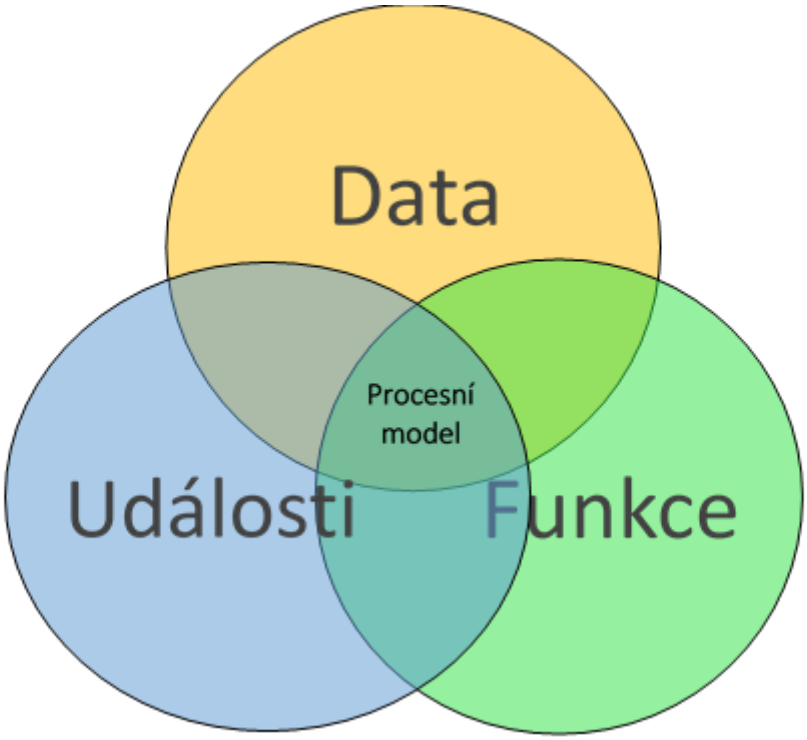
Výsledky z obou metrik lze použít při rozhodování o volbě struktury diagramu, která se následně promítne do komplexity v další fázi vývoje.

2.5.2. Data/informace

Datová dimenze se zabývá typy informací, které jsou zapotřebí při jednotlivých podnikových aktivitách, obsahem a strukturou datové základny podniku a jejím fyzickým uložením. Výstupem této dimenze je realizovaná datová struktura IS/ICT. Při řešení IS jsou analyzována a navrhována řada významných charakteristik informace a je určen typ informace, zda se jedná o **signální** (vzniká při výměně informací mezi prvky systému, představují vstupy a výstupy funkcí IS), **strukturální** (popisuje strukturu, pravomoci a pravidla chování funkcí systému) nebo **genetickou** (uložená ve vlastní paměti systému a předurčuje jeho chování).[4]

V rámci životního cyklu projektu informačního systému se pracuje zejména se signálními informacemi.

Dále je stanovena úroveň obecnosti informace, zda je konkrétní (dopis zákazníkovi) či abstraktní (šablona a styly dopisu), předmět, ke kterému je informace vztažena, časové omezení informace, pravděpodobnostní hodnota informace (u informací, bez zaručené pravdivosti), úroveň agregace, zdroj, objem (objem uložených a zpracovaných dat měřený v bitech je charakteristikou, která ovlivňuje parametry aplikace i technologickou infrastrukturu), místo a způsob uložení.[4]



Obrázek 12: Vztah mezi daty, funkcemi a událostmi

Zdroj: upraveno dle [4]

Vztah mezi daty, funkcemi a událostmi popisuje Obrázek 12. Výsledkem průniku je procesní model. Procesní model a jeho stupeň abstrakce, jeho optimalizace a velikost má zásadní dopad na komplexitu celého IS. Pro procesní model musí platit kritéria uvedená v kapitole 2.2.1.

Vztah ke komplexitě IS

Množství a struktura dat, které je třeba v rámci podnikových procesů zpracovat významně ovlivňuje databázi IS. Vlastní databázové řešení je pak nositelem této komplexity. Pro kvantifikaci komplexity databáze je možno použít metriku zvanou „Database complexity" dále jen DC. Před konstrukcí každé databáze musí být vytvořeny nejméně dva různé modely dat, které popisují datovou strukturu v systému. Tyto modely jsou ERD a RMD. ERD popisuje všechny entity s jejich atributy v systému a vzájemné vztahy mezi nimi. RMD je na základě daných pravidel vytvořen z ERD. RMD vedle atributů a obsahuje také primární a cizí klíče v každém vztahu. Uvedené modely zajistí minimální redundanci v budoucí databázi. Pokud IS má databázi namodelovanou a normalizovanou ve tvaru 3NF, lze pomocí DC snadno odhadnout složitost databáze. Nenormalizovaná databáze nemůže být měřena pomocí metody DC.[24]

Pro výpočet složitosti databáze je třeba určit složitost každé relace v databázi. Pro každý vztah je třeba sečíst všechny příslušné prvky a výsledek představuje váhu relace **W**. Nechť **A** je počet atributů v daném vztahu, **K** je počet klíčů ve vztahu (součet primárních a sekundárních klíčů), **I** je počet indexů ve vztahu a **F** je počet cizích klíčů ve vztahu, pak váha **W** každé relace je může být popsána vzorcem[24]:

$$W = A + K + I + F \tag{3}$$

Pro výpočet složitosti **C** celé databáze je třeba sečíst všechny váhy W všech relací databáze. Toto je znázorněno vzorcem:

$$C = \sum_{i=1}^{n} W_i \tag{4}$$

Písmeno **n** ve vzorci reprezentuje počet vztahů v databázi a **W** představuje váhu každé relace a **C** představuje složitost databáze. i je libovolné číslo od 1 do **n**.

Výsledek metody DC je možno použít pro odhad doby vývoje softwaru pro databázi, na základě čehož lze odpovídajícím způsobem plánovat zdroje. Lze také odhadnout vlastní náklady na IS nebo náklady na reengineering softwaru, který bude v budoucnu třeba. **Rozhodnutí o tom, zda je výsledná komplexita vysoká či nízká musí provézt expert v dané oblasti se zkušenostmi s návrhy databází.** [24]

Pro modelování struktury systému se v UML využívá také diagramu tříd. V diagramu jsou znázorněny jednotlivé třídy systému včetně jejich atributů a operací mezi nimi. Tento diagram je platformově závislý a je tvořen již s ohledem na zvolený programovací jazyk. Pro kvantifikaci komplexity tohoto diagramu lze využít Merchesiho metriku a In metriku. Obě metriky jsou popsány níže:

- **Merchesiho metrika *M* (dle M. Merchesiho), kde jsou využity následující veličiny:**
    - celkový počet tříd (OA_1),
    - celkový počet asociací (OA_2).[21]

Výsledná komplexita je pak dána vztahem:

$$M = OA_1 + OA_2 \tag{5}$$

- **In metrika *I* (dle P.Ina)**

  V metrice jsou určeny klíčové indikátory složitosti, jako je počet: tříd (TNC), vztahů dědičnost (TNIR), vztahů použití (TNUR), asociačních vztahů (TNA), rolí (TNR), operací (TNO), parametrů (TNP) a atributů (TNCA).[11]

Celkovou komplexitu výsledného digramu tříd pak lze vyjádřit vztahem:

$$I = TNC + TNIR + TNUR + TNA + TNR + TNO + TNP + TNCA \tag{6}$$

Obě metriky budou použity v poslední kapitole při analýze komplexity reálného informačního systému.

2.5.3. Organizační a legislativní aspekty

Podnikové aplikace a podnik samotný musí splňovat platnou legislativu. Také IS/ICT by měl plnit platné normy a standardy. V této dimenzi se určují zákony, normy a směrnice, které musí být při tvorbě IS respektovány.

V rámci organizační dimenze se popisuje organizační struktura podniku platná v době provozu IS, dislokace jednotlivých útvarů, typy a počty funkčních míst v jednotlivých útvarech a pracovní náplň jednotlivých funkcí. Od pracovní náplně funkčního místo jsou také odvozena přístupová práva skupin uživatelů IS. Od počtu uživatelů je následně také odvozen počet licencí daného SW. Organizační strukturu lze ve většině podniků rozdělit na **primární**, které má obvykle tvar stromu a určuje podřízenost a nadřízenost útvarů a funkcí v podniku a **sekundární,** které jsou vytvářeny ad hoc pro řešení specifických problémů (projektů). [4] V této dimenzi

jsou také určeny pravidla pro vývoj, údržbu a provoz IS, definují se útvary zaměřené na vývoj a provoz IS, zodpovědnosti a pravomoci jejich pracovníků, postupy platné při vývoji IS.

Vztah ke komplexitě IS

Do organizační dimenze spadá také vlastní struktura podniku jako systému, která se následně promítá do IS. Každý systém je tím více složitý, čím je jeho struktura neuspořádanější. Dle teorie informace má vysokou míru neuspořádanosti, a také vysokou K-složitost, neboť jeho popis bude velmi dlouhý. Snížení složitosti v takovýchto systémech lze pomocí uspořádání a třídění.

Uspořádání a třídění pomáhá bojovat proti poruchám každého systému. Organizování systému znamená uvedení struktury do něčeho, co původně strukturu nemá. V kontextu IT existuje řada známých faktorů, které přispívají ke zvýšení počtu poruch v IS, pokud nejsou ošetřeny a zvládnuty. Jedná se o:

- akumulaci „quick-and-dirty „částí kódu,
- rozmanitost technologií, které je třeba vzájemně propojit,
- postupné zastarávání technologií,
- postupná ztráta znalostí o systému v důsledku výměny zaměstnanců,
- chybějící celkové modelování IS,
- chaotické politické síly, které často hrají významnou nebo dokonce dominantní roli.[16]

Komplexní systém je složený z řady subsystémů nebo částí. Statisticky, existují převážně více nestrukturované konfigurace než strukturované. Čím více konfigurací systému je k dispozici, tím vyšší bude jeho poruchovost. Organizační procesy jsou zpravidla pomalé, protože vyžaduje nahlédnutí a pečlivé přemýšlení.

Zkušenost ukazuje, že organizování systému je iterativní proces. Po dokončení by měl mít výsledný systém s menším počtem součástí, než bylo v původním systému. Systém se tak stává více předvídatelným a snadnějším na pochopení jako celek, což usnadňuje i jeho úpravy. Uspořádání složitosti snižuje počet dílčích částí systému, který je pak pro lidskou mysl srozumitelnější a více předvídatelný, čímž je úsilí potřebné k tomu, aby měl uživatel rozumný přehled o systému jako celku sníženo. Realizace změn v případě jejich potřeby bude jednodušší a rychlejší.[16] Uspořádání a třídění je třeba v podniku aplikovat před zavedením organizační složitosti do IS.

2.5.4. Personální, sociální a etické aspekty

Cílem dimenze je zjistit, jaká je stávající struktura pracovníků, jejich znalosti, zvyky, schopnosti a případná omezení v užití IS, určit potřebnou strukturu pracovníků podniku (počet, kvalifikace) po zavedení nového IS, analyzovat, jaké sociální a etické dopady může mít přechod na novou verzi IS a jaká opatření (personální) si zavedení nové verze vyžádá.

Kvalitní systém školení jak nových, tak stávajících pracovníků podniku je jednou z klíčových otázek efektivity IS. Za druh školení s nízkou efektivitou lze považovat systém školení, kdy školení provádí služebně starší kolega, který předává jen ty znalosti, které sám využívá, přičemž nejsou využity všechny funkce IS. Efektivnějším systémem školení je použít e-learningu nebo instruktážních videí.

Tato dimenze má zásadní vliv na úspěch ICT projektu. Změny musí brát v úvahu úroveň znalostí a dovedností uživatelů. V případě, že uživatelé nejsou na novou technologii řádně připravení, může i velmi kvalitní, sofistikovaná a drahá aplikace způsobit ztráty podniku díky neschopnosti nebo neochotě uživatelů s aplikací pracovat.[4]

Vztah ke komplexitě IS

Schopnosti uživatele jsou aspekt, značně ovlivňující komplexitu IS v tom smyslu, že pro zdatnější a dobře zacvičené uživatele může být IS složitější (zejména uživatelské rozhraní) a při tom práce s ním nebude činit pracovníků žádné potíže a nebude docházet k chybám způsobeným lidským faktorem. Je tedy důležité si práci s IS dobře osvojit, protože znalost a pochopení dělají ze složitých problémů jednodušší. Obojího lze dosaženou pomocí učení.

Učení je proces, kterým lidská mysl vytváří účinnou mentální mapu, která se vztahuje na danou kategorii problémů. Ve vztahu na logickou hloubku by neměla znalost být příliš hluboká, tedy měla by být výsledkem velkého úsilí a snadno použitelná v praktické situaci. V kontextu informačních technologií je všeobecně potřeba učení tam, kde jsou v interakci technické a lidské aspekty IS. Z tohoto plyne, že je třeba věnovat velkou pozornost analýze budoucích uživatelů systému již při návrhu systému, alespoň při návrhu uživatelského rozhraní a zároveň věnovat dost času školení a tréninku zaměstnanců při práci se systémem ve fázi zavádění. Zvýšením schopností a prohloubením znalosti uživatelů, lze ušetřit nemalé finanční prostředky, které by museli být vynaloženy na dodatečné zjednodušování a „zapouzdřování“ složitosti. [16]

2.5.5. Aplikace (aplikační software)

V rámci této dimenze je určena aplikační architektura IS, tedy určení typů a vzájemných vazeb jednotlivých komponent programového vybavení. Zde se určuje, zda a jaká komponenta bude vyvinuta vlastnímu silami, a která bude nakoupena. V případě vývoje vlastními silami je v této dimenzi rozhodnuto o vývojovém prostředí, nástrojích, modulech a vzájemných vazbách mezi moduly. [4]

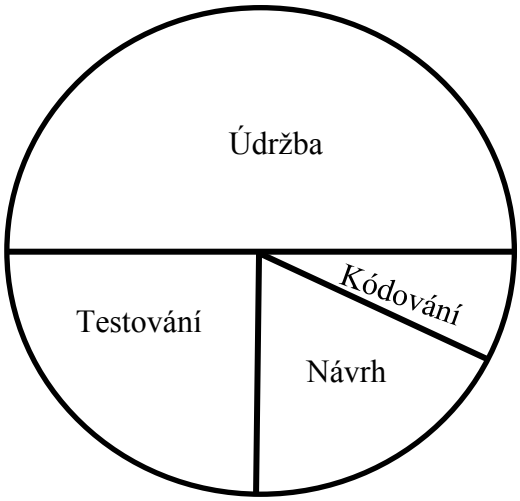
Vztah ke komplexitě IS

Architektura softwaru je zásadním aspektem ovlivňujícím složitost IS, a proto je důležité pochopit co je příčinnou růstu složitosti a tyto příčiny zmírnit. Výše bylo uvedeno, že vyčíslit komplexitu složitého systému je nemožné. Při zkoumání složitosti SW architektury, však existuje několik metod, které umožňují matematicky vyjádřit její komplexitu. Bližší zkoumání softwarové architektury je motivováno potřebou zmírnit zdroj složitost, kterou představuje v rámci IS.[16] Každý informační systém je tvořen zdrojovým kódem.

Pro kvantifikaci a „hlídání“ složitosti kódu se používá několik metod. Níže bude přiblížena McCabeova cyklomatická metoda a Halsteadovi metriky. Pro kvantifikaci komplexity vlastní SW architektury bude přiblížena metrika zvaná invariantní měřítko složitosti.

- **McCabeova cyklomatická metoda (MCC)**

Informační systém je vždy nějaký programový prostředek. Pro určení míry komplexity každého programu je třeba komplexitu kvantifikovat. Metodu pro změření komplexity uvedl T. J. McCabe, který využívá pro kvantifikaci teorii grafů a zabývá se tzv. cyklomatickou komplexitou. V této metodě jsou využity pojmy z teorie grafů. Na složitosti grafu je pak prokázáno, že komplexita je nezávislá na fyzické velikosti (přidání nebo ubrání funkce ponechává složitost beze změny) a složitost závisí pouze na rozhodovací struktuře programu. V softwarovém inženýrstvím je řešena kritická otázka, a to, jak modularizovat softwarový systém, tak aby výsledné moduly byly testovatelné a snadno udržovatelné. To, že otázky ohledně testovatelnosti a udržovatelnosti jsou důležité, je potvrzeno skutečností, že fáze testování softwaru zabere často polovinu vývojového času a tím pádem s sebou nese i nemalé náklady. [20]. Typické členění nákladu na software popisuje Obrázek 13



Obrázek 13: Typické členění nákladů na SW

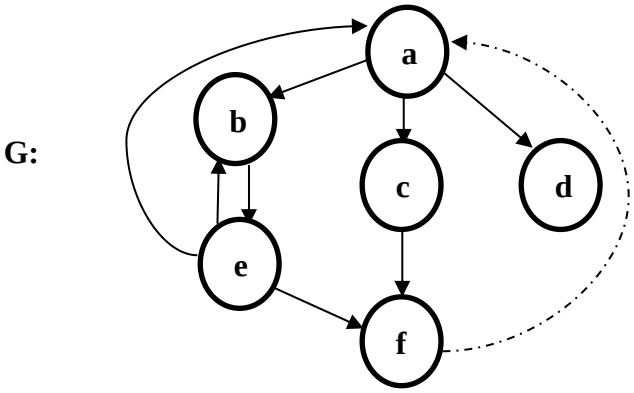
Zdroj: upraveno dle [30]

McCabeova matematická technika je založena na toku v control-flow grafech programu. Předchůdci této metody se zabývaly pouze fyzickou velikostí programu, což není adekvátní, neboť program s 50 řádky sestávající se z 25 po sobě jdoucích "IF-THEN" příkazů by mohl mít až 33,5 milionu odlišných cest, z nichž by pravděpodobně jen malá část byla někdy testována. Program by byl tedy fyzicky malý, ale jeho složitost by byla na vysoké úrovni, proto nelze program posuzovat pouze dle velikosti kódu.

McCabeova metoda nepracuje s počtem řádků programu, ale používá tzv. cyklomatické číslo, které odpovídá počtu cest přes program. [20] Cyklomatické číslo ***v(G)*** grafu **G** s **n** vrcholy, **e** hranami a **p** připojenými komponentami lze popsat vztahem:

$$v(G) = e - n + 2p \tag{7}$$

Cyklomatické číslo se rovná maximálnímu počtu lineárně nezávislých větví v programu. Při řešení se k programu přidruží orientovaný graf, který má jedinečné vstupní a výstupní uzly. Následující graf na Obrázek 14 popisuje program se vstupním uzlem **"a"** a koncovým uzlem **"f"**.



Obrázek 14: Control-flow graf programu

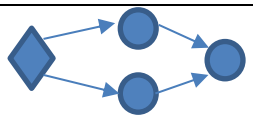
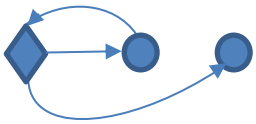
Zdroj:[20]

Pro získání cyklomatického čísla je třeba počítat se spojením výstupního a vstupního uzlu tak, aby existovala cesta mezi jakoukoli dvojicí libovolných vrcholů. Maximální počet lineárně nezávislých větví v grafu **G** je pak ***v(G)* = 9 (hran) – 6 (vrcholů) + 2 = 5**. Tedy 5 nezávislých větví: (abefa), (beb), (abea), (acfa), (adcfa).

Vztah mezi spolehlivostí výpisů programu a komplexitou dle cyklomatického čísla byl testován s výsledkem, že v testovaných algoritmech se složitostí ***v(G)* ≥10** docházelo k 5,6 chybám na 100 výpisů, kdežto u ***v(G)*≤10** došlo k 4.59 chybám ze stejného počtu výpisů. [30]

Příklady několika contro-flow grafů jednoduchých struktur a jejich cyklomatických čísel je v Tabulka 2 níže.

Tabulka 2: Příklady cyklomatické složitosti vybraných struktur

podmínka	**Struktura**	**Cyklomatické číslo** ***(v(G) = e–n + 2p)***
posloupnost		1
IF-THEN		2
WHILE		2
UNTIL		2

Zdroj: upraveno dle [20]

Vlastnosti cyklomatické složitosti lze shrnout takto:

1) $v(G) > 1$.

2) $v(G)$ je maximální počet lineárně nezávislých cest v G.

3) Vložení nebo odstranění funkčních příkazů do G neovlivňuje $v(G)$.

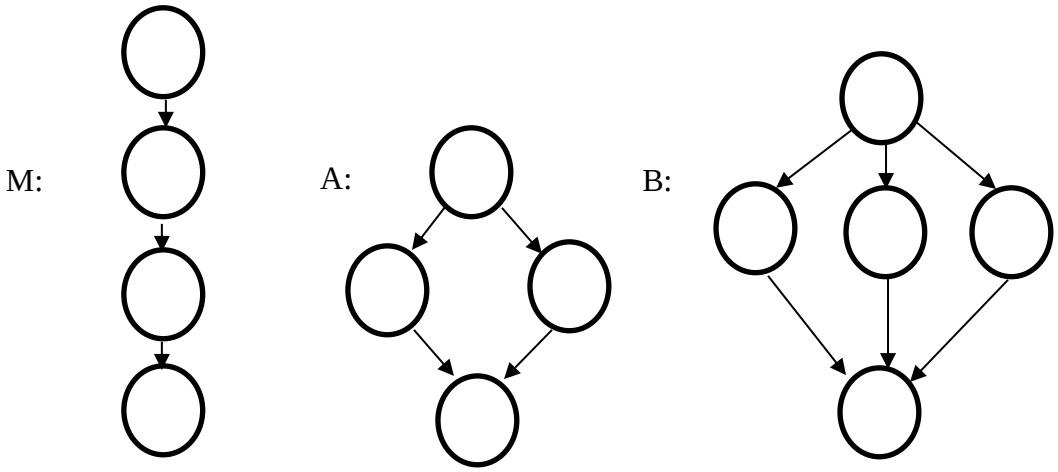
4) G má pouze jednu cestu, pokud a pouze pokud $v(G) = 1$.

5) Vložení nové hrany do G zvyšuje $v(G)$ o jedna.

6) $v(G)$ závisí pouze na rozhodovací struktuře G.

Velikost ***v(G)*** je dle vzorce ovlivněna také veličinou **p**, která udává počet připojených komponent. Výše bylo napsáno, že graf pro výpočet cyklomatické komplexity programu má jedinečné vstupní a výstupní uzly, všechny uzly jsou přístupné od vstupního a ze všech uzlů je dosažitelný výstupní uzel. Dle tohoto by všechny řídicí grafy měly pouze jeden připojený komponent.

Pro složité programy by byl graf velký a výpočet složitý, proto lze program rozdělit například na hlavní program M a dva volané podprogramy A a B s níže uvedenými strukturami.[20]



Obrázek 15: Rozložení programu na podprogramy

Zdroj: upraveno dle [20]

Výpočet MUAUB je nyní při **p** = 3 takovýto: ***v(MUAUB)* = e–n + 2 p = 13-13 + 2*3 = 6**. Tato metoda s **p ≠ l** může být použita pro výpočet složitosti kolekce programů. Z uvedeného výpočtu také plyne, že ***v(MUAUB) = v(M) + v(A) + v(B)*** a složitost sbírky **C** řídicích grafů s připojeným **k** komponenty se rovná součtu jejich složitosti.

Pro $\mathbf{C_i}$, kde platí **l <i <k** a **k** označuje odlišné připojení komponenty a nechť $\mathbf{e_i}$ a $\mathbf{n_i}$ je počet hran a uzlů v i-té připojené komponentě, pak pro složitost programu platí:

$$v(C) = e - n + 2p = \sum_1^k e_i - \sum_1^k n_i + 2k = \sum_1^k (e_i - n_i + 2) = \sum_1^k v(C_i) \qquad (8)$$

Vzhledem k tomu, že výpočet *v(G)* může být pro velké programy docela zdlouhavý, došlo se k závěru, že složitost lze vypočítat jednoduchým součtem predikátů v kódu a není nutno se zabývat control-flow grafem. Ovšem vzhledem k tomu, že v praxi se používají složené predikáty, jako např. IF "C1 AND C2", kde bychom měli pro výpočet složitosti počítat s hodnotou 2, bylo zjištěno, že je vhodnější spočítat počet podmínek namísto predikátů. [22]

Využit MCC lze i u objektově orientovaných programovacích jazyků, kdy množina operací objektu reprezentuje souhrn algoritmů, který, jak ukazuje McCabe, má cyklomatickou složitost rovnající se součtu složitostí každého z jednotlivých algoritmů. Zde však omezení složitosti každé metody pro objekt na doporučený limit 10 bez omezení počtu operací nemusí snižovat složitost objektu. Spíše než omezit počet operací pro objekt na 10, zdá se být efektivnější omezit součet komplexity operací objektu na 100. Pak objekt může mít více než 10 operací.

Pokud jsou některé objekty složitější, měly by být shromážděny další empirické důkazy, aby se zjistilo, zda je doporučený limit nepovinný. Samozřejmě je třeba zvážit, jak počet, tak i individuální složitost objektových metod. [6]

- **Halsteadovi metriky**

Dalším druhem metrik pro kvantifikaci komplexity SW jsou objemové metriky, které se snaží reprezentovat složitost jako měřítko velikosti programu. Tyto metriky jsou založeny na počtech operandů a operátorů v programu. Zaměřují se na objem, který přímo narůstá s počtem jedinečných operátorů a operandů. Metriky vychází z toho, že počítačový program je implementací algoritmu považovaného za soubor tokenů, které lze klasifikovat jako operátory nebo operandy. Tyto metriky jsou zahrnuty v řadě současných komerčních nástrojů, které počítají softwarové řádky kódu.[6] Počítáním tokenů a určením operátorů a operandů lze získat následující veličiny:

n_1=počet různých operátorů.

n_2 = počet různých operandů.

N_1 = celkový počet výskytů operátorů.

N_2 = celkový počet výskytů operandů

Z těchto veličin jsou následně zjišteny jednotlivé metriky, jsou to například:

- **Délka programu** (celkový počet výskytů operátorů a celkový počet výskytů operandů).

$$N = N_1 + N_2 \tag{9}$$

- **Slovní zásoba** (celkový počet různých operátorů a operandů)

$$n = n_1 + n_2 \tag{10}$$

- **Objem programu**, který je úměrný k velikosti programu, představuje velikost programu v bitech, kterou je třeba uložit. Tento parametr závisí na konkrétní implementaci algoritmu.

$$V = N * \log_2 (n) \tag{11}$$

Potenciální objem je pak: $V^*=(2+n_2) \log_2(2+n_2)$, z čehož lze odvodit úroveň algoritmu vztahem:

$$L=V^*/V \tag{12}$$

- **Programová obtížnost**

$$D = (n_1 / 2) * (N_2 / n_2) \tag{13}$$

$$D = 1 / L \tag{14}$$

- **Programovací úsilí** (množství duševní činnosti potřebné k překladu stávajícího algoritmu do implementace ve specifikovaném programovém jazyce.[6]

$$E = V / L \tag{15}$$

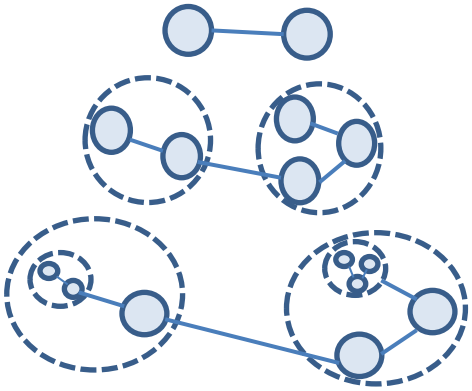
- **Programovací čas** (čas v minutách potřebný k překladu stávajícího algoritmu do implementace ve specifikovaném programovém jazyce.

$$T = E / (f * S) \tag{16}$$

Veličina S vychází z času, který potřebuje lidský mozek k provedení nejzákladnějšího rozhodnutí. Nazývá se Stroudovo číslo S (dle Johna M. Strouda) jsou to okamžiky (krátké časové úseky) za sekundu. Zde je použito S = 18 momentů / sekundu. Pro převod na minuty je ve vzorci dále použita veličina f = 60. [26]

- **Měřítko-invariantní složitosti architektury**

Informační systém je složen z několika provázaných vrstev. Jedná se tedy o víceúrovňový systém, kde je složitost ovlivněna zvolenou architekturou. Složitost této architektury lze měřit pomocí měřítka invariantní složitosti. Pro definici měřítka invariantní složitosti je třeba definovat matematicky praktickou abstrakci víceúrovňové architektury. K tomuto účelu lze využít rekurzivní grafy, které mohou být považovány za soubor vnořených grafů v různých měřítcích. [16] Příklad vnořených grafů je na Obrázek 16



Obrázek 16: Vnořené grafy

Zdroj:[30]

Dále je třeba určit operace přiblížení a oddálení na rekurzivním grafu, protože při oddálení se malé detaily v malém měřítku rozmazávají. Po přiblížení se detaily opět objeví. Dále se předpokládá, že každý uzel rekurzivního grafu je charakterizován jednotkovými náklady složitosti – váhou. Tato váha musí splňovat podmínku kompatibility, kdy váha uzlu v daném měřítku by měla být součtem vah uzlů (v menším měřítku), které obsahuje.

Definice měřítka-invariantní složitosti je pak následující:

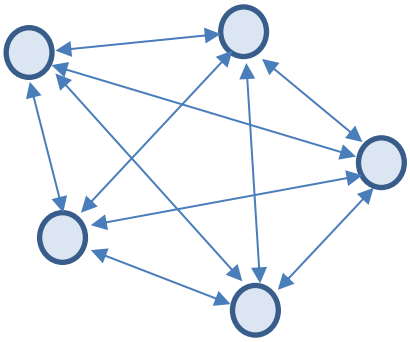
Nechť v_j označuje j-tý uzel v nejmenším měřítku v grafu s více měřítky. Nechť $(v_1 \dots v_p)$ je cesta délky p na těchto uzlech. Nechť $w_n(v_j)$ označuje váhu uzlu v_j. [16] Explicitní vzorec pro měřítko-invariantní složitosti je pak definován pomocí součtu nad takovými cestami délky p:

$$\text{měřítko invariantní složitosti} = \sqrt[p]{\sum_{cesta} w_n(v_1) \dots w_n(v_p)} \qquad (17)$$

Důkaz, že tento výraz je skutečně měřítko invariantní složitosti při výše uvedených operacích přiblížení a oddálení, závisí na elementární kombinatorické analýze. Pro získání lepší představy o tom, co měřítko invariantní složitosti představuje, jsou níže uvedeny dva extrémní případy. [16]

Špagetová struktura

Jako měřítko maximálně „špinavé“ IT architektury lze považovat tzv. špagetový graf s ***n*** uzly, které jsou všechny spojeny se všemi ostatními. Takovouto strukturu popisuje Obrázek 17.



Obrázek 17: Špagetový graf

Zdroj:[30]

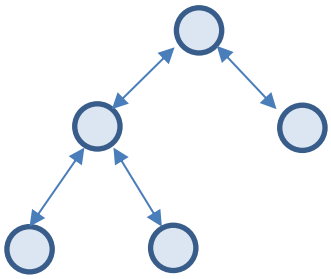
Při zvolení ***p*** a váhy w_i**= 1** na všech uzlech, je výpočet, vzhledem k tomu, že každý uzel ***v*** v cestě $(v_1,\ldots,v_p)$ může být vybrán samostatně, takovýto:

$$\text{Měřítko invariantní složitosti (špagetový graf)} = (n^p * 1)^{1/p} = n \qquad (18)$$

Složitost se tedy rovná počtu všech uzlů grafu.[16]

Hierarchický graf

Pro srovnání je nyní použita hierarchický struktura s ***n*** vrcholy, která je na první pohled více organizovaná, než je tomu v případě špagetového grafu, což je patrné při pohledu na Obrázek 18.



Obrázek 18: Hierarchický graf

Zdroj:[30]

V případě, že je výchozí uzel v_1 vybrán (existuje ***n*** možnosti) každý uzel ***v*** v cestě $(v_1,\ldots,v_p)$ může být vybrána nejvýše čtyřmi různými způsoby. Z tohoto plyne vztah:

$$\text{Měřítko invariantní složitosti (hierarchický graf)} = (n^p * 4^p * 1)^{1/p} = 4n^{1/p} \qquad (19)$$

Při použití **n = 100** a **p = 2** vyjde u špagetového grafu složitost **100**, zatímco složitost hierarchického grafu je **40**. To tedy potvrzuje skutečnost, že špagetová architektura je složitější než hierarchická. [30]

2.5.6. Technologická infrastruktura

Dimenze zahrnuje určení komponent a parametrů technologické infrastruktury, na které bude IS provozován. Infrastruktura zahrnuje technologické platformy což je souhrn hardwaru, základního softwaru a služeb navržených tak, aby umožnily provoz aplikací. Součástí infrastruktury je i síťová infrastruktura, zajišťující propojení počítačových systémů do sítí a dále systémy pro řízení infrastruktury. Technologickou infrastrukturu lze chápat jako jednu z vrstev IS, a to pod vrstvou aplikační, jednotlivé komponenty je možno měnit či doplňovat dle potřeby bez nutnosti přerušit fungování IS. [4]

Vztah ke komplexitě IS

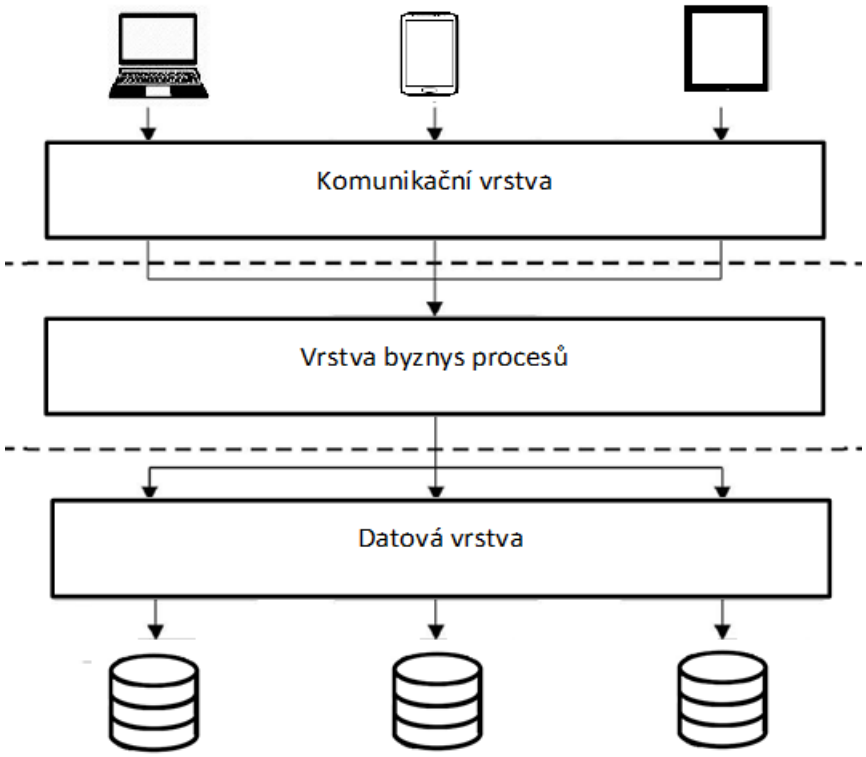
Vzhledem k tomu, že životní cyklus technologické infrastruktury se od životního aplikace liší, je vhodné ji navrhovat tak, aby při výměně infrastruktury, nemusela být upravována aplikace. Obecně lze tvrdit, že čím více různých technologií, tím více různých zdrojů změn v infrastruktuře, a tím častěji je třeba měnit nějakou část systému.

To významně zvyšuje komplexitu a rizika potíží při provozu systému plynoucí z nedostatečného zvážení dopadu změn.

2.5.7. Uživatelské rozhraní

V této dimenzi jsou určeny a definovány komunikační kanály, kterými uživatel bude moci využívat funkce IS a určení formy uživatelského rozhraní pro každá kanál. Komunikačním kanálem může být PC, mobilní zařízení, operátor helpdesku. Každý kanál má určité výhody a nevýhody. Je třeba zvážit počet komunikačních kanálů s ohledem na finanční náročnost každého z nich.[4] Pro udržení nákladů v přijatelné úrovni využívá se třívrstvá architektura aplikace znázorněná na Obrázek 19.

Jednotlivé vrstvy jsou: komunikační, vrstva byznys procesů, a datová. Řešení uživatelského rozhraní je soustředěno do komunikační vrstvy. Každý modul komunikačního rozhraní realizuje jeden komunikační kanál s aplikací. Vrstva byznys funkcí je reprezentována programovými moduly, které realizují byznys logiku aplikace. Jejich funkcionalita je volána standardně ze všech komunikačních rozhraní. Vrstva datových funkcí obsahuje moduly, které realizují datové operace v konkrétním systému řízení báze dat.[4]



Obrázek 19: Třívrstvá architektura aplikace

Zdroj: upraveno dle [4]

Po určení komunikačních kanálů aplikace následuje definice vlastního uživatelského rozhraní a popis průběhu komunikace. Jednoduchost rozhraní má zásadní dopad na komplexitu systému a jeho budoucí provoz.

Vztah ke komplexitě IS

Většina interakcí s IS se provádí prostřednictvím uživatelských rozhraní, jejichž použitelnost je přímo spojena s vnímáním větší či menší jednoduchosti uživateli. IS hraje roli poskytovatele služeb, který by měl poskytovat informace rychle, spolehlivě a očekává se, že bude náhradou tradičního papírování. Předpokládá se, že IS rychle reaguje na měnící se požadavky.

Zatímco snižování složitosti potlačením funkcí nebo služeb ze systému, tak jak bylo napsáno v dimenzi o funkcích a procesech je objektivní transformace, existují další subjektivnější způsoby, jak zvýšit jednoduchost. Skrytí složitosti může být užitečné pro uživatele a často souvisí s odstraněním jeho omezení. Příkladem může být jednotné přihlášení do několik aplikací. V obecné rovině platí, že počítače nejsou nic jiného než stroje pro skrytí komplexních výpočetních úkolů. Skrývání složitosti s sebou nese náklady v podobě práce programátorů a designérů. Ergonomie grafického rozhraní vyvolává pocit okamžité znalosti a není třeba usilovného učení.[16]

Jak K-složitost, tak logický hloubka se pravděpodobně při skrývání komplexity zvýší. Popis systému bude delší a skrytí jeho složitosti vyžaduje více práce při návrhu, a tím i větší náklady. Poměr mezi zvýšením těchto nákladů a přínosy je třeba důkladně zvážit.

Skrývat složitost před zákazníky je životně důležité pro společnost, která prodává nebo inzeruje zboží či služby prostřednictvím on-line platformy. Pro zajištění toho, aby se zákazníci vraceli, je třeba vše nakonfigurovat tak, aby se cítili pohodlně. Zvláštní pozornost je třeba věnovat zajištění snadné dostupnosti všech důležitých informací. Sebemenší obtíže při hledání informací může mít za následek, že zákazníci opustí web.[16]

Pomocí učení, které bylo zmíněno v dimenzi týkající se personálu, se lze vyhnout potřebě skrýt složitost. Příkladem skrývání složitosti z oblasti softwarové architektury jsou rámce, kdy rámce nejsou nic jiného než kusy softwarové architektury, které řeší dobře identifikované a opakující se problémy s konstrukcí, které by neměly být znovu řešeny v každém projektu. Složitost skrytá v takových rámcích je často poměrně velká (až desítky tisíc řádků kódu).

V IT však existují četné situace, kdy používání rámců je zbytečné a příliš složité, a to tím spíše, když jsou použity pro řešení relativně jednoduchých problémů. Z čehož plyne, že znalost například explicitních SQL dotazů nebo toho, jak převést data z tabulek do grafů je stále užitečné a cenné. Často lze skrývání složitosti vyměnit za učení lidí, a to i s úsporou nákladů. Použití rámců je však v mnoha případech opodstatněné a nutné, neboť v případě řešení mnoha složitých problémů, by ruční tvorba kódu vyžadovalo nerealistické množství odborné znalosti a bylo by to velmi pracné a nákladné.[16]

Obrázek 20: Multiplikativní růst komplexity

Zdroj: upraveno dle [15]

Při provozu IS dříve či později vznikne tlak na přidání dalších funkcí. Je třeba ověřit, že významný počet uživatelů skutečně potřebuje další funkce, protože každá další funkce zvyšuje složitost softwaru. Navíc je to další věc, kterou se uživatelé musí naučit. Ve skutečnosti v systému nepřibude jeden prvek, protože každý nový prvek interaguje s ostatními, což vede ke zvyšování složitosti.[15] Z tohoto vyplývá, že komplexita neroste lineárně, ale multiplikativně tak jak je to na Obrázek 20.

2.5.8. Bezpečnost a kvalita

Cílem je zajistit přiměřenou bezpečnost a další kvalitativní charakteristiky aplikace. Kvalita je úměrná nákladům a je třeba nároky na kvalitu klást s ohledem na omezený rozpočet. Avšak nedostatečná kvalita práce může znehodnotit precizní návrhy v ostatních dimenzích. Kvalitní systém je takový, který splňuje uživatelské požadavky a funguje bezpečně a spolehlivě. K porušení kvality může dojít v jakékoliv komponentě IS a kterákoliv činnost při vývoji a provozu aplikace. Kvalita aplikace je řešena ve všech fázích vývoje a provozu aplikace. Kvalita aplikace je ověřována metrikami, jako je dostupnost, doba odezvy nebo doba opravy vzniklého incidentu. Jsou definovány ICT procesy identifikující chybové stavy a tyto odstraňují nebo jim předchází. S kvalitou také souvisí bezpečnost systému. V rámci analýzy bezpečnosti jsou analyzovány rizika, která mohou bezpečnost ovlivnit a pro rizika, která jsou významná, jsou definována opatření. Hlavními nástroji, jak zabránit neoprávněným osobám využívat funkce a data IS je definice a kontrola přístupových práv, která vymezují oprávnění uživatelů a funkcí, které mohou v rámci IS využívat.[4]

Vztah ke komplexitě IS

Aspekt bezpečnosti v kontextu komplexity je velmi důležitý. Neboť můžeme buď důvěřovat lidem a složitost systému nezvyšovat bezpečnostními prvky, nebo důvěřovat systému. Vzhledem k tomu, že nelze stoprocentně věřit lidem například, že nebudou zcizovat peníze z bankovního účtu nebo využívat soukromou komunikaci pro svůj vlastní účel, je třeba zabezpečit informační systém. Technická následky této skutečnosti vedou k otázce IT bezpečnosti a kryptografie, která je jedním z nejobtížnějších oblastí IT. Vzhledem k tomu, že zůstává všeobecný nedostatek důvěry v lidské chování, je jedna z vrstev IS bezpečnostní.

Nedostatek důvěry vyžaduje navržení mnoha ověřovacích mechanismů[16]:

- kontrola přístupových práv uživatele,
- identifikace a autentizace uživatele,
- prokazatelné doručení zpráv,
- kontrola zachycení nebo upravení soukromé zprávy,
- kontrola podpisu zprávy, zda pochází z určitého zdroje.

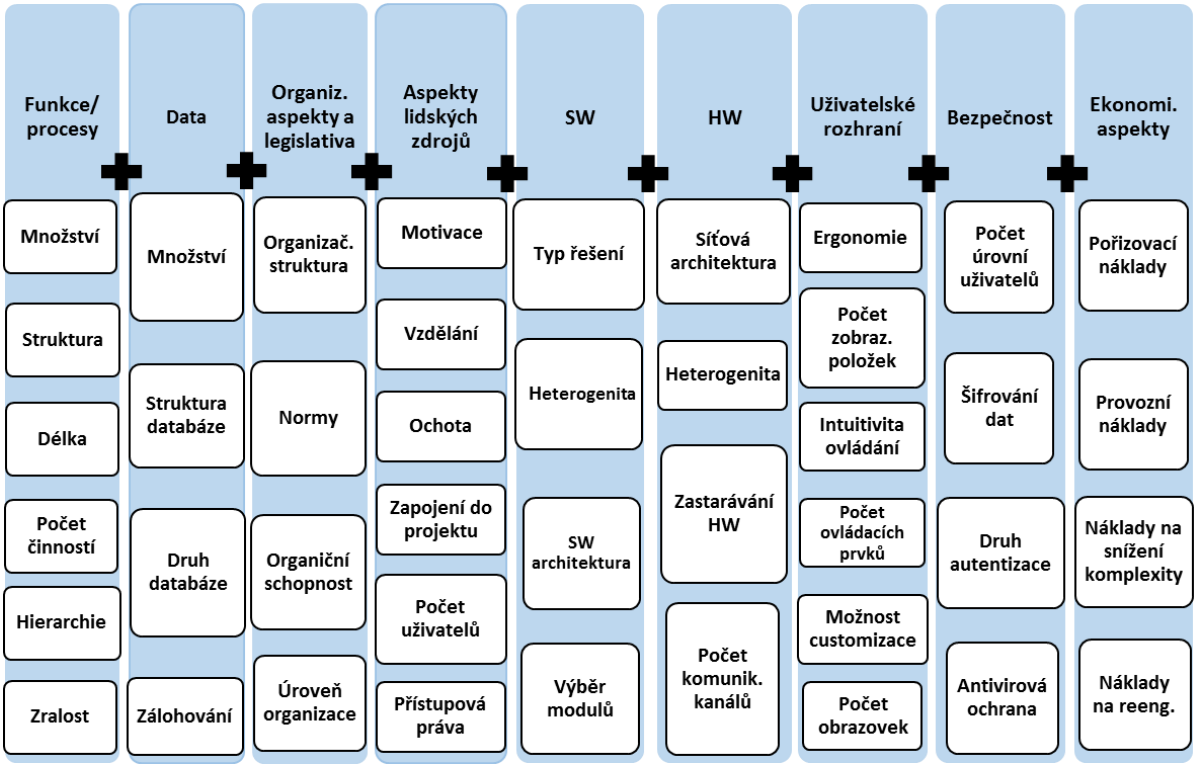
Výše zmíněné mechanismy jsou každému běžnému uživateli dobře známy a vzhledem k tomu, že IS jsou určeny pro běžné uživatele, o kterých nelze tvrdit, že jsou stoprocentně spolehliví, jsou pro moderní IS nepostradatelné. Vlastní složitost IS se zvyšuje s množstvím bezpečnostních prvků, což s sebou nese také vyšší náklady. Ty jsou však vyváženy snížením rizika škod způsobených například zneužitím obchodních informací.

2.5.9. Ekonomické a finanční aspekty

Ekonomická dimenze v sobě zahrnuje sledování finančních nákladů na tvorbu a provoz IS a přínosy podniku z jeho užití. Je určeno, v jakém období a jakých zdrojů budou náklady na IS kryty. Hlavním výstupem této dimenze je určení, které podnikové aktivity mají být prioritně podpořeny funkcemi IS, aby bylo dosaženo maximálních ekonomických efektů. Sledovat je třeba investiční náklady, operativní náklady na tvorbu IS (vývojové nástroje, spotřební materiál, práce analytiků…), operativní náklady na provoz (provoz a údržba technologické infrastruktury, upgrade SW, komunikační náklady, práce provozního personálu…).[4]

Vztah ke komplexitě IS

Vzhledem k tomu, že komplexita má přímí dopad na náklady spojené s vývojem i provozem IS je třeba důkladně zvažovat, jak velké náklady budou vynaloženy na to, aby byla složitost systému co nejnižší. Práce analytiků a programátorů patří mezi ty největší položky v rozpočtu projektu vývoje IS. Vyladění systému a odstranění nadbytečné komplexity je z pohledu investora vzhledem k termínům projektu často zanedbáváno. Investor také obvykle nemá zájem vynakládat dodatečné finance na optimalizaci SW, což může znamenat v budoucnu pro podnik větší náklady skryté v podobě pracnější údržby nebo nižší efektivity zaměstnanců.



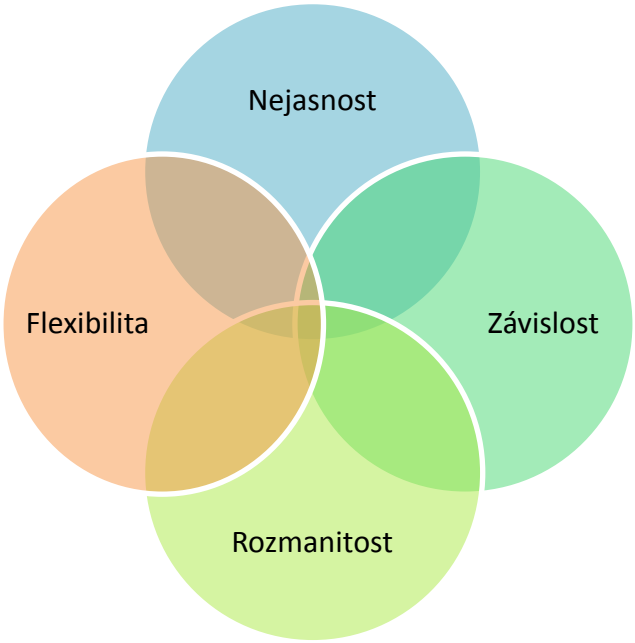
Obrázek 21: Prvky komplexity obsahových dimenzí IS

Zdroj: vlastní zpracování

Jednotlivé prvky komplexity v rámci všech obsahových dimenzí shrnuje Obrázek 21, který znázorňuje zdroje komplexity v rozdělení dle obsahových dimenzí, ke kterým jsou tyto zdroje vztaženy. Výsledná komplexita IS je přímo závislá na těchto prvcích.

2.6. Zdroje komplexity IS v podniku

Následující kapitola bude zaměřena na přehled největších zdrojů komplexity informačního systému v podnicích. Většina z nich byla zmíněna výše v rámci kapitoly o obsahových dimenzích a životním cyklu IS. Lze definovat čtyři interakční aspekty složitosti systému: rozmanitost, nejasnost, vzájemná závislost a reakce na změny. Vztah těchto aspektů ilustruje Obrázek 22.



Obrázek 22: Zdroje složitosti IS v podniku

Zdroj: upraveno dle [1]

Rozmanitost nebo také různorodost lze považovat se za jeden z klíčových faktorů složitosti. Rozmanitost je vlastností velkých systému s několika podsystémy. Lze ji definovat jako schopnost systému zahrnout určitý počet různých stavů v daném časovém rozpětí.

Zdrojem nejasnosti v podnicích můžou být nejasně definované organizační cíle nebo úkoly stejně jako neschopnost odhadnout budoucí vývoj, informace pak nejsou úplné nebo nejsou platné, což vede ke složitosti [25]. To může vézt k nejasnosti rolí v podniku a následné snížení produktivity práce.

Uvnitř každé organizace existuje vzájemná závislost v rámci jeho organizační struktury, tyto silná spojení slouží k přenosu informace. A právě struktura těchto vazeb přímo a zásadním způsobem ovlivňuje komplexitu systém. Závislosti mezi moduly IS mají také velmi důležitý dopad na provoz systému. Vlivem závislosti modulů může při chybě jednoho z nich, dojít ke zhroucení závislých modulů a tím například ke ztrátě dat.[1]

Prvek flexibility souvisí se změnou v organizaci v závislosti na změně prostředí. Ke změnám v prostředí dochází v případě změn na trhu nebo při změně legislativy. To, jak dokáže systém na změny reagovat, ovlivňuje jeho složitost. Zavádění těchto změn je třeba provádět s cílem nezvyšovat složitost samotné organizace a IS. [1]

3. ANALÝZA DOPADŮ KOMPLEXITY DLE ÚROVNĚ ABSTRAKCE

Sled fázi životního cyklu lze rozdělit dle úrovní abstrakce na konceptuální, technologickou a implementační úroveň tak jak je to na Obrázek 2. V každé úrovni je třeba zabývat se všemi obsahovými dimenzemi, nicméně na každé úrovni je pohled na dimenze odlišný. Na každé úrovni je realita modelována pomocí vybraných grafických nástrojů-diagramů. Pro objektově orientované jazyky je určena UML notace diagramů modelování informačního systému. Níže bude pro každou úroveň pomocí upravené SWOT analýzy určeny dopady vysoké a nízké komplexity a určeny metriky pro kvantifikaci komplexity diagramů patřících do dané úrovně.

3.1. Konceptuální úroveň

Prvním krokem vývoje IS je vytvoření modelu reality (konceptuální úroveň), tedy vytvoření modelů procesů, které mají být do systému zavedeny. Před automatizováním procesů pomocí informačního systému je třeba se nejdříve zaměřit na vlastní procesy a posoudit jejich komplexitu ještě před jejich zavedením do systému. Pro posouzení komplexity procesů lze přistupovat několika způsoby.

Lze posuzovat proces dle obtížnosti jeho popisu. O složitosti procesu vypovídá také obtížnost jeho vytvoření, s tímto souvisí již definovaná K-složitost a logická hloubka. Obtížnost popisu organizační struktury a množství informace rozdělená mezi jednotlivé části určuje stupeň organizovanosti procesu. Mezi metody pro určení složitosti organizační struktury procesu patří hierarchická komplexita, délka schématu, sofistikovanost nebo algoritmus vzájemné výměny informací.

Jako metriku pro určení složitosti procesů lze využít MCC a control-flow komplexitu (CFC). Pro výpočet obou je třeba proces převézt do podoby control-flow grafu. Princip MCC byl vysvětlen v kapitole 2.5.5. a CFC v kapitole 2.5.1.

Vysoká komplexita procesů ovlivní jejich model a složitost se následně promítne do dalších fází vývoje systému. Dopady vysoké a nízké komplexity konceptuálního modelu jsou uvedeny v Tabulka 3 níže, která je obdobou SWOT analýzy, kde S – „Strenghts“ jsou výhody nízké komplexity, W – „Weaknesses“ jsou slabiny vysoké komplexity, O – „Opportunities“ jsou příležitosti plynoucí z optimální komplexity a T – „Threats“ jsou hrozby plynoucí z vysoké komplexity.

Tabulka 3: Dopady komplexity v konceptuální úrovni

Konceptuální úroveň	
Výhody nízké komplexity	**Nevýhody vysoké komplexity**
• Spolehlivost procesů • Srozumitelnost procesů • Efektivita pracovní síly • Přehlednost databáze • Nízká redundance dat • Nízké nároky na HW • Snadné zálohování	• Nepředvídatelnost procesů • Chybovost procesů • Zdlouhavá analýza procesu • Vysoké nároky na HW
Dopady nízké komplexity	**Dopady vysoké komplexity**
• Zvýšení efektivnosti podniku • Zrychlení procesů • Snížení náročnosti procesu • Úspora nákladů	• Nesplnění cílů projektu IS • Odstoupení od projektu

Zdroj: vlastní zpracování

V Tabulka 3 výše jsou uvedeny aspekty týkající se vysoké a nízké komplexity v konceptuální fázi. Výstup z konceptuální úrovně by, měl obsahovat pouze prvky a vazby nezbytné pro zajištění všech žádaných funkcí na podporu procesů definovaných v rámci informační strategie. V modelu by měla být obsažena pouze **nezbytná komplexita.**

V této úrovni je model systému vytvořen v případě UML pomocí diagramu užití včetně detailních scénářů jednotlivých případů užití. Tyto digramy zachycují datovou a funkční podstatu budoucího systému. Pro podporu rozhodování, zda vytvořené diagramy mají vysokou či nízkou komplexitu je třeba ji kvantifikovat pomocí metriky.

3.2. Technologická úroveň

V technologické úrovni je vytvořen technologický model. V rámci tohoto jsou vytvořeny jednotlivé funkcionality, jsou vymezeny uživatelské role, je definován hardware, datová struktura a grafické uživatelské rozhraní. Technologický model je tvořen již na vybrané softwarové řešení. Dalším krokem je vlastní naprogramování systému přesně dle technologického modelu. Dopady vysoké a nízké komplexity technologického modelu jsou uvedeny v Tabulka 4 níže, které je opět modifikovanou SWOT analýzou, tak jak tomu bylo v konceptuální úrovni.

V technologické úrovni jsou vytvářeny modely přímo na zvolené technické řešení. Mezi tyto modely patří RMD diagram, který je vytvářen z diagramu tříd a dle kterého je vytvořena relační databáze. Jako metriku určující komplexitu databáze lze požít tzv. DC metriku uvedenou v kapitole 2.5.2 . Strukturu systému lze popsat také diagramem tříd. Metriky pro kvantifikaci diagramu tříd byly popsány v kapitole 2.5.2. Funkcionalita systému může být popsána sekvenčním diagramem. V této úrovni je navržena vlastní architektura IS. Složitost architektury je jedním z nejzásadnějších aspektů, určující komplexitu systému. Pro kvantifikaci této složitosti lze využít měřítko invariantní složitosti, které je popsáno v kapitole 2.5.5.

Vysoká komplexita diagramů popisujících chování systému se projeví zejména složitostí uživatelského rozhraní a dobou zpracování procesu systémem a vysokými hardwarovými nároky. Nezanedbatelný dopad na následný provoz aplikace má nadbytečná složitost uživatelského rozhraní, neboť má přímý vliv na práci zaměstnanců. Musí-li pracovník provést příliš mnoho operací (například kliknutí nebo přihlášení), stává se jeho práce neefektivní a rostou náklady na jeho práci, neboť stráví vykonáním daného úkonu více času, než je nutné. Vysoká složitost uživatelského rozhraní prodlužuje dobu zácviku nových zaměstnanců a může vézt k odporu zaměstnanců systém používat. Také v případě, že proces vyžaduje vysoké nároky na hardware (velký počet operací, které musí výpočetní technika vykonat) má negativní vliv na efektivitu práce (dlouhé odezvy systému) a zvyšuje nestabilitu systému (hrozí přetížení výpočetní techniky), proto musí organizace adekvátně dimenzovat použitý hardware, což opět zvyšuje náklady.

Tabulka 4: Dopady komplexity IS v technologické úrovni

Technologická úroveň	
Výhody nízké komplexity	**Nevýhody vysoké komplexity**
• Optimálně zvolená IT architektura • Intuitivní uživatelské rozhraní • Optimální velikost databáze • Optimální množství funkcionalit • Automatizace vybraných procesů • Zjednodušení vybraných procesů • Optimální zabezpečení informací • Vyvážená hardwarová infrastruktura	• Nepřehledné uživatelské rozhraní • Složitá datová struktura s redundancí • Nepotřebné funkce • Vysoká heterogenita technologické infrastruktury • Vysoká pracnost následné implementace • Náchylnost k chybám
Dopady nízké komplexity	**Dopady vysoké komplexity**
• Rychlý zácvik zaměstnanců • Snížení nákladů na zaměstnance • Úspora nákladů za HW	• Odpor zaměstnanců využívat IS • Neefektivita pracovní doby zaměstnanců • Chyby aplikace a dlouhá odezva

Zdroj: vlastní zpracování

3.3. Implementační úroveň

Implementační úroveň zahrnuje vlastní programování a následné testování softwaru. Dále je na této úrovni provedeno zavádění systému do praxe. Možnosti zavádění systému byly uvedeny v kapitole 2.4.3 Při výběru postupu zavádění systému je třeba brát ohled na jeho uživatelskou složitost. Nárazová strategie je vhodná pro systémy s nízkou mírou složitosti uživatelského rozhraní, kdy se uživatel v systému rychle zorientuje a riziko chyb je minimální. Postupné a pilotní zavádění je vhodné u komplexnějších systémů, kdy je složitost větší, a proto je uživateli „dávkována“ postupně a ten má čas se na systém adaptovat. Pro velké systémy je ze zmíněných metod zavádění nejvhodnější strategie souběžného provozu. Je však bezpodmínečně nutné mít zacvičování uživatelů pod kontrolou, neboť hrozí, v důsledku

existence starého systému, odkládání učení se nového systému, což prodlužuje souběžný provoz, který je nákladný.

Dopady vysoké a nízké komplexity na implementační úrovni jsou uvedeny v Tabulka 5 níže, tabulka je sestavena stejným způsobem jako v konceptuální a technologické úrovni.

Tabulka 5: Dopady komplexity v implementační úrovni

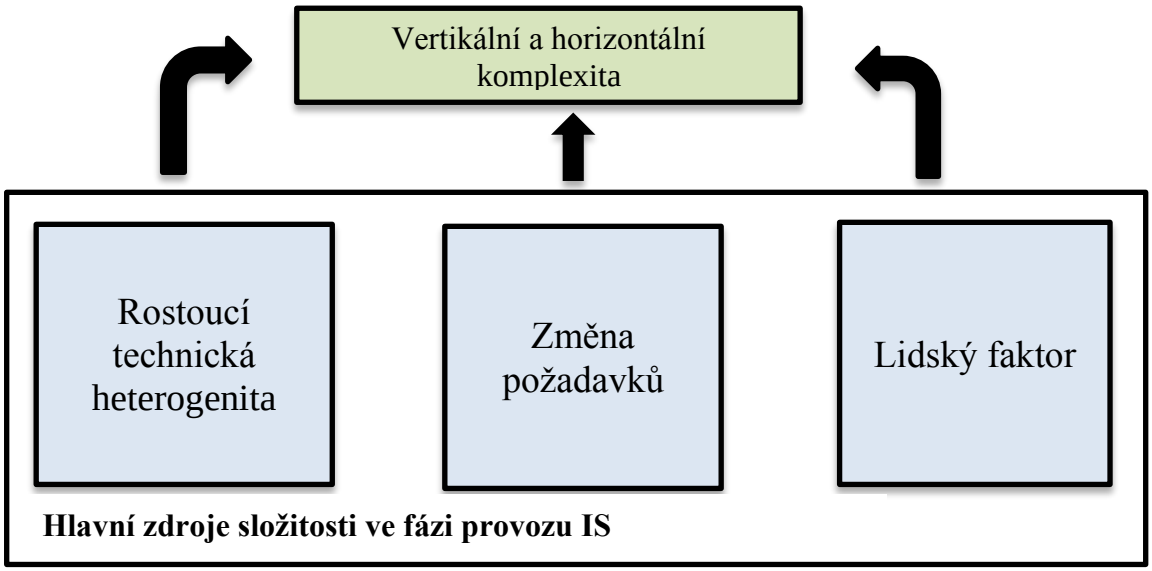
Implementační úroveň	
Výhody nízké komplexity	**Nevýhody vysoké komplexity**
• Přehlednost kódu • Snadná údržba • Rychlý zácvik zaměstnanců • Přehledná dokumentace • Rychlé testování a zavádění • Není nutno komplexitu skrývat	• Obtížná údržba • Obtížná nebo nemožná modifikovatelnost programu • Vysoké nároky na uživatele • Zdlouhavé zavádění a zácvik uživatelů
Dopady nízké komplexity	**Dopady vysoké komplexity**
• Rychlé a spolehlivé nasazení nového systému • Úspora nákladů na údržbu a optimalizaci	• Chyby při modifikaci • Chyby při údržbě • Výpadky systému

Zdroj: vlastní zpracování

Vysoká složitost systému má také vliv na dobu programování a zejména testování. V systému, kde existuje mnoho prvků a vazeb s vnitřními závislostmi je na otestování všech variant třeba dostatek času pro snížení pravděpodobnosti chyby v reálném provozu. To s sebou nese opět vysoké náklady. Nízká komplexita systému nese riziko podcenění doby testování i doby potřebné pro zaučení pracovníků. To může způsobit chyby v provozu relativně jednoduchého systému. Pro měření složitosti vlastního kódu je možno použít metriky uvedené v kapitole 2.5.5, tedy McCabeovu cyklomatickou komplexitu a Halsteadovi metriky. Udržení složitosti kódu na co nejnižší možné úrovní má zásadní dopad na jeho budoucí udržovatelnost a modifikovatelnost.

3.4. Provoz a údržba

Po zavedení systému následuje časově nejdelší etapa života informačního systému, a to jeho rutinní provoz, který je spojený s pravidelnou údržbou. Komplexita informačního systému v čase roste. Toto tvrzení lze všeobecně považovat za platné. Za 3 hlavní zdroje nekontrolovaného zvyšování komplexity během provozu lze označit: rostoucí technickou heterogenitu, měnící se požadavky a soubor lidských faktorů. Těmto třem zdrojům se bude práce blíže věnovat níže. Všechny tři aspekty se podílí na zvyšování horizontální i vertikální komplexity systému tak, jak to znázorňuje Obrázek 23.



Obrázek 23: Zdroje zvyšování složitosti v čase

Zdroj: upraveno dle [16][4]

3.4.1. Rostoucí technická heterogenita

Jedná se o pravděpodobně nejzřejmější příčinu nekontrolované složitosti IT. Tento aspekt vychází ze skutečnosti, že různé technologie nebo nástroje lze používat společně pro podobné účely. Tato heterogenita se může týkat operačních systémů, obchodních aplikací nebo programovacích jazyků používaných pro konkrétní vývoj softwaru.

Příklady rostoucí technické různorodosti:

- použití a propojení různých operačních systémů (Windows, Linux a Mac OS) je velmi běžným zdrojem IT heterogenity,
- použití různých programovacích jazyků (Java, .NET nebo Cobol) pro konkrétní vývoj,
- využití různých nástrojů pro spolupráci v jedné společnost (Microsoft Exchange, Google Apps nebo Lotus Notes),

- middleware a software pro infrastrukturu,
- koexistence několika verzí stejného produktu, programovacího jazyka, operačního systému nebo aplikace.[16]

Horizontální složitost

Tímto pojmem lze označit situaci, kdy spolu existuje několik ekvivalentních technologií, které provádějí stejné úkoly. Úkoly mohou být buď technické, nebo obchodní.

Vertikální složitost

Vertikální složitost je situace, kdy jsou technologie hierarchicky vnořené do sebe. Taková situace je obvykle následkem iterativního skrývání složitosti, kdy nová vrstva architektury obtéká předchozí v pokusu skrýt složitost.

Důsledky technické heterogenity:

- zvyšuje se rozsah potřebných technických dovedností na udržení IS v provozu,
- zvyšuje se množství čistě technického kódu, který nemá obchodní hodnotu,
- zvyšují se pracovní nároky údržbu IT,
- komplikuje vzájemnou závislost mezi subsystémy, čímž je celkový systém méně předvídatelný a obtížnější udržovatelný.

Další častou příčinou rostoucí heterogenity je absence technologické vize, která je často důsledkem rychlosti, jakou se IT vyvíjí.[16]

Bez vize a řízení dochází k tomu, že i začátečníci v IT, zůstávají bez vedení a koordinace a mohou svobodně zvolit technologie. Za takových okolností neexistují nebo nejsou respektovány technologické standardy. Technologická řešení jsou pak volena projekt od projektu bez jednotné koncepce, což znamená, že se komplexita technologické infrastruktury zvyšuje.[16]

3.4.2. Změna požadavků

V průběhu provozu dochází vždy dříve či později ke změnám požadavků na systém, které byly definovány při jeho vývoji. Zde se dostává do konfliktu dva požadavky, a to, aby byl systém optimalizován a zároveň adaptibilní. Důvod konfliktu je prostý, a to, že systémy, které jsou optimální, obvykle nejsou flexibilní. V kontextu flexibility systému roste složitost dvěma způsoby. Prvním je, kdy nebylo s větší flexibilitou systému počítáno při vývoji a je třeba zpracovávat větší množství dat, než se původně očekávalo. To si vyžádá nove prvky a struktury,

které je třeba do systému zavézt. Nové prvky s sebou nesou nové mechanismy manipulace, které zajistí, že předložené údaje jsou platné a zachová se jejich celistvost při zpracování. Všechny tyto nové prvky a mechanismy vnášejí do systému nepředvídatelnost a zvyšují jeho složitost. Druhým případem je, že bylo při vývoji počítáno s větší flexibilitou než, která byla následně skutečně využita. Zavádění flexibility rozhodně vyžaduje další projektové úsilí. V případě, že nebyla tato flexibilita potřeba, byla tato práce zbytečná a došlo pouze ke zvýšení složitosti systému a plýtvání penězi.[16]

3.4.3. Lidské faktory

Hodnocení složitosti IS je obtížný a nejednoznačný úkol, protože technické aspekty IT jsou hluboce zapletené s těmi lidskými. Pod lidskými faktory si lze představit: koordinaci a řízení týmů, celoživotní učení se novým technologiím nebo vztahy se zákazníky. Rozsah sociálních a kognitivních dovedností, který hraje významnou roli při formování vývoje IS, je široký. Tři důležité lidské faktory, které mohou výrazně přispět k nekontrolovanému nárůstu složitosti, jsou: **multidisciplinarita dovedností, demotivace jednotlivců a skutečnost, že globální zájem je obvykle odlišný od místního zájmu**.[16]

a) **Multidisciplinarita**

Úkol budování a udržování informačních technologií v provozních podmínkách vyžaduje, aby spolupracovalo mnoho různých profesí, pravděpodobně víc než ve většině jiných oblastí. Nekontrolovaná složitost může být důsledkem mnoha nejednoznačností a nedorozumění, k nimž dochází, když si tito lidé s různými dovednostmi, slovní zásobou a zázemím musí vyměňovat informace. Navíc u jednotlivců, zejména u IT specialistů, se obecně očekává, že zvládnou mnoho různých technologií. Když tento počet technologií narůstá, stoupá také náklonost k ledabylému jednání a podcenění znalostí, což zase podpoří nepořádek a nepředvídatelnost.[16]

b) **Demotivace**

Informační systém má velkou vertikální složitost (tj. mnoho vnořených úrovní nebo stupňů abstrakce) a rovněž velkou K-složitost ve většině svých architektonických vrstev (tj. modely jsou rozsáhlé a podrobné). Tato složitost ztěžuje účastníkům vytvářet přehled o systému jako celku. Za takových okolností se rozhoduje o technických nebo funkčních změnách v IS v částečné nevědomosti. Nikdo, dokonce ani v rámci IT oddělení, nemá jasnou představu o vzájemné závislosti různých složek. Výsledkem je, že zbytečná složitost a náhodnost se může dále zvyšovat.

Řešením této situace je zachování konzistentní množiny modelů pro celý systém. Hardwarová infrastruktura, softwarová architektura, podnikové procesy atd. by měly mít aktuální model a tyto by měly být k dispozici v celé společnosti. Toto však v praxi naráží na nechuť účastníků, protože udržet si celkový přehled nad systémem vyžaduje velké úsilí, což většina ze zúčastněných odmítá nebo předem vzdává. Zvládnutí složitosti IS můžou pouze dobře motivovaní jedinci.[16]

S tímto souvisí snaha o skrytí složitosti a automatizace práce se systémem. Skrytí složitosti pro uživatele například použitím rámců však vlastní složitost systému zvyšuje. Demotivovaní lidé pak nakonec dospějí k závěru, že neexistuje reálná naděje v takovém technickém a organizačním nepořádku vyznat. Pro tyto uživatele je pak jednodušší zvyknout si na zdánlivě, nesmyslný systém než se ho snažit pochopit. Takto rezignovaní lidé ztrácí smysl pro jakoukoliv iniciativu a výsledkem je v krajním případě to, že lidé se chovají jako stroje, jen méně spolehlivým způsobem.

c) Místní zájem neodpovídá zájmu globálnímu

Hledání nejlepší strategie pro maximalizaci globálního zájmu společnosti oproti individuálnímu zájmu jedince je obecný sociální problém, kterému čelí všechny lidské uskupení, bez ohledu na jejich velikost. Tento problém se samozřejmě dotýká i oblasti IT. Dva běžné způsoby, jakými se jednotlivé zájmy liší od globálního zájmu, jsou: „syndrom odporu ke změně" na jedné straně a „syndrom hřiště“ na straně druhé.

Konzervativní přístup a odmítání změny je nejpřirozenější způsob reakce těch, kteří vnímají změny především jako ohrožení jejich výsad, pohodlí nebo vlivu. Složitost se pak hromadí jako důsledek toho, že jednodušší řešení nejsou implementována z důvodu strachu, že některé z vlivných zainteresovaných stran, budou muset upravit své návyky.

Na druhém konci psychologického spektra lze najít jedince, kteří jsou technickými inovacemi uchváceni a věří tomu, že nejnovější technologie jsou nutně rychlejší, lepší nebo spolehlivější. Změna technologie nebo nástrojů pro ně je záležitostí radosti. Inovace berou jako „nové hřiště“, které se právě otevřelo a nepřichází v úvahu, aby ho nevyzkoušeli. Složitost se pak hromadí v důsledku rychlého zastarávání technologií, které jsou rychle nahrazovány novými.[16]

3.4.4. Další dílčí komponenty složitosti

Níže budou blíže popsány další dílčí komponenty, které přispívají ke složitosti systému během jeho provozu. Většina vychází z třech výše uvedených, ale jsou zde popsány i nové zdroje komplexity, které jsou způsobeny novými trendy v IT.

a) Zastaralý kód

Stále existuje mnoho systémů, které používají starší hardware nebo vlastní kód. Software byl vyvinut pomocí tehdejších jazyků jako je COBOL, FORTRAN nebo PASCAL a postupem času, tyto programy přestávají být užitečné pro podnikání a musí být obsluhovány a spravovány za vysoké náklady kvůli nedostatku podpory, která je dále komplikována chybějící dokumentací. [25] Starší kód a systémy vytvářejí další problémy z důvodu nedostatku jednotlivců, kteří jsou schopni spravovat tyto systémy při normálním provozu nebo při migraci.

b) Legislativa

Informační systémy také podléhají vysokému počtu nařízení, které mohou být specifické pro jednotlivé státy nebo odvětví. Tyto předpisy se také často mění a stávají se přísnějšími a složitějšími, což se odráží i v IS.

c) Chybějící dokumentace

Informační systémy často trpí chybějící, zastaralou nebo neúplnou dokumentací. Toto je velmi problematické při provádění změn v systému, což může také vytvářet bezpečnostní rizika. Mnoho společnosti udržují databáze pro správu změn, nicméně přesnost databáze závisí na tom, kdo ji aktualizuje a na frekvenci aktualizace.[1]

d) Řízení změn

Informační systémy mohou být změněny z jednoho nebo více důvodů, a to například z důvodu opravy chyb, přidání nových funkcí nebo nahrazení zastaralého hardwaru nebo nepodporovaného softwaru. Toto jsou složité úkoly, a i když se provedení změn testuje tak žádné množství testů není 100 % zárukou úspěchu, a ne všechny scénáře lze testovat kvůli omezením jako je množství finančních zdrojů nebo času.[1]

e) Bezpečnost informací

Všechny informace v informačním systému musí splňovat podmínky důvěrnosti, bezúhonnosti, dostupnosti a pravosti. Probíhá neustálý boj mezi tvůrci zabezpečení a útočníky, kteří používají nové metody útoků a hacků. Asociace pro audit a kontrolu informačních systémů

(ISACA) identifikovala 5 domén a 80 oblastí pro řízení bezpečnosti IT. Blíže je tato problematika probrána v dokumentu CISA[2].[1]

f) Globalizace

Informační systémy jsou důležitou součástí globálního obchodu. To zvyšuje složitost podpory, který je třeba 24 hodin denně 7 dní v týdnu i přes řadu regionálních a kulturních hranic, jazykových bariér a legislativy. Očekává se, že informační systémy budou běžet nepřetržitě navzdory všem překážkám, které z globalizace plynou.[1]

g) Nové metody a výpočetní technika

Dnes čím dál modernější cloud computing nabízí nové dimenze v informačních systémech, jako je utilita výpočetní techniky ve formě veřejných, soukromých a hybridních cloudů.[1]

h) BYOD[3] („přineste své vlastní zařízení“)

Novou výzvou pro informační systémy je jejich zpřístupnění pro libovolné typy mobilního telefonu, tabletu a počítače. To vyžaduje speciální metody a systémy pro identifikaci těchto systémů a kontrolu přístupu k systémům organizace, což zvyšuje komplexitu IS. [1]

i) Sociální média

Mnoho firem se reprezentuje formou sociální sítě jako je Facebook a Twitter. Některé používají informace shromážděné ze sociálních médií v marketingu, při rozhodování a spolupráci. To vytváří požadavek na informační systémy, které interagují a využívají sociální média jak veřejná, tak soukromá.[1]

j) Řízení

Informační systémy podléhají řadě rámců správy, auditu a zlepšování, jako jsou COBIT, TOGAF, CMMI a SIX-SIGMA. Tyto rámce přicházejí s různými definicemi procesů a stupněm překrývání, který vytváří odpad na IS a dále ho komplikují.[1]

k) Obchodní architektura

Podniky podléhají řadě změn, jako je zlepšování obchodních procesů a jiné strategie závislé na vizích a výstupech z nástrojů, jako jsou SWOT, PESTLE atd. Ty musí být dále využívány formou změn v systémech, pracovních tocích atd., aby bylo lépe vyhověno změněným

[2] CISA – Certified information systems auditor study – příručka pro auditory IS
[3] BYOD – Bring Your Own Device – využití běžně dostupných zařízení pro práci s IS

obchodním požadavkům a prioritám. Toto někdy vede spíše ke vzniku nových architektur než pouze k malým změnám v technologických aplikacích a datech. [1]

Během provozu je z výše uvedených důvodu systém upravován a jeho komplexita přirozeně roste. Růst komplexity, jak bylo ukázáno v kapitole 2.5.7 týkající se uživatelského rozhraní roste multiplikativním způsobem. K jejímu cílenému snižování například v důsledku odstranění nepoužívaných procesů však zpravidla nedochází nebo to není možné například z důvodu existence záznamů v závislých tabulkách, zároveň je tato činnost velmi riziková a nákladná. V případě změny informační strategie může být přikročeno k reengineeringu systému, ten však u systému s vysokou komplexitou není snadný ani levný a je třeba zvážit, zda není výhodnější vyvinout systém nový.

4. URČOVÁNÍ KOMPLEXITY V JEDNOTLIVÝCH ÚROVNÍCH VÝVOJE IS

V této kapitole bude na reálném IS ukázáno určování komplexity pomocí dříve uvedených metrik. Výstupy z metrik mohou sloužit jako podklad pro rozhodování při volbě řešení daného systému, zejména ve fázi analýzy a návrhu. Pro určení komplexity ve fázi analýzy, návrhu a provozu bude požita část modulu PM systému SAP, který je využíván ve společnosti Sev.en EC a.s. Pro porovnání metriky pro určení složitosti zdrojového kódu bude z důvodu nedostupnosti kódu systému SAP využita část kódu aplikace, která je využívána ve stejné společnosti.

4.1. Popis podniku

Společnost Sev.en EC a.s. se zabývá výrobou elektřiny v klasické elektrárně Chvaletice. Provoz se nachází v Polabí nedaleko Pardubic. Byl postaven v letech 1973–1979 na území bývalých Mangan-kyzových závodů, v nichž právě tehdy končila těžba pyritu. S výstavbou elektrárny souviselo dobudování Labské vodní cesty, protože severočeské hnědé uhlí, které se ve Chvaleticích spaluje, sem bylo do poloviny roku 1996 dopravováno z Lovosic po vodě. Celkový instalovaný výkon elektrárny je 820 MW a tvoří ho čtyři 205MW bloky. Výkon je vyveden dvěma 400 kV linkami do rozvodny Týnec nad Labem.

Podnik kromě elektřiny také obchoduje s plně certifikovanými vedlejšími energetickými produkty spalování uhlí (struska, popílek, energo sádrovec, stabilizát) a dodává teplo do přilehlé průmyslové zóny a města Chvaletice.

4.1.1. Modul PM

Modul PM (Plant Maintenance) slouží pro řízení údržby veškeré výrobní i nevýrobní technologie v podniku. V rámci modulu PM lze všeobecně rozlišovat dva druhy údržby**. Preventivní údržbu**, kde jsou tvořeny pečlivé plány, například rok dopředu, a je generován plán údržby. **Korektivní / všeobecná** údržba, vychází z toho, že došlo k závadě na nějakém zařízení a je nutné je opravit, co nejrychleji, aby nedošlo ke zdržení výroby, což je v energetice velmi klíčový faktor, neboť každá hodina výpadku výroby bloku o výkonu 205MW s sebou nese významnou finanční ztrátu, a to zejména z důvodu toho, že vyráběná elektřina je již dopředu zobchodovaná v rámci trhu s elektrickou energií.

Modul PM používají zaměstnanci **provozu a péče o zařízení (PoZ)**. Je využíván pro údržbu veškerého zařízení v elektrárně. Zahrnuje jak údržbu zařízení související s výrobou, tak

i ostatních vybavení jako jsou budovy a komunikace. V rámci oddělení péče o zařízení jsou v systému zavedeny tyto plánovací skupiny: stavební, elektro, kotelna, měření a regulace, odsíření, strojovna, vodní hospodářství a palivové hospodářství. Opravy v podniku probíhají pomocí zakázek. V těch je uvedeno, co je třeba opravit a kdo opravu provede. Údržba je prováděna pomocí externích firem a dodavatelů. Správa zakázek probíhá v PM modulu systému SAP.

4.2. Popis vybraného procesu

Pro analýzu byl vybrán proces opravy závady. Zaměstnanci provozu při zjištění závady na zařízení vystaví hlášení, ve kterém specifikují závadu. Dále vypíší a umístí na místo závady poruchovou visačku. Zaměstnanec PoZ (dle plánovací skupiny) vystaví z hlášení zakázku na údržbářskou firmu. Ta převezme papírovou formu zakázky a jde za pracovníkem provozu, který potvrdí, zda lze na zařízení pracovat (podpis) a v systému změní stav zakázky na „V realizaci“. Po ukončení opravy jde údržbář k pracovníkovi provozu, který provede kontrolu opravy a v případě, že je oprava provedena řádně, provede stvrzení dokončení opravy podpisem zakázky a v systému SAP změní stav zakázky na „Z realizace“. Pracovník údržby následně donese zakázku danému pracovníkovy PoZ, a ten provede ukončení zakázky. Ukončená zakázka je dále zpracovávána v modulu CO. Každá zakázka je vztažena na daný technický objekt.

Pro evidenci technických objektů údržby jsou v systému SAP použita označení „Technické místo“ a „Vybavení“. Každá závada je vztažena k danému „Vybavení“. To představuje konkrétní detailní část zařízení. Každé vybavení má svůj specifický kód KKS. Při tvorbě hlášení je třeba zadat KKS poškozeného zařízení, dle tohoto je v systému automaticky hlášení zařazeno do dané plánovací skupiny danému pracovníkovi PoZ. Příkladem technických míst mohou být větší celky, které zůstávají v celé době životnosti stabilní (např. stanice, potrubí, vedení, transformovny, budovy atd.).

Vybavení představuje individuální fyzické zařízení, které je z hlediska údržby samostatně udržováno a má vlastní kmenový záznam. Vybavení mohou být v průběhu životnosti instalována na různých technických místech. Kromě toho je možné, že po určitou dobu jsou nainstalována na fiktivních technických místech („skladech“), když se nacházejí ve skladu nebo v opravě.

4.3. Diagram vybraného procesu

Diagram vybraného procesu je na Obrázek 24, který byl vytvořen v programu ARIS Express. Seznam činností procesu:

A. Zjištění závady.

B. Otevření systému SAP.

C. Přihlášení do systému.

D. Založení hlášení o závadě.

E. Uložení hlášení.

F. Vytvoření zakázky z hlášení.

G. Zamítnutí opravy za provozu.

H. Schválení opravy za provozu.

I. Odstavení zařízení.

J. Změna statusu zakázky na „V realizaci“.

K. Oprava závady.

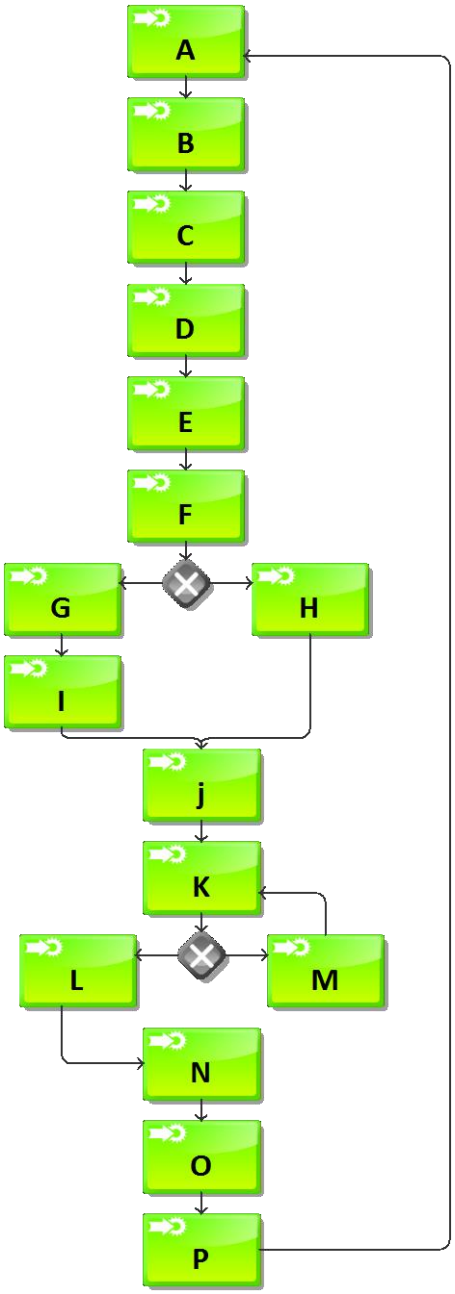
L. Schválení opravy.

M. Zamítnutí opravy.

N. Změna statusu zakázky.

O. Ukončení zakázky.

P. Uložení a odeslání do modulu CO.



Obrázek 24: Diagram vybraného procesu – oprava závady

Zdroj: [vlastní zpracování]

Z diagramu procesu lze vypočítat metriku MCC a CFC následovně:

- **McCabeovo cyklomatické číslo**

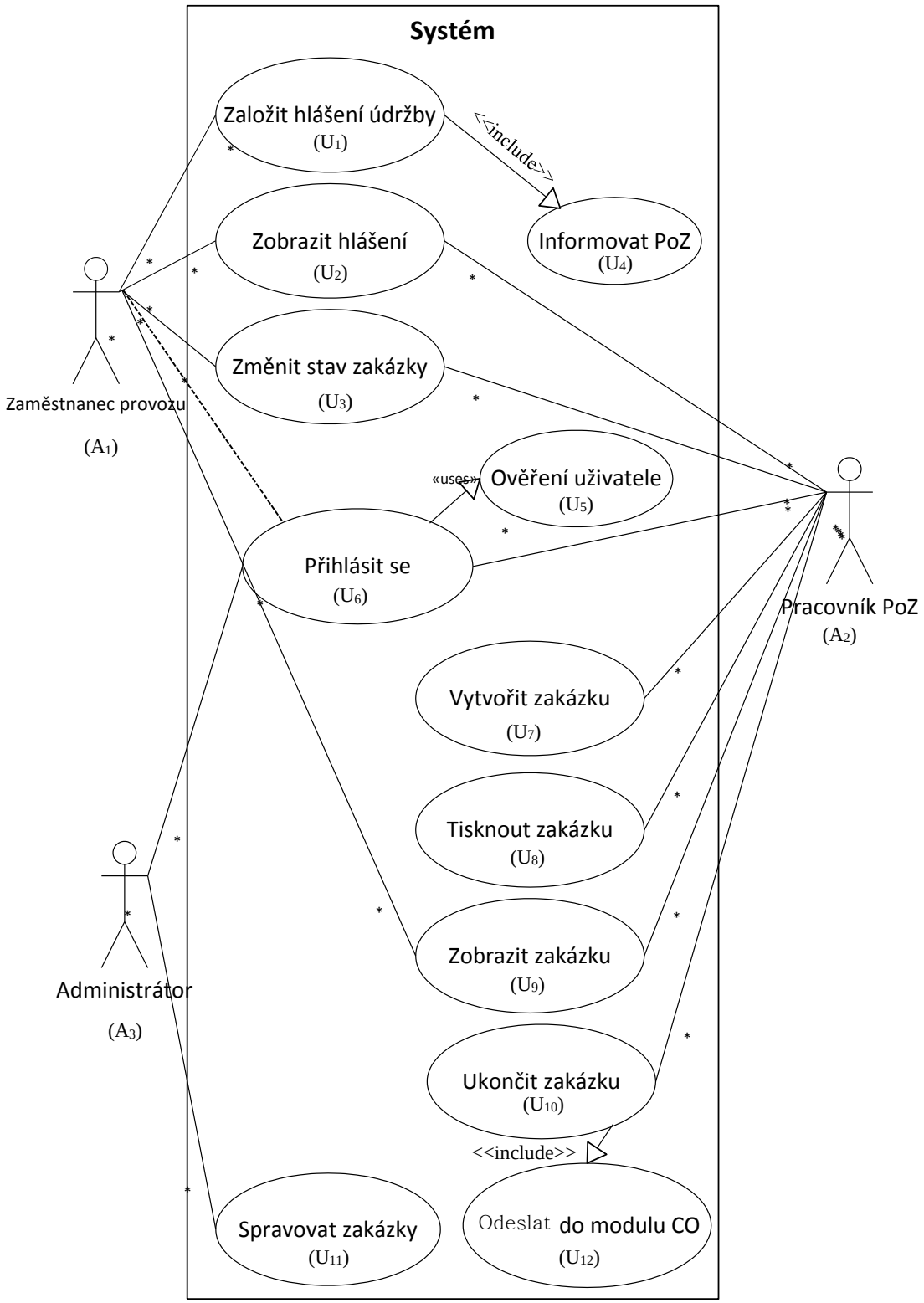
Proces obsahuje 16 činností, mezi kterými je 18 přechodů. Diagram obsahuje také dva uzly typu XOR. Pro výpočet cyklomatického čísla ***v(G)*** bude použita rovnice (7), kdy po dosazení: $v(G) = e - n + 2p = 18 - 16 + 2 = \mathbf{4}$**,** což splňuje kritérium ***v(G)*** **<*10*** označuje jednoduchý proces bez velkého rizika.

- **CFC metrika**

Dle postupu uvedeného v kapitole 2.5.1 vychází metrika z workflow grafu procesu. V grafu tohoto procesu jsou obsaženy dva uzly typu XOR (uzel F a K). Celková CFC je tedy rovna součtu CFC těchto uzlů. CFC_{XOR} (F)=2 a CFC_{XOR} (K)=2. Celková hodnota CFC je rovna **4** a i tato hodnota vypovídají o tom, že zkoumaný proces není složitý, neboť obsahuje jen dva uzly typu XOR.

4.4. Use Case diagram

Na Obrázek 25 je USE CASE digram vycházející z výše uvedeného popisu procesu. Diagram ukazuje užití systému při procesu opravy závady.



Obrázek 25: Use Case diagram – oprava závady

Zdroj: vlastní zpracování

4.4.1. Kvantifikace složitosti diagramu užití

Pro kvantifikaci složitosti budou použity metriky uvedená v kapitole 2.5.1. Výpočet těchto metrik je následující:

- **Merchesiho metrika**

Pro využití metriky 4UC je nejprve třeba určit jednotlivé veličiny vyskytující se v rovnici:

$$UC4 = K_1 UC1^2 + UC3 + K_2[smm([C]) - smm([E])|$$

- **K1** a **K2** jsou konstanty a pro zjednodušení budou obě rovny 0,5.
- **UC1** představuje počet případů použití, zde tedy 12.
- **UC3** je celkové množství vazeb mezi případy užití a aktéry bez redundantních vztahů zahrnutí (include), použití (uses) a rozšíření (extends), tedy 14.
- **[C]** je matice (s rozměrem N_a x N_{CU}) a prvek c_{ik} matice [C] má hodnotu 1, pokud aktér k má vztah s případem užití k, v jiném případě má hodnotu 0,

	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**	**U11**	**U12**
A_1	1	1	1	1	1	1	0	0	1	0	0	0
A_2	0	1	1	0	1	1	1	1	1	1	0	1
A_3	0	0	0	0	1	1	0	0	0	0	1	0

Po sečtení všech „1“ matice získáme: **smm([C]) =19**

- **[E]** je matice (N_a x N_{CU}) reprezentující vztahy mezi případy užití po odstranění redundantních vztahů zahrnutí (include), použití (uses) a rozšíření (extends).

	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**	**U9**	**U10**	**U11**	**U12**
A_1	1	1	1	0	0	1	0	0	1	0	0	0
A_2	0	1	1	0	0	1	1	1	1	1	0	0
A_3	0	0	0	0	0	1	0	0	0	0	1	0

Po sečtení všech „1“ matice získáme: **smm([E]) =14**

Po dosazení do rovnice (1) získáme složitost diagram:

$$UC4 = K_1 UC1^2 + UC3 + K_2[smm([C]) - smm([E])]$$
$$= 0{,}5 * 12^2 + 14 + 0{,}5 * (19 - 14) = \mathbf{88,5}$$

Složitost dle Merchesiho metriky vychází 88,5. Koeficienty K1 a K2 byly zvoleny oba stejné, v rovnici přestavují váhy počtu užití a počtu vazeb v diagramu.

- **Součet prvků diagramu**

Prostým součtem všech prvků v diagramu (užití, aktérů a vazeb) a dosazením do rovnice (2), je komplexita diagramu rovna:

$$C_{uc} = \sum aktérů + \sum užití + \sum vazeb = 3 + 12 + 17 = \mathbf{32}$$

Výsledek této metriky poskytuje informaci o velikosti diagramu, ale neodráží jeho strukturu a hustotu vazeb, nicméně pro srovnání digramů, řešících stejný problém, lze i z výsledku této metriky hodnotit jejich vliv na výslednou komplexitu IS.

4.5. Diagram tříd

Diagram tříd popisuje statickou strukturu systému. Popisuje objekty, jejich obsah a vzájemné vztahy mezi nimi. Diagram tříd pro popisovaný systém je znázorněn na Obrázek 26. Pro kvantifikaci složitosti budou využity metriky z kapitoly 2.5.2.:

- **Marchesiho metrika**

V diagramu je 6 tříd (OA_1) a 7 asociací (OA_2), z čehož plyne po dosazení do: $M = OA_1 + OA_2$ komplexita diagramu rovna **13.**

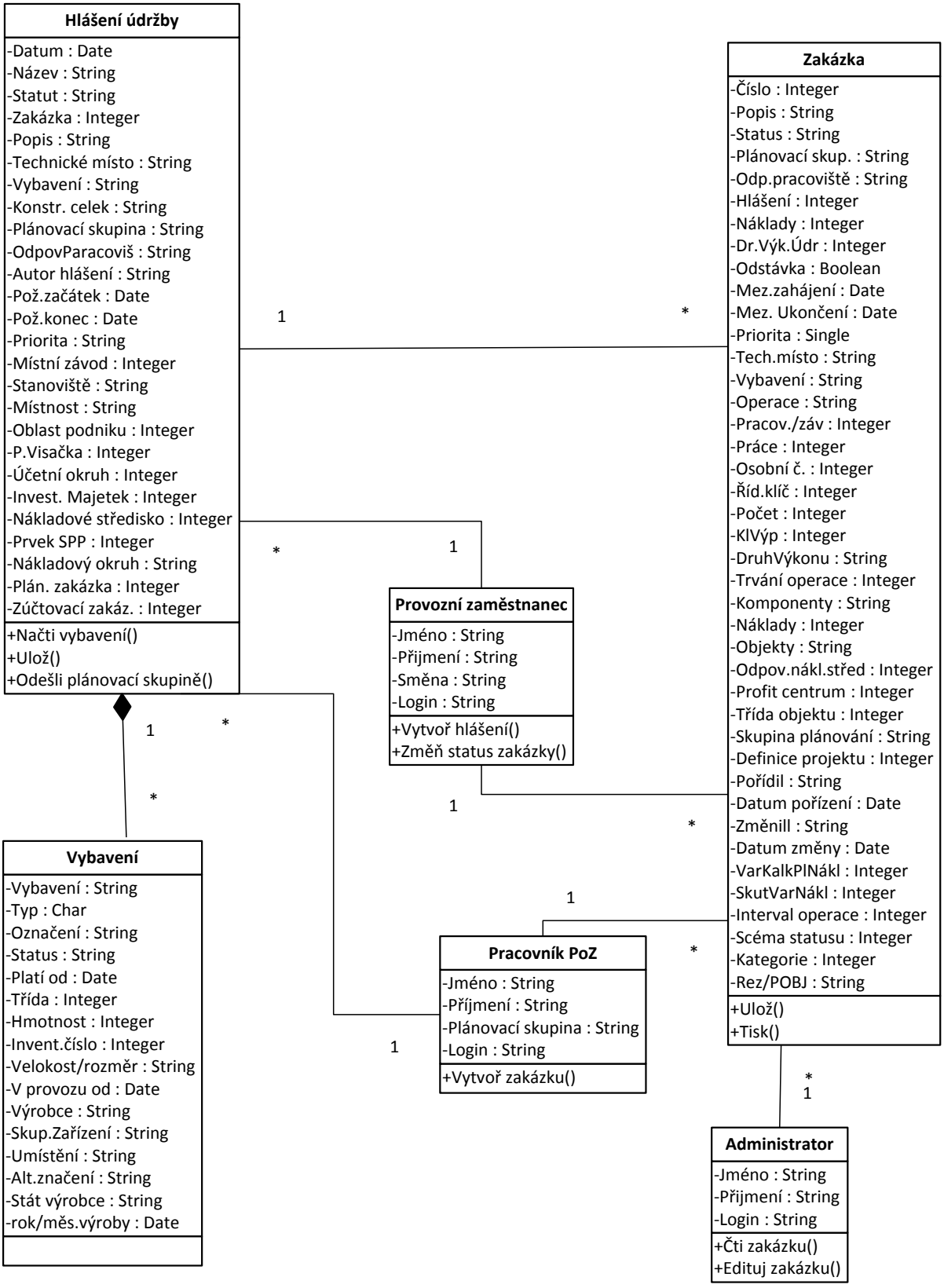
- **In metrika (dle P.Ina)**

Tato metrika zahrnuje také atributy a operace tříd. Ve zkoumaném diagramu je počet:

- tříd (TNC) = 6
- asociačních vztahů (TNA) = 7
- operací (TNO) = 10
- atributů (TNCA) = 94

Celkovou komplexitu digramu tříd získáme součtem:

$I = TNC + TNA + TNO + TNCA = 6 + 7 + 10 + 94 = \mathbf{117}$

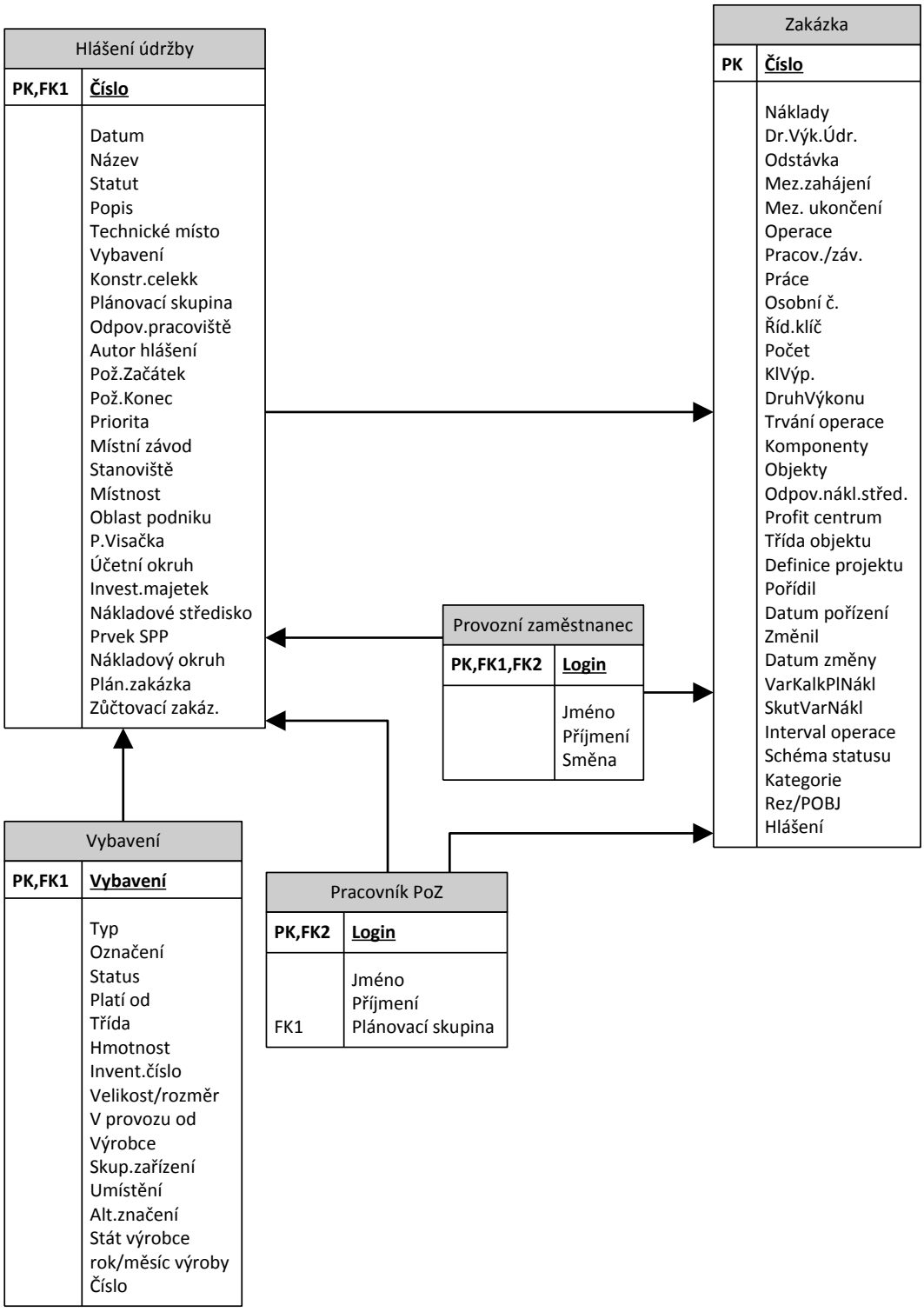


Obrázek 26: Diagram tříd – oprava závady

Zdroj: vlastní zpracování

4.6. Relační model dat

Relační model je model již pro zvolené řešení databázové struktury, konkrétně relační databáze. Pro řešený systém je model databáze na Obrázek 27. Pro kvantifikaci komplexity bude použita metoda DC, které je uvedena v kapitole 2.5.2



Obrázek 27: RMD – oprava závady

Zdroj: vlastní zpracování

Pro stanovení složitosti DC metodou je nejprve třeba pro každou relaci sečíst všechny příslušné prvky a výsledek představuje váhu relace **W**, která vychází z rovnice (3), kde **A** je počet atributů v daném vztahu, **K** je počet klíčů ve vztahu (součet primárních a sekundárních klíčů), **I** je počet indexů ve vztahu a **F** je počet cizích klíčů ve vztahu.

V modelu je 6 vztahů. Kvantifikace jednotlivých vztahů je popsána v tabulce níže

Relace	**A**	**K**	**I**	**F**	**W**
Hlášení – Zakázka	56	2	0	1	59
Hlášení – Provoz. zaměstnanec	28	2	0	3	33
Hlášení – Vybavení	41	2	0	2	45
Hlášení – Pracovník PoZ	28	3	0	3	34
Provoz. zaměstnanec – Zakázka	34	2	0	2	38
Pracovník PoZ – Zakázka	34	2	0	2	38

Výsledná složitosti celé databáze je dle rovnice (4):

$$C = \sum_{i=1}^{n} W_i = 59 + 33 + 45 + 34 + 38 + 38 = \mathbf{248}.$$

Prostým součtem prvků, tedy tabulek, atributů a relací získáme **94**. Metoda DC více odráží strukturu databázového modelu, a proto je komplexita více než dvojnásobná oproti prostému součtu, vypovídá tedy více o reálné komplexitě databáze.

4.7. Implementační úroveň

Při implementaci je tvořen vlastní zdrojový kód. Pro určení komplexity bude použita MCC metoda a vybrané Halsteadovi metriky uvedené v kapitole 2.5.5. Vzhledem k tomu, že zdrojový kód systému SAP není dostupný, bude využita část kódu menší aplikace požívané ve stejném podniku. Aplikace je programována v programovacím jazyce Python. Cílem je tedy ukázka vlastního postupu měření komplexity různými metrikami a vzájemné porovnání výsledků. Zkoumaný kód je na Obrázek 28.

```
def sendPVMail(ini):
  logging.info("Starting...")

  excel = startExcel()
  # open original workbook
  fnameIn = ini.get("excel", "input")
  wbkIn = openWorkbook(excel, fnameIn)
  # run macros
  for macroName in ini.getarray("excel", "macro"):
    runExcelMacro(wbkIn, macroName)
  # create new workbook
  wbkOut = newWorkbook(excel)
  sheetOutName = wbkOut.Sheets(1).Name
  # copy sheets
  for sheetName in ini.getarray("excel", "sheet"):
    copyExcelSheetBefore(wbkIn, wbkOut, sheetName, sheetOutName)
  # save and close new workbook
  fnameOut = makeFileName(fnameIn, ini.get("excel", "output"))
  saveWorkbook(wbkOut, fnameOut, ini.getint("excel", "outputformat"))
  closeWorkbook(wbkOut)
  # close original workbook
  closeWorkbook(wbkIn)
  # quit excel
  quitExcel(excel)

  # prepare attachment
  attachment = createMimeAttachmentFromFileName(fnameOut)
  # prepare message
  msg = mailMessage(
    ini.get("mail", "sender"),
    ini.getarray("mail", "recipient"),
    makeFileName(fnameIn, ini.get("mail", "subject")),
    "\r\n\r\n".join(ini.getarray("mail", "message"))
  )
  # attach attachment
  msg.attach(attachment)
  # serialize message
  msgstr = serializeMessage(msg)

  # get retry count
  r_count = 1
  r_time = 300
  if ini.has_section("retry"):
    r_count = int(ini.get("retry", "count"))
    r_time = int(ini.get("retry", "time"))

  while r_count > 0:
    r_count -= 1

    try:
      # connect to mail server
      conn = connectMail(**ini.getdict("send", "", optionseparator=""))
      # send message
      conn.sendmail(ini.get("mail", "sender"), ini.getarray("mail", "recipient"), msgstr)
      # disconnect
      conn.close()
      break
    except Exception, e:
      logging.exception("Error during send.")

    time.sleep(r_time)

  logging.info("Finished.")
```

Obrázek 28: Ukázka zdrojového kódu

Zdroj: aplikace IND

Výše uvedený kód slouží k automatickému vytvoření a odeslání mailu s přílohou, která obsahuje vybraná data z provozu elektrárny. Tuto aplikaci vytvořila společnost IND s.r.o., která pro podnik spravuje databázi technologických dat. Kód této procedury je na Obrázek 28 výše. Výše uvedený kód lze popsat control-flow grafem vedeným na Obrázek 29 níže.

Seznam uzlů diagramu:

a) SendPVMail Start.

b) Start aplikace excel.

c) Otevření excelového sešitu s makrem, i=1.

d) Cyklus nalezení makra (i).

e) Spuštění makra i++.

f) Vytvoření nového sešitu b, i=0.

g) Cyklus kopírování záložek sešitů.

h) Kopírování záložek ze sešitu a do b.

i) Uložení a zavření sešitů a, b, ukončení aplikace excel.

j) Nastavení emailu s přílohou (sešit b), i=ini.r_count.

k) Rozhodnutí o opakování odesílání mailu, i>0.

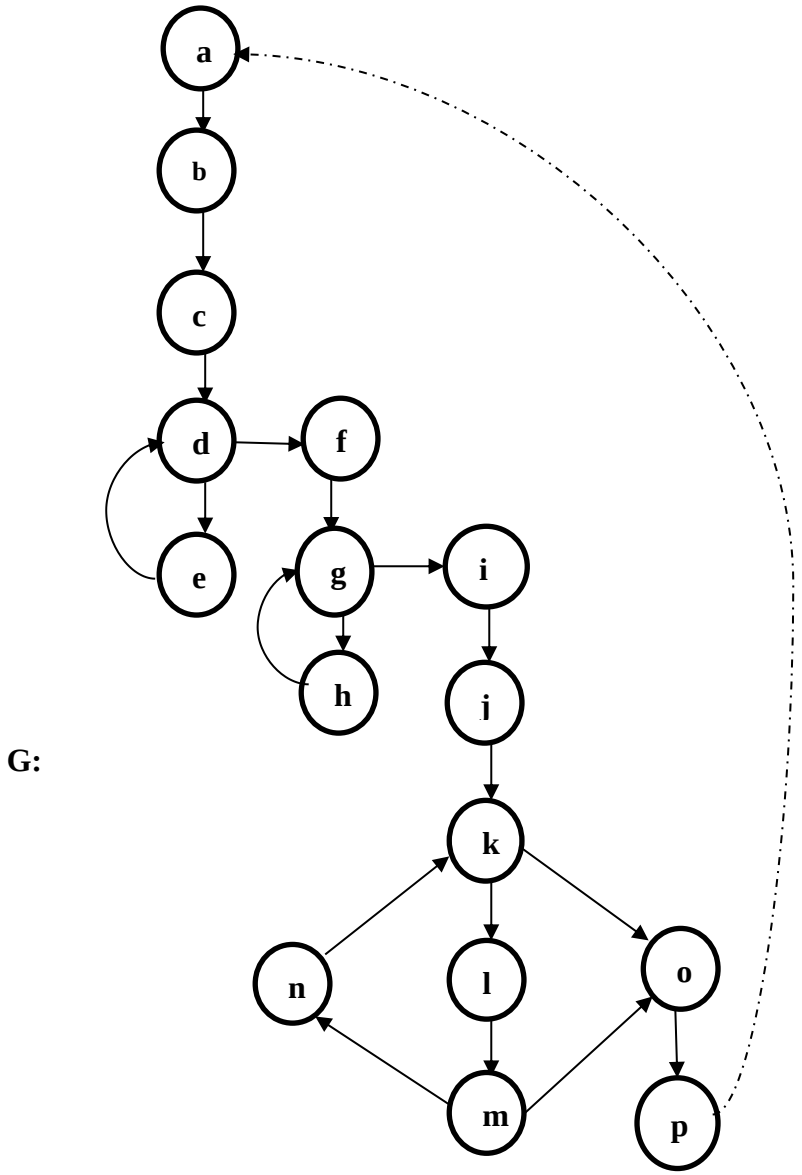
l) Odeslat email, i--.

m) Odesílání úspěšné.

n) Čekat (300 vteřin).

o) Informace o stavu

p) Ukončení procedury.



Obrázek 29: Control-flow graf vybrané procedury

Zdroj: vlastní zpracování

Ze zdrojového kódu a control-flow grafu lze učit komplexitu dle následujících metrik:

- **Cyklomatická komplexita**

Pro výpočet cyklomatické komplexity bude použita rovnice (7). V control-flow grafu na Obrázek 29 je 16 vrcholů a 20 spojnic. Cyklomatické číslo je tedy:

$$v(G) = e - n + 2p = 20 - 16 + 2 = \mathbf{6}$$

Výsledná hodnota cyklomatického čísla je **6,** což dle kritéria ***v(G)*≤10** ukazuje, že se jedná o jednoduchý program bez velkého rizika chyb.

- **Halsteadovi metriky**

Pro aplikaci jednotlivých metrik je třeba nejprve určit a sečíst všechny operátory a operandy a tak, vypočítat veličiny **n_1 (počet různých operátorů)**, **n_2 (počet různých operandů)**, **N_1 (celkový počet výskytů operátorů)** a **N_2 (celkový počet výskytů operandů)**. Operátory a operandy byly určeny dle následujících pravidel:

1. Komentáře nejsou počítány.
2. Identifikátory a deklarace funkcí nejsou počítány.
3. Všechny proměnné a konstanty jsou považovány za operandy.
4. Volání funkcí se počítá jako operátor.
5. Všechny cykly se považují za operátory.
6. Vyhrazená slova jako return, default, continue, break, atd. jsou považovány za operátory.
7. Všechny závorky, čárky a terminátory jsou považovány za operátory.
8. Matematické operátory jako "+" a "++" se počítají odděleně

Dle výše uvedených pravidel byl analyzován kód z Obrázek 28, soupis operátorů a operandů včetně počtu jejich výskytů je uveden v Tabulka 6 níže.

Tabulka 6: Soupis operátorů a operandů kódu

Operátor	Počet výskytů	Operand	Počet výskytů
logging	3	info	1
startExcel	1	excel	8
()	40	fnameIn	2
„“	33	WbkIn	3
.	17	macroName	2
=	13	WbkOut	5
-=	1	sheetoutName	1
>	1	sheetName	2
for	2	ini	13
get	7	fnameOut	2
getarray	5	attachment	2
openWorbook	1	msg	2
runExcelMacro	1	msgstr	2
newWorkbook	1	r_count	2
copyExcelSheetBefore	1	r_time	2
makeFileName	1	conn	3
saveWorkbook	1	e	1
closeWorkbook	1	1	3
quitExcel	1	0	1
createMimeAttachmentFromFileName	1	300	1
join	1	exception	2
serializeMessage	1	time	1
has_section	1	**n_2=22**	**N_2=61**
while	1		
try	1		
connectMail	1		
sendmail	1		
close	1		
break	1		
except	1		
sleep	1		
getdict	1		
n_1 = 32	**N_1=144**		

Zdroj: vlastní zpracování

Z počtu operátorů a operandů budou určeny následující metriky: délka programu N, slovní zásoba n, objem programu V, programová obtížnost D a inteligentní obsah I. Pro výpočet budou využity rovnice z kapitoly 2.5.5. Výsledky shrnuje Tabulka 7.

Tabulka 7: Výsledky vybraných Halsteadových metrik

Metrika	Hodnota metriky
délka programu	205
slovní zásoba	54
objem programu	1179,75
programová obtížnost	44,36
inteligentní obsah	26,59

Zdroj: vlastní zpracování

V další části bude hodnocena provozní fáze vybrané části IS, a to z pohledu uživatelského rozhraní, konkrétně dle počtu používaných a zobrazovaných položek formulářů, které se používaní při procesu opravy závady, který je popsán v kapitole 4.2.

4.8. Provoz

Pro zhodnocení komplexity ve fázi provozu bude analyzováno uživatelské rozhraní z pohledu nadbytečné komplexity. Pro proces tvorby zakázky na údržbu zařízení elektrárny jsou využity uživateli 3 formuláře: Vybavení, Hlášení údržby, Zakázka. Pro zhodnocení komplexity byly sumarizovány veškeré položky těchto formulářů. Dále byly jednotlivé položky rozděleny povinné, automaticky vyplňované a nepoužívané. Povinná pole jsou taková, která jsou nutná pro úspěšný průběh procesu. Automatická pole, jsou taková, která systém vyplní po zadání daného povinného pole. Poli nepoužívanými se rozumí taková, která zůstávají nevyplněná a nejsou pro proces nezbytná. Duplicitní pole jsou taková, která obsahují informaci, která je již uvedena v jiném poli stejného formuláře například na jiné záložce. Výsledek tohoto rozdělení je v Tabulka 8 níže.

Tabulka 8: Počty položek uživatelského rozhraní

	Vybavení	Hlášení	Zakázka
celkem	34	35	80
povinných	10	5	26
automatických	5	23	32
nepoužívaných	16	6	15
duplicitní	3	1	7

Zdroj: vlastní zpracování

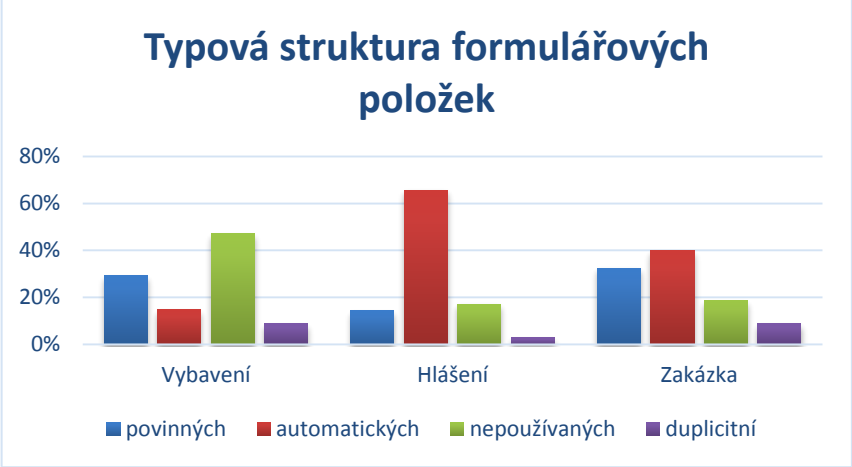
Procentuální vyjádření počtu jednotlivých typů položek je v Tabulka 9.

Tabulka 9: Procentuální zastoupení jednotlivých typů položek

	Vybavení	Hlášení	Zakázka
povinných	29 %	14 %	33 %
automatických	15 %	66 %	40 %
nepoužívaných	47 %	17 %	19 %
duplicitní	9 %	3 %	9 %

Zdroj: vlastní zpracování

Názorné porovnání jednotlivých typů položek je na grafu na Obrázek 30. Vzhledem k celkovému počtu položek je největší počet nepoužívaných polí ve formuláři Vybavení a nejvíce duplicitních položek obsahuje formulář Zakázka. Položky nepoužívané a duplicitní lze považovat za nadbytečné a položky povinné a automatické za nezbytné. Na základě tohoto tvrzení je vytvořena sumarizovaná Tabulka 10.



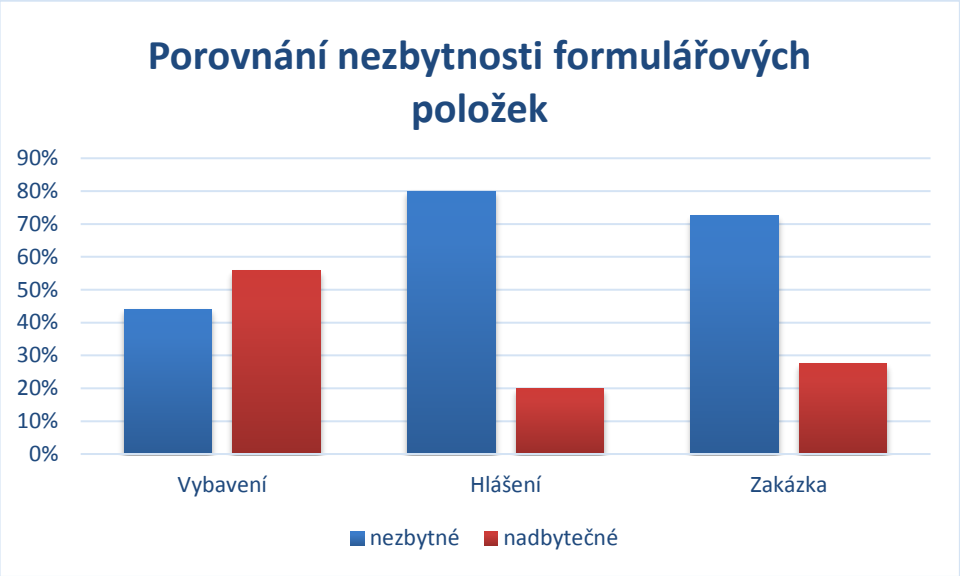
Obrázek 30: Položky formulářů

Zdroj: vlastní zpracování

Tabulka 10: Sumarizované položky

	Vybavení	Hlášení	Zakázka	Průměr
nezbytné	44 %	80 %	73 %	66 %
nadbytečné	56 %	20 %	28 %	34 %

Zdroj: vlastní zpracování



Obrázek 31: Porovnání nezbytnosti formulářových položek

Zdroj: vlastní zpracování

Ze sumarizované Tabulka 10 plyne, že nejvíce nadbytečných položek obsahuje formulář „Vybavení“, a to 56 %, lze tedy říci, že komplexita tohoto formuláře je optimální pouze ze 44 %. Toto je způsobeno nedůslednou analýzou systému, kdy nebyly přesně specifikovány požadované funkcionality. Následkem tohoto nebyl formulář optimalizován a zbylo v něm 56 % nadbytečných polí, které neobsahují žádnou informaci. Formulář „Hlášení“ je z pohledu počtu položek optimální z 80 % a ze zkoumaných formulářů se nejvíce blíží optimální komplexitě. Formulář „Zakázka“ obsahuje 28 % nadbytečných polí, zde je velké zastoupení duplicitních položek, které je způsobeno tím, že formulář má 10 záložek, kdy na každé záložce jsou opakující se údaje. Tyto údaje by měly být v hlavičce formuláře, čímž by kles celkový počet položek i položek duplicitních. Veškerá nadbytečná formulářová pole zabírají místo v databázi a zhoršují přehlednost uživatelského rozhraní.

Z grafu na Obrázek 31 plyne, že v případě formuláře „Vybavení“ je počet nadbytečných polí vyšší než počet nezbytných. U tohoto formuláře je velké množství nepoužívaných polí. Důvod nepoužívání polí je, že uživatelé systému pro svoji práci nepotřebují uvádět tak detailní informace o vybavení, které jsou ve formuláři k dispozici.

Výše provedená analýza uživatelského rozhraní se vztahuje na hodnocení zavedeného hotového systému. Nadbytečná pole by se v systému nevyskytovala, kdyby byl systém vytvářen již od fáze analýzy s větším důrazem na optimální komplexity. Překážkou pro snížení této komplexity za provozu systému je, že systém SAP je spravován externí IT firmou, kdy veškeré úpravy znamenají náklady navíc. Existuje, zde také hrozba ztráty dat a porušení konzistence databáze.

Tabulka 11: Sumarizace použitých metrik

Konceptuální úroveň		**Technologická úroveň**		**Implementační úroveň**		**Provoz**	
McCabeova metrika	4	DC metrika	248	McCabeova metrika	6	Nezbytné položky	101
CFC	4	Součet prvků (RMD)	94	Délka programu	205	Nadbytečné položky	48
Merchesiho metrika (use case)	88,5	Merchesiho metrika (diagram tříd)	13	Slovní zásoba	54		
Součet prvků (use case)	32	In metrika	117	Objem programu	1179,7		
				Programová obtížnost	44,3		
				Inteligentní obsah	26,6		

Zdroj: vlastní zpracování

4.9. Sumarizace použitých metrik pro měření komplexity

V této kapitole byly použity vybrané metriky pro měření komplexity IS v jednotlivých úrovních abstrakce. Výsledky těchto metrik lze využit jako podklad pro rozhodování při výběru řešení vybraného systému. Metriky byly demonstrovány na části informačního systému, který je využíván v reálné společnosti a je tedy ve fázi provozu. Pro demonstraci použití metrik i v úrovni konceptuální a technologické bylo použito metody reverzního inženýrství, kdy z hotového systému byl odvozen jeho vývoj. Pro analýzu implementační úrovně byl použit zdrojový kód procedury, která je využívána v aplikaci ve stejném podniku. Provozní fáze byla analyzována z pohledu optimalizace uživatelského rozhraní, kde byla analyzovány formuláře IS, používané při vybraném procesu. Hodnoty vybraných metrik shrnuje Tabulka 11 výše**. Rozhodnutí, zda získané výsledky představují vysokou či nízkou komplexitu v daném systému však musí provézt expert v dané problematice, však dle výsledků MCC a CFC lze usuzovat, že se jedná o systém s nízkou komplexitou.**

4.10. Změna komplexity systému při modifikaci procesu

Níže bude provedena modifikace procesu, kdy bude z důvodu zvýšení bezpečnosti práce, proces rozšířen o prokazatelné zajištění zařízení v případě nutnosti. Zajištění provede pracovník provozu a zkontroluje koordinátor BOZP. S modulem PM bude tedy nový aktér v podobě koordinátora BOZP. Upravený proces znázorňuje diagram na Obrázek 32 a seznam činností je níže. V diagramu jsou červeně označeny činnosti, které jsou navíc oproti původnímu procesu.

Seznam činností procesu:

- **A.** Zjištění závady.
- **B.** Otevření systému SAP.
- **C.** Přihlášení do systému.
- **D.** Založení hlášení o závadě.
- **E.** Uložení hlášení.
- **F.** Vytvoření zakázky z hlášení.
- **G.** Zamítnutí opravy za provozu.
- **H.** Schválení opravy za provozu.
- **I.** Odstavení zařízení.
- **J.** Změna statusu zakázky na „V realizaci“.

K. Oprava závady.

L. Schválení opravy.

M. Zamítnutí opravy.

N. Změna statusu zakázky.

O. Ukončení zakázky.

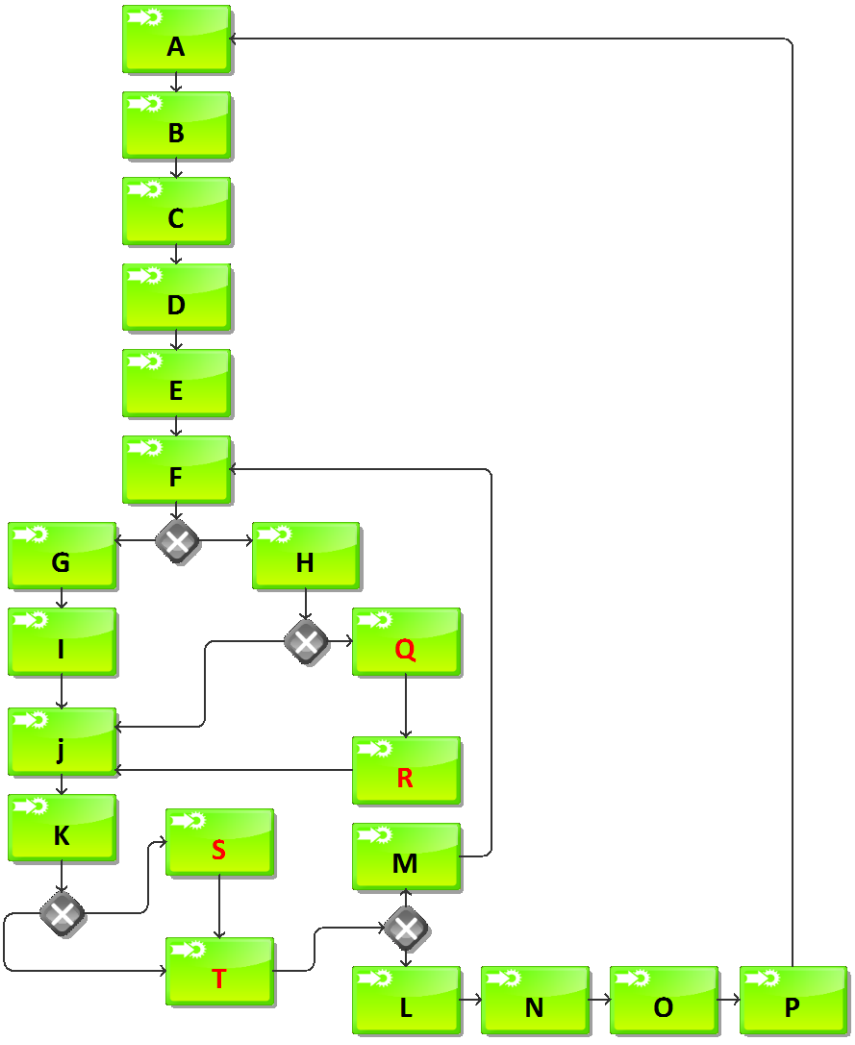
P. Uložení.

Q. Zajištění zařízení.

R. Potvrzení zajištění v systému.

S. Odjištění.

T. Odzkoušení zařízení



Obrázek 32: Diagram upraveného procesu

Zdroj: vlastní zpracování

V jednotlivých úrovních informačního systému se změna procesu projeví:

- zvýšením počtu aktérů a případů užití (nový aktér se bude do systému přihlašovat, bude prohlížet zakázky a potvrzovat zajištění),
- v diagramu tříd vznikne nová třída BOZP (bude měnit nový atribut třídy zakázka),
- v databázovém modelu vznikne další tabulka s atributy pověřených pracovníků BOZP s relaci se zakázkou (relace bude mít integritní omezení takové, že zakázka může být potvrzena jedním pracovníkem BOZP a jeden pracovník může potvrdit více zakázek,
- ve formuláři zakázky přibydou dvě pole (zajištění (ANO/NE) a jméno pracovníka).

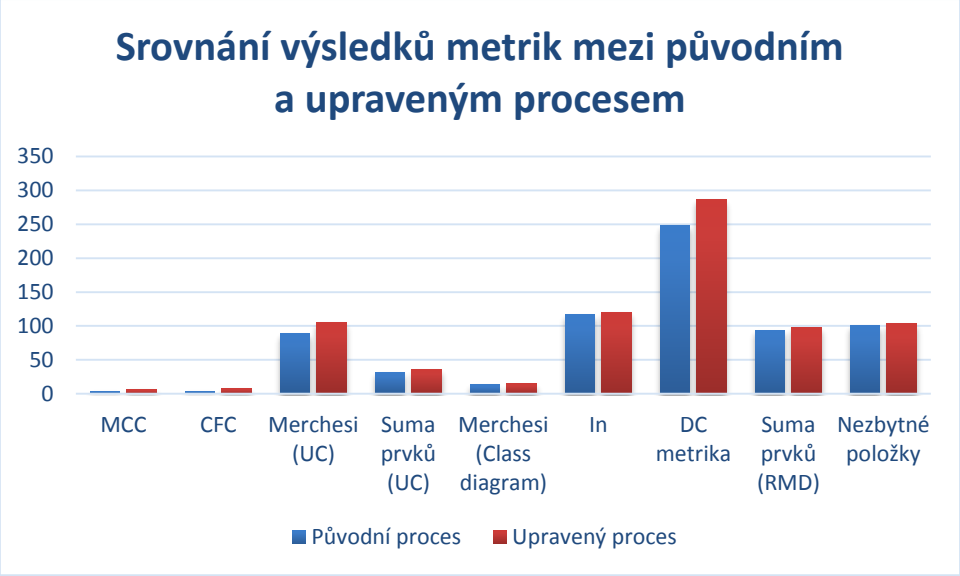
To, jak se zvýší komplexita kódu, není možno posoudit, takže v tomto případě nebude změna komplexity implementační úrovně zkoumána. To, jak se projeví tyto změny v ostatních úrovních, ukazuje Tabulka 12.

Tabulka 12: Sumarizace metrik upraveného procesu

Konceptuální úroveň		**Technologická úroveň**		**Provoz**	
McCabeova metrika	6	DC metrika	287	Nezbytné položky	103
CFC	8	Součet prvků (RMD)	98	Nadbytečné položky	48
Merchesiho metrika (use case)	105	Merchesiho metrika (diagram tříd)	15		
Součet prvků (use case)	36	In metrika	120		

Zdroj: vlastní zpracování

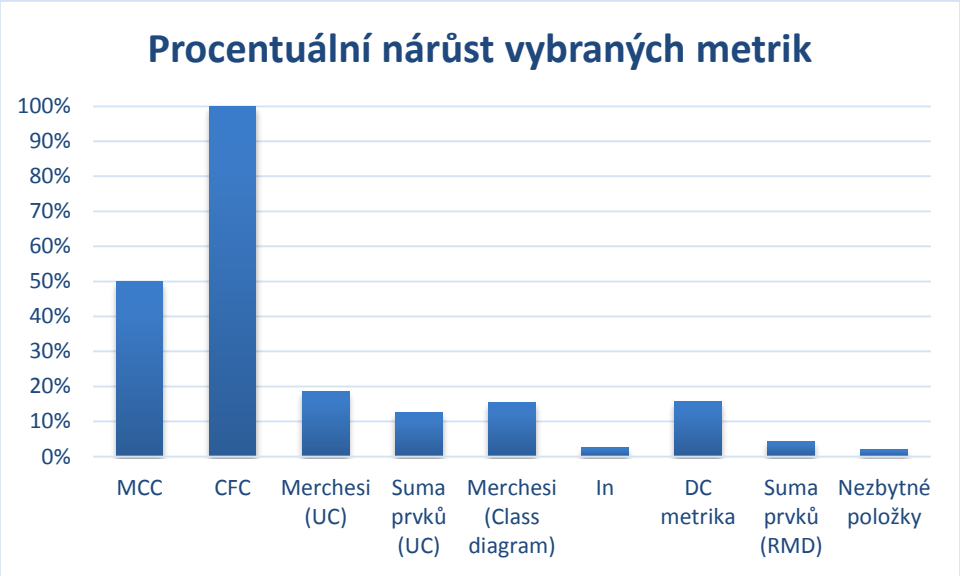
Porovnání nárůstu jednotlivých metrik ilustruje graf na Obrázek 33. Z grafu je patrné, že došlo k nárůstu všech použitých metrik.



Obrázek 33: Porovnání metrik původního a upraveného procesu

Zdroj: vlastní zpracování

Procentuálně se nejvíce oproti původní hodnotě zvýšila hodnota metriky CFC, která se zdvojnásobila. Metrika MCC se zvýšila o 50 % původní hodnoty. Merchesiho metriky, součet prvků diagramu use case a DC metrika se zvýšily o cca 15–20 %. Ostatní metriky nezvýšily svoje hodnoty o více než 4 %. Procentuální změny metrik popisuje graf na Obrázek 34.



Obrázek 34: Procentuální nárůst metrik po úpravě procesu

Zdroj: vlastní zpracování

Odezva vybraných metrik na modifikaci procesu odráží robustnost, modifikovatelnost a další kvalitativní vlastnosti systému. V tomto případě došlo ke zdvojnásobení komplexity procesu dle metriky CFC a nárůstu o 50% dle MCC. To, že metriky týkající se vlastního IS vzrostly maximálně o cca 20%, naznačuje, že informační systém je flexibilní do takové míry, že úprava tohoto procesu u něj nevyvolá extrémní nárůst komplexity. Takto lze na systému testovat různé scénáře (simulace) v rámci fází životního cyklu a následně posuzovat dopad provedených změn.

Nárůst komplexity systému vlivem modifikace procesu, bude mít v praxi reálný finanční dopad, protože je třeba zavézt úpravy do informačního systému. Počáteční a cíloví stav procesu zůstal nezměněn, ale došlo ke zvýšení bezpečnosti procesu. K takovémuto upravení procesu často dochází například při vzniku pracovního úrazu.

Při hodnocení obou variant, tedy původního a modifikovaného procesu je třeba rozhodnout, zda je zvýšení bezpečnosti vyváženo zvýšením nákladů. Obě kritéria lze přiřadit k jednotlivým variantám a společně s výsledky metrik využít ve vybrané metodě vícekriteriálního rozhodování. Použití metrik jako kritérií jednotlivých variant je také možno využít pro srovnání řešení od jednotlivých dodavatelů softwaru. Komplexita dle metrik nemusí vždy korelovat s cenou, kterou dodavatel požaduje za nabízený systém nebo jen za provedení úprav na systému požívaném. Také je třeba zvážit možné zvýšené náklady na údržbu, kterou s sebou nese zvýšená komplexita.

4.11. Shrnutí

Tato kapitola se věnovala kvantifikaci komplexity použitím vybraných metrik. Metriky lze rozdělit na ty, které odrážejí složitost struktury a ty, které jsou zaměřený spíše na velikost systému. Za přínosnější metriky lze jednoznačně považovat ty jenž jejich velikost je závislá na počtu a hustotě vazeb mezi prvky. V této práci těmito metrikami jsou McCabeova metrika, Cardosova CFC, Merchesiho metrika pro use case diagram, DC metrika pro databázový model. Halsteadovi metriky jsou přínosné pro porovnání jednotlivých řešení stejného problému různými způsoby, kdy program dává stejný výsledek, ale dle metrik lze jednoznačně určit kód s nižší komplexitou. Z tohoto důvodu jsou také velmi často implementovány do softwarů určených pro automatickou analýzu zdrojového kódu jako je Testwell CMT nebo Radon.

Metriky založené na součtu prvků neodráží přímo složitost, ale pouze velikost nebo členitost. Výsledky těchto metrik však mohou být také přínosné pro potvrzení rozhodnutí o volbě lepšího řešení v případě, že metriky popisující strukturu nabývají velmi podobných hodnot.

Výsledky vybraných metrik lze využít například pro podporu rozhodování při volbě optimální varianty řešení. Hodnoty (většinou bezrozměrné číslo) by bylo možné také využít pro porovnání variant a následné použití při vícekriteriálním rozhodování. Kdy hodnoty metrik v jednotlivých úrovních by představovaly kritéria jednotlivých variant řešení. Pro vytvoření pořadí variant by bylo možné následně využít metodu TOPSIS.[4] Výstup z této metody by mohl posloužit i osobě, která není expertem v oblasti komplexity informačních systémů.

[4] Technique for Order of Preference by Similarity to Ideal Solution – metoda vícekriteriálního rozhodování založená na měření vzdálenosti od optimální a bazální varianty.

ZÁVĚR

Práce byla věnována komplexitě informačních systému. Byly vymezeny jednotlivé typy informačních systému. Hlavním tématem byl komplexita. Nejprve byla komplexita definována v kontextu informační teorie. Byly vysvětleny 3 přístupy ke složitosti, a to Shannonova entropie, Kolmogorova složitost a Benettova logická hloubka. Tyto tři druhy složitosti poskytly teoretický základ pro další část, kde byla komplexita zkoumána v rámci fází životního cyklu a obsahových dimenzí infomačního systému. Pro tento účel bylo použito členění fází a dimenzí dle metodiky MMDIS. Tato metodika, která je vyvinuta na Fakultě informatiky a statistiky, Vysoké školy ekonomické v Praze, byla využita z důvodu logického členění fází a obsahových dimenzí, což zvyšuje komunikovatelnost a pochopitelnost jejich vztahů ke komplexitě systému.

Vztah jednotlivých fází a dimenzí ke komplexitě poskytuje náhled na příčiny a důsledky zvyšující se složitosti informačního systému. Tyto jsou shrnuty v sumarizačním Obrázek 8 a Obrázek 21. Vztah komplexity k dané fázi či dimenzi poskytuje informaci o tom, jaké jsou její aspekty složitosti, což lze využít při vývoji infomačního systému a poskytuje jistou nástavbu metodiky MMDIS.

Dopady vysoké komplexity a výhody optimální složitosti informačního systému pro organizace jsou sumarizovány v kapitole 3. V této kapitole byly určeny výhody nízké komplexity, slabiny vysoké komplexity, příležitosti pro podnik plynoucí z optimální komplexity a hrozby plynoucí z vysoké komplexity dle dané úrovně abstrakce. Dle těchto dopadů lze jednoznačně tvrdit, že je třeba nejpečlivěji se komplexitě věnovat v konceptuální úrovni, neboť její následky jsou pro výsledek projektu informačního systému nejzávažnější. Komplexita této úrovně se přenáší do dalších fází a významně ovlivňuje výsledný systém i celkové náklady, které bude třeba vynaložit. Komplexita informačního systému ve fázi provozu roste, a to z důvodu mnoha příčin, tyto byly uvedeny v podkapitole 3.4.

Poslední kapitola byla věnována aplikaci vybraných metrik na reálné části infomačního systému ve třech úrovních, konceptuální, technologické a implementační. Výsledky těchto metrik shrnuje Tabulka 11: Sumarizace použitých metrik.Hodnoty z použitých metrik, lze však posoudit objektivně pouze expertem v daném oboru, ale lze je použít jako podporu při rozhodování o výběru softwarového řešení v dané aplikaci nebo jako podklad pro využití metod vícekriteriálního rozhodování.

Byla také hodnocena provozní fáze systému, kde bylo hodnoceno uživatelské rozhraní. V tomto byla rozlišena komplexita nezbytná a nadbytečná pomocí rozdělení položek formulářů na nezbytné a nadbytečné.

Pro demonstraci toho, jak změna požadavků na systém způsobená modifikací procesu, působí na složitost informačního systému a ovlivňuje jednotlivé vybrané metriky jejího měření, byl upraven původní proces. Zvýšení metrik komplexity v jednotlivých sledovaných úrovních je v Tabulka 12. Z uvedených výsledků je patné, že zvýšení bezpečnosti práce při provádění oprav, zvýší komplexitu informačního systému. Obdobně lze systém testovat na různé modifikace a sledovat odezvu jednotlivých metrik. Z reakce metrik je pak možno usuzovat o kvalitativních vlastnostech informačního systému.

Lze sumarizovat, že příliš vysoká komplexita má za následek růst nákladů a neefektivní práci, proto je třeba komplexitu sledovat v rámci celého životního cyklu a nenavyšovat ji, ale zároveň je třeba vytvořit systém dostatečně složitý, aby obsáhl všechny aspekty daného procesu, který má být v nejlepším případě automatizován a dovolil přiměřenou flexibilitu pro provádění případných úprav při změně požadavků.

POUŽITÁ LITERATURA

[1] ALAMOUDI, Doaa a Amlendu KUMAR. Information System Complexity and Business Value. *International Journal of Economics & Management Sciences* [online]. 2017, **06**(02) [cit. 2018-11-19]. DOI: 10.4172/2162-6359.1000400. ISSN 21626359.

[2] BASL, Josef. Podnikové informační systémy: podnik v informační společnosti. 2., výrazně přeprac. a rozš. vyd. Praha: Grada, 2008. Management v informační společnosti. ISBN 8024722798.

[3] BOEHM, Barry. Software and its impact: A quantitative assessment. *Datamation*. 1972, 1972(19), 48-59.

[4] BRUCKNER, Tomáš, Jiří VOŘÍŠEK, Alena BUCHALCEVOVÁ, Iva STANOVSKÁ, Dušan CHLÁPEK a Václav ŘEPA. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012, s. 144-200. Management v informační společnosti. ISBN 978-80-247-4153-6.

[5] CARDOSO, J. 2008. Business Process Control-Flow Complexity: Metric, Evaluation, and Validation. International Journal of Web Services Research (IJWSR), 5, 49-76.

[6] COPPICK, J. Chris a Thomas J. CHEATHAM. Software metrics for object-oriented systems. In: *Proceedings of the 1992 ACM annual conference on Communications - CSC '92* [online]. New York, New York, USA: ACM Press, 1992, 1992, s. 317-322 [cit. 2018-11-15]. DOI: 10.1145/131214.131254. ISBN 0897914724.

[7] Druhy informačních systémů. *Prostředky informačních technologií* [online]. Masarykova univerzita, 2002 [cit. 2018-03-06]. Dostupné z: http://pit.wz.cz/informacni-systemy.php

[8] *Globální architektura IS/IT* [online]. Mendelova univerzita v Brně [cit. 2018-03-06]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=4756

[9] HOLUB, I. Jak a proč vzniká komplexita v IS. Systémová integrace. Praha: Česká společnost pro systémovou integraci, 2012, 1, s. 74-93. ISSN: 1804-2716

[10] HOLUB IT CONSULTING. Komplexní systémy. holub.cz [online]. [cit. 2017-03-03]. Dostupné z: http://www.holub.cz/index.php/sk/komplexni-systemy.

[11] IN, P., Kim, S., Barry, M. UML-based object-oriented metrics for architecture complexity analysis. Department of computer science, Texas A&M University, 2003. http://faculty.cs.tamu.edu/hohin

[12] Informační systémy: Základní pojmy a souvislosti. *Dokumentační portál Centra informačních technologií VŠB-TUO* [online]. Ostrava: VŠB-TUO Ostrava, 2016 [cit. 2018-03-07]. Dostupné z: http://homen.vsb.cz/~s1i95/isvdas/is/is_uvod.htm

[13] ISAIAS, Pedro a Tomayess ISSA. Information System Development Life Cycle Models. *High Level Models and Methodologies for Information Systems* [online]. New York, NY: Springer New York, 2015, 2015-9-25, , 21-40 [cit. 2018-12-03]. DOI: 10.1007/978-1-4614-9254-2_2. ISBN 978-1-4614-9253-5. Dostupné z: http://link.springer.com/10.1007/978-1-4614-9254-2_2

[14] JANÍČEK, Přemysl a Jiří MAREK. *Expertní inženýrství v systémovém pojetí*. Praha: Grada, 2013, s. 60-62. Expert (Grada). ISBN 978-80-247-4127-7.

[15] JOHNSON, Jeff. *Designing with the mind in mind: simple guide to understanding user interface design rules*. 1. Boston: Morgan Kaufmann Publishers/Elsevier, c2010, s. 137. Expert (Grada). ISBN 978-0-12-375030-3.

[16] LEMBERGER, Pirmin a Médéric MOREL. *Managing complexity of information systems: the value of simplicity*. Hoboken, NJ: Wiley, 2011, s. 17-145. ISBN 978-1-84821-341-8.

[17] LLOYD, Seth. Measures of complexity: a nonexhaustive list. *IEEE Control Systems* [online]. 2001, **21**(4), 7-8 [cit. 2018-12-03]. DOI: 10.1109/MCS.2001.939938. ISSN 1066-033X. Dostupné z: https://ieeexplore.ieee.org/document/939938/

[18] MALEČKOVÁ, D. 2013. Informační věda a teorie komplexity. Univerzita Karlova v Praze, 11–15.

[19] MANAGEMENT, MARKETING. Vše, co student potřebuje vědět. Komplexita se vztahuje. studentske.eu [online]. [cit. 2017-03-03]. Dostupné z: http://managment-marketing.studentske.eu/2008/06/komplexita-se-vztahuje.html

[20] MCCABE, Thomas. A Complexity Measure. *IEEE Transactions on software engineering*. 1976, **SE-2**(4), 308-320.

[21] MARCHESI, M. OOA metrics for the unified modeling languages. In Proceedings of 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR'98), Palazzo degli Affari, Italy, March, 1998, s. 67-73.

[22] MILLS, Harland D."Mathematical foundations for structured programming,"Federal System Division, IBM Corp., Gaithersburg, MD, FSC 72-6012, 1972.

[23] MOLNÁR, Zdeněk. *Moderní metody řízení informačních systémů*. Praha: Grada, 1992. ISBN 8085623072.

[24] PAVLIC, Mile, Marin KALUZA a Neven VRCEK. *DATABASE COMPLEXITY MEASURING METHOD*. Faculty of Organization and Informatics Varazdin, 2008.

[25] SCHWANDT, Alexander. *Measuring organizational complexity and its impact on organizational performance: A comprehensive conceptual model and empirical study*. Berlin, 2009. Dissertation. Technischen Universität Berlin.

[26] STROUD, J. M. (1956). The fine structure of psychological time. In H. Quastler (Ed.), *Information theory in psychology: problems and methods* (pp. 174-207). New York, NY, US: Free Press.

[27] SPRAGUE, Ralph H. A Framework for the Development of Decision Support Systems. *MIS Quarterly* [online]. 1980, **4**(4), 1- [cit. 2018-03-06]. DOI: 10.2307/248957. ISSN 02767783. Dostupné z: http://www.jstor.org/stable/248957?origin=crossref

[28] ŠMÍD, Vladimír. Pojem informačního systému. *Management informačního systému* [online]. Masarykova univerzita: Fakulta informatiky [cit. 2018-03-06]. Dostupné z: https://www.fi.muni.cz/~smid/mis-infsys.htm

[29] TVRDÍKOVÁ, Milena. EIS – nezbytná součást business intelligence. *Systém online* [online]. Systém online, 2002 [cit. 2018-03-06]. Dostupné z: https://www.systemonline.cz/clanky/eis-nezbytna-soucast-business-intelligence.htm

[30] T. J. Walsh, "A software reliability study using a complexity measure," Managing Requirements Knowledge, International Workshop on(AFIPS), NEW YORK, 1899, pp. 761. doi:10.1109/AFIPS.1979.16