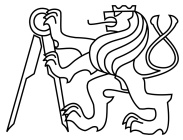
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Nalezení vhodných formátů BPM pro účel optimalizace obchodních procesů
Student:	Bc. Pavel Brabec
Vedoucí:	Ing. Josef Pavlíček, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem diplomové práce je prostudovat dostupné formáty určené k ukládání BPM modelů. Na základě výsledků rešerše a konzultace s vedoucím navrhněte sjednocující formát, který bude sloužit pro implementaci měr kvality.
1. Seznamte se s mírami kvality BPM.
2. Seznamte se s nástroji umožňujícími měření kvality BPM.
3. Seznamte se s formáty pro ukládání BPM procesů: BPMN, XMI, XPDL.
4. Navrhněte snadno rozšiřitelný nástroj pro měření kvality procesů.
5. Implementujte tento nástroj dle návrhu.
6. Nástroj řádně otestujte a zdokumentujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 23. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Nalezení vhodných formátů BPM pro účel optimalizace obchodních procesů

Bc. Pavel Brabec

Vedoucí práce: Ing. Josef Pavlíček, Ph.D.

2. května 2017

Poděkování

Zde bych rád poděkoval rodině a přátelům za podporu při studiu a také panu Ing. Josefu Pavlíčkovi, Ph.D. za vedení této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. května 2017 .....

Odkaz na tuto práci

Brabec, Pavel. *Nalezení vhodných formátů BPM pro účel optimalizace obchodních procesů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá tvorbou nástroje pro analýzu kvality modelů obchodních procesů. Teoretická část se zabývá notacemi pro modelování obchodních procesů, metrikami kvality obchodních procesů a okrajově i kritérii, které by měly kvalitní metriky splňovat. Praktická část popisuje implementaci nástroje Q4BPM, který umožňuje vypočítat metriky kvality pro modely ve formátech BPMN, XMI, XPDL.

Klíčová slova BPM, modely procesů, složitost procesů, metriky procesů, BPMN, XMI, XPDL, XQuery

Abstract

The aim of this thesis is the development of a tool used to analyze the quality of business process models. The theoretical part of the thesis deals with business process modelling notation, business process quality metrics and marginally with criteria that quality metrics should fulfill. The practical part describes the implementation of the Q4BPM tool, which is able to calculate quality metrics for business process models in BPMN, XMI and XPDL formats.

Keywords BPM, process models, process complexity, process metrics, BPMN, XMI, XPDL, XQuery

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době, kdy se každá organizace snaží získat jakoukoli konkurenční výhodu, je velice důležité mít správně nastaveny všechny procesy v organizaci. Správné nastavení procesů může maximalizovat vytížení jednotlivých oddělení nebo minimalizovat komunikaci mezi zaměstnanci a tím šetřit finanční prostředky. S modely procesů pracuje hodně zaměstnanců, proto je důležité, aby tyto modely byly jednoduché, snadno čitelné a pochopitelné. Jednoduché modely se lépe upravují a obsahují méně chyb. Metriky, které jsou popsány v teoretické části práce, jsou nástrojem, který umožňuje měřit složitost procesních modelů. Tyto metriky jsou často odvozeny z metrik složitosti software. Díky tomu, že existuje možnost měřit složitosti procesních modelů se můžeme zabývat jejich zjednodušováním.

Praktická část práce se zabývá vývojem nástroje, který umožňuje automatizovat výpočet metrik složitosti procesních modelů. Nástroj řeší nevýhodu současných řešení jako jsou nízký počet podporovaných vstupních formátů modelů nebo nízký počet implementovaných metrik.

První část práce se zabývá notacemi pro modelování obchodních procesů a formáty, do kterých lze modely exportovat.

Druhá část práce popisuje metriky kvality procesních modelů. Dále tato část definuje kritéria, která by měla kvalitní metrika splňovat.

Třetí část práce popisuje současný stav řešení problematiky a identifikuje slabé stránky a nedostatky současných řešení.

Čtvrtá část práce vychází z identifikovaných nedostatků a zabývá se analýzou a návrhem nově vzniklého nástroje.

Pátá část popisuje implementaci nástroje, použité technologie a problémy, které bylo nutné při implementaci řešit.

Šestá část se zabývá tím, jak byl nově vzniklý nástroj otestován a zdokumentován.

Kapitola 1

Formáty pro ukládání BPM procesů

Pro modelování BPM diagramů je dostupná široká škála nástrojů. Existují jak placené nástroje tak zdarma dostupné nebo dokonce open-source alternativy. Většina nástrojů pro modelování BPM jsou desktopové aplikace. Lze ale nalézt i webové varianty.

Modelovací nástroje umožňují modely procesů exportovat do standardizovaných formátů. Pro implementaci nástroje pro výpočet metrik složitosti procesních modelů je důležité těmto formátům porozumět, protože právě tyto exportované soubory budou vstupem tohoto nástroje.

1.1 Business Process Model and Notation

Business Process Model and Notation[5] (BPMN) je grafický jazyk a notace pro určená pro modelování procesních diagramů obchodních procesů. V současné době je BPMN nejrozšířenějším jazykem pro modelování obchodních procesů a je považováno za nepsaný standard v této oblasti.[6] Jeho hlavním cílem je poskytnout notaci, která je snadno pochopitelná pro všechny účastníky projektů od business analytiků, kteří vytvoří první návrh procesu až po vývojáře, kteří budou podle těchto návrhů implementovat softwarový produkt. Dalším cílem BPMN je komunikační prostředek mezi business a technickými účastníky projektů.[7]

BPMN 1.0 bylo vyvinuto organizací BPMI (Business Process Management Institute). V roce 2004 se BPMI sloučilo s OMG (Object Management Group), která jej nepřestala vyvíjet a následně ho v únoru 2006 přijala za standard. Postupně byly vydány verze BPMN 1.1 a 1.2. V červnu 2010 byla vydána formální specifikace BPMN 2.0, která přinesla řadu rozšíření. V současné době je aktuální verze 2.0.2.

BPMN poskytuje organizacím užitečný nástroj pro pochopení vlastních

procesů a jejich modelování standardní cestou. Notace BPMN je jednoduchá a intuitivní, ale zároveň je schopna zachytit i komplexní procesy. Díky tomu usnadňuje komunikaci mezi všemi osobami zainteresovanými v organizaci nebo projektu.

1.1.1 Základní elementy jazyka BPMN

Účelem této části textu je stručné přiblížení elementů jazyka BPMN. Podle oficiální referenční příručky BPMN lze všechny elementy jazyka BPMN rozdělit do pěti skupin. Tyto skupiny jsou popsány v další části textu. Podrobnější popis a další elementy jazyka lze nalézt v BPMN 2.0 plakátu[1] nebo v referenční příručce jazyka. [2]

1.1.1.1 Flow Objects

Flow objects jsou objekty pro řízení toku jsou základními grafickými prvky, které definují jak se bude proces chovat. Objeky pro řízení toku můžeme rozdělit do tří skupin.

Event Událost je něco, co se stalo v průběhu procesu. Události mají obvykle příčinu nebo dopad. Termín *event* je dostatečně obecný pro pokrytí mnoha aspektů procesu jako je např: začátek aktivity, ukončení aktivity, začátek nebo konec samotného procesu.

Activity Aktivita je určitá činnost, která má být vykonána v rámci procesu. Aktivity mohou být atomické nebo neatomické. Typy aktivit jsou *task*, *subprocess* and *call activity*, který umožňuje znovupoužití tasků a procesů.

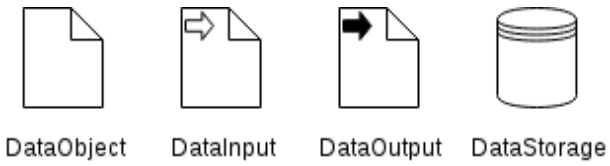
Gateway Gateways, neboli brány, jsou využívány pro kontrolu toku procesem. Brány umožňují tok procesem rozdělit a sloučit. Každá brána může mít více vstupů i výstupů a tak jedna brána může reprezentovat sloučení i rozdělení toku procesu. Návrháři procesů i modelovací nástroje doporučují pro lepší čitelnost procesu, aby každá brána zastávala pouze jednu z těchto funkcí. V případě, že proces nepotřebuje tok rozdělovat nebo slučovat, brány nemusí být použity. Pojem brána vychází z toho, že tyto elementy poskytují mechanizmus, který umožňuje povolit nebo zakázat průchod. Jedním z hlavních rozdílů oproti aktivitám je, že brány nepředstavují úkon, který má být vykonán.

[1] `http://www.bpmb.de/images/BPMN2_0_Poster_EN.pdf`
[2] `http://www.omg.org/spec/BPMN/2.0.2/PDF`

Obrázek 1.1: Ukázka základních Flow objektů jazyka BPMN

1.1.1.2 Data

Jedním ze základních požadavků procesního modelovaní je možnost pracovat s položkami (fyzické předměty nebo informace), které mohou být během procesu vytvořeny, změněny nebo využity procesem. Tento požadavek je v BPMN realizován několika konstrukty jako jsou Data Objects, Data Inputs, Data Outputs, Data Stores.



Obrázek 1.2: Ukázka Data objektů jazyka BPMN

1.1.1.3 Connecting Objects

BPMN nabízí čtyři možnosti určení vztahu mezi elementy:

Sequence Flows Spojuje objekty v pořadí, ve kterém budou jednotlivé aktivity procesu spouštěny. Každý sequence flow má právě jeden výchozí bod a právě jeden cílový bod. Sequence flow je reprezentován nepřerušovanou čárou, která je zakončená vyplněnou šipkou. Sequence flow může překročit hranice lanes, ale nemohou překročit hranice pools.

Message Flows značí tok zprávy mezi odesílatelem a adresátem. Message flows je reprezentován přerušovanou čárou zakončenou nevyplněnou šipkou.

Associations jsou objekty spojující Artefakty s Flow Objekty. Asociace jsou reprezentovány tečkovanou čárou zakončenou šipkou, která označuje směr asociace.

Data Associations Datové asociace jsou modelují přemisťování dat mezi datovými objekty. Datové asociace používají stejnou notaci jako asociace.

Refereční příručka [6] doporučuje, aby procesy byly modelovány tak, aby jejich pochopení bylo co nejsnadnější a zároveň apeluje na dodržování best-practices. Směr toku procesem by měl být modelován shora dolů nebo zprava doleva a toky zpráv by měly být kolmé na tok procesu.

1.1.1.4 Swimlanes

Pro znázornění odpovědností využívá BPMN koncept takzvaných Swimlanes, který odpovědnosti rozděluje do bazénů (pool) a plaveckých drah (lane).

Pools jsou grafickou reprezentací účastníka procesu. Mohou být modelovány jako „black-box“.

Lanes umožňují přiřazení aktivit specifické roli nebo oddělení. Lanes jsou podčástí pools.

Bazény a plavecké dráhy se znázorňují vždy horizontálně nebo vertikálně a každá dráha by měla pokrývat celou délku bazénu.

Obrázek 1.3: Ukázka Swimlanes

1.1.1.5 Artifacts

BPMN umožňuje návrhářům přidat další informace, které přímo nesouvisejí se sekvenčním tokem nebo tokem zpráv v procesu. Tuto možnost v BPMN implementují artefakty skupina a textová poznámka.

Group Slučování elementů do skupin může model procesu zpřehlednit, ale nemá žádný vliv na sekvenční tok procesu.

Text annotation Poznámky také nemají žádný vliv na proces pouze poskytují další informace pro čtenáře modelu procesu.

1.1.2 Formát exportovaných BPMN modelů

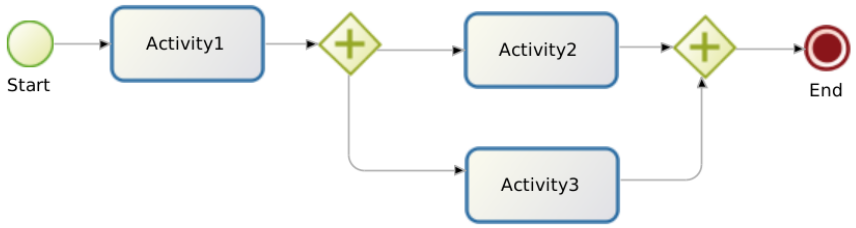
Většina modelovacích nástrojů pracujících s BPMN modely umožňuje modely ukládat jak v interním formátu, se kterým umí pracovat často jen jeden konkrétní nástroj, tak je exportovat do společně definovaného formátu BPMN. Díky exportu do společně definovaného formátu lze modely procesů přenášet mezi nástroji různých výrobců. Většina těchto formátů je založena na XML, tudíž je lze zpracovávat automatizovaně nebo snadno využít ve vlastním nástroji. Na ukázce XML kódu vidíme reprezentaci modelu vyobrazeného na obrázku 1.4 po exportu z modelovacího nástroje Bonita BPM.

```
<?xml version="1.0" encoding="UTF-8"?>
<model:definitions
xmlns:model="http://www.omg.org/spec/BPMN/20100524/MODEL">
 <model:process id="9r-Xof5X">
  <model:startEvent id="AMyHg" name="Start"/>
  <model:endEvent id="D8qSk" name="End" />
  <model:task id="OjVg4" name="Activity1"/>
  <model:task id="W9Ahk" name="Activity3"/>
  <model:task id="eVbgo" name="Activity2"/>
  <model:parallelGateway id="j01TI" name="Gateway"/>
  <model:parallelGateway id="kkE-w" name="Join"/>
  <model:sequenceFlow id="apUbQ" sourceRef="AMyHg"
      targetRef="OjVg4" />
  <model:sequenceFlow id="affkA" sourceRef="OjVg4"
      targetRef="j01TI"/>
  <model:sequenceFlow id="hcPCe" sourceRef="j01TI"
      targetRef="eVbgo"/>
  <model:sequenceFlow id="itcpw" sourceRef="eVbgo"
      targetRef="kkE-w"/>
  <model:sequenceFlow id="kVSxU" sourceRef="kkE-w"
      targetRef="D8qSk"/>
  <model:sequenceFlow id="ManYd" sourceRef="j01TI"
      targetRef="W9Ahk"/>
  <model:sequenceFlow id="vOIjO" sourceRef="W9Ahk"
      targetRef="kkE-w"/>
 </model:process>
</model:definitions>
```

Ukázka je do značné míry pouze ilustrativní. Ve skutečnosti exportovaný soubor obsahuje mnohem více informací. Velká část je ale pro měření kvality diagramu bezvýznamná, proto nejsou v rámci přehlednosti ukázky zachyceny.

Jedná se hlavně o informace týkající se grafického vzhledu diagramu jako jsou souřadnice nebo barva jednotlivých elementů.

Pro analýzu složitosti diagram je nejdůležitější namespace *model*. Tento namespace zachycuje model procesu jako graf a poskytuje všechny důležité informace potřebné pro analýzu složitosti procesu. Element *process* poskytuje kontejner, ve kterém může být model procesu vytvořen. Elementy *startEven* a *endEvent* reprezentují odpovídající události. Element *task* reprezentuje aktivitu. Element *parallelGateway* reprezentuje GatewayAND, další typy bran mohou reprezentovat elementy *exclusiveGateway* nebo *inclusiveGateway* pro GatewayXOR respektive GatewayOR. Posledním elementem v ukázce je element *sequenceFlow*. Podle referenční příručky BPMN musí mít *sequenceFlow* vždy právě jeden výchozí element a právě jeden cílový element. Tato skutečnost je v elementu *sequenceFlow* zachycena atributy *sourceRef* a *targetRef*. Každý element má atribut *id*, ve kterém je uveden jeho jednoznačný identifikátor.



Obrázek 1.4: Zjednodušená ukázka BPMN procesu

1.2 XML Metadata Interchange

XML Metadata Interchange (XMI) [8] je množina pravidel, které definují mapování z MetaObject Facility (MOF) [9] do XML. MOF je průmyslový standard skupiny OMG pro export různých modelů z aplikací. MOF umožňuje modely přenášet mezi aplikacemi různých výrobců, přenášet po síti, skladovat nebo modely renderovant do jiných formátů, jako je například XML. Toto mapování je využito pro exporty Unified Modeling Language (UML) modelů z modelovacích nástrojů. V další části textu se budeme zabývat hlavně *Activity diagramy* jazyka UML a způsobem jejich exportu.[10]

1.2.1 Unified Modeling Language

UML[11] je dalším standardem skupiny OMG, poskytuje nástroj pro specifikaci, návrh a dokumentování softwarových systémů. Dalším využitím UML

může být modelování obchodních procesů nebo návrh jiných, než pouze softwarových systémů. Jazyk UML vznikl proto, aby sjednotil nejlepší existující postupy modelovacích technik a softwarového inženýrství. Je navržen tak, aby jej mohly implementovat všechny CASE (computer-aided software engineering). Jazyk UML není svázaný s žádnou specifickou metodikou vývoje softwaru. [12] UML je podporováno velkým množství modelovacích nástrojů jak zdarma dostupných tak placených jako je například Enterprise Architect společnosti Sparx Systems.

Nejnovější verze UML je 2.5 a byla vydána v červnu 2015. Verze UML 2.4.1 byla v roce 2012 uznána Mezinárodní organizací pro normalnizaci (ISO) jako normy ISO/IEC 19505-1 a 19505-2.[11]

Podle [13] UML definuje třináct typů diagramů, které lze rozdělit do třech skupin:

Structure Diagrams reprezentující statickou strukturu aplikace: Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram a Deployment Diagram

Behaviour Diagrams reprezentující chování aplikace: Use Case Diagram, Activity Diagram a State Machine Diagram

Interaction Diagrams zpřesňují diagramy chování: Timing Diagram, Communication Diagram, Sequence Diagram a Interaction Overview Diagram.

Ze všech výše popsaných diagramů je pro návrh nástroje pro měření kvalit business procesů zajímavý hlavně Activity Diagram, a proto se jím budeme zabývat v další části textu.

1.2.2 UML Activity Diagram

Diagramy aktivit jsou „objektově orientovanými vývojovými digramy“. Díky nim lze procesy modelovat jako aktivity, které jsou složeny z uzlů a hran. Diagramy aktivit lze využít pro modelování business procesů a pracovních postupů.

Diagramy aktivit mají poměrně intuitivní sémantiku, která je založena na Petriho sítích. Chování diagramů aktivit je modelováno pomocí tokenů. Tokeny reprezentují stav, ve kterém se proces právě nachází. V diagramech aktivit mohou tokeny zastupovat postup řízení, objekt, určitá data. Cesta, kterou se mohou tokeny pohybovat, je určena strukturou diagramu. Pohyb tokenu je závislý na přechodových podmínkách a může k němu dojít pouze při splnění všech podmínek. Typicky se jedná o následující podmínky:

- výstupní podmínky zdrojového uzlu
- kontrolní podmínky přechodu přes hranu

- vstupní podmínky cílového uzlu

Základním stavebním kamenem diagramů aktivit tedy jsou uzly a hrany. Rozlišujeme následující tři typy uzlů: akční uzly, řídící uzly, objektové uzly a dva typy hran: řídící hrany a objektové hrany. [12]

Pro rozdělení odpovědností za jednotlivé činnosti využívají UML Activity Diagramy koncept swimlanes stejně jako BPMN.

1.2.2.1 Action nodes

Akční uzly zastupují samostatné jednotky, které jsou v rámci aktivity nedělitelné. Akční uzel je spuštěn pokud token projde všemi vstupními hranami a zároveň jsou splněny všechy podmínky akčního uzlu. Po dokončení akce se kontroluje výstupní podmínka, pokud je tato podmínka splněna jsou vyslány tokeny do všech výstupních hran akčního uzlu. UML Activity diagram rozlišuje následující typy akčních uzlů [12]:

Action iniciuje akci, chování nebo operaci

Send Event odešle událost asynchronně. Při tvorbě události může přijímat vstupní argumenty.

Receive Event čeká na přijetí události. Je aktivován jakmile se objeví token na jeho vstupní hraně. Nemá-li vstupní hranu spouští se v okamžiku přijetí události.

Timed event přijímá časovou událost (reaguje na čas).

Na obrázku 1.5 jsou vyobrazeny ilustrativní ukázky UML action nodes.

Obrázek 1.5: Ukázka UML Action Nodes

1.2.2.2 Control nodes

Řídící uzly řídí postup v rámci aktivity.

Initial node je bodem, ve kterém aktivita začíná. Pokud má aktivita více počátečních uzlů tak každý počáteční uzel vygeneruje token. Tyto tokeny následně aktivitou proudí najednou.

Flow final ukončuje cestu tokenu, který na něj dorazí. Ostatní tokeny zůstavají nedotčeny.

Final ukončuje celou aktivitu.

Decision umožňuje tokenu přejít přes výstupní hranu pokud je splněna její kontrolní podmínka. Důležité je, že token může pokračovat maximálně jednou hranou, a proto je nutné, aby se podmínky vzájemně vylučovaly.

Merge přenáší tokeny ze vstupních hran na jednu výstupní hranu. Uzel sloučení má vždy právě jednu výstupní hranu a několik vstupních hran.

Fork umožňuje rozvětvit cestu do několika souběžných cest. Tokeny, které se dostanou na vstupní hranu *Fork* uzlu jsou duplikovány na všechny jeho výstupní hrany. Tím dojde k rozdělení toku na několik paralelních větví. Každá výstupní hrana může mít vlastní kontrolní podmínka a token tak přes ni může přejít jen v případě, že tuto podmínku splňuje. *Fork* uzly mají několik výstupních hran a právě jednu vstupní.

Join synchronizuje paralelně běžící větve zpět do jedné cesty. Vždy čeká, až dorazí všechny tokeny, které byly vyslány uzlem rozvětvení. Uzel spojení má několik vstupních a právě jednu výstupní hranu.

Na obrázku 1.6 jsou vyobrazeny ilustrativní ukázky UML action nodes.

Obrázek 1.6: Ukázka UML Control Nodes

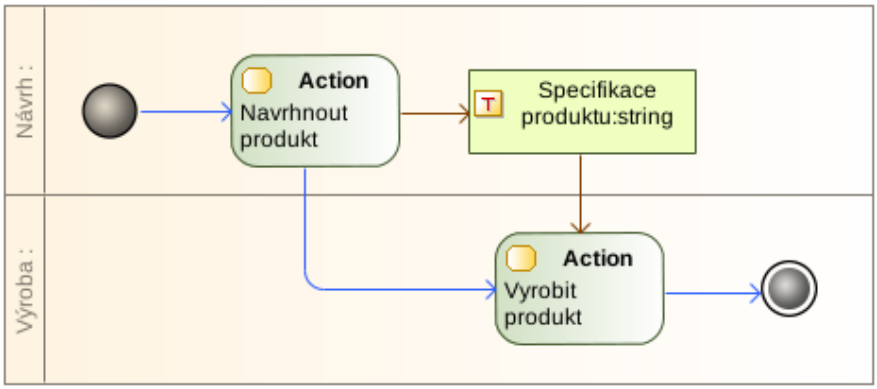
1.2.2.3 Object nodes

Objektové uzly zastupují objekty použité v rámci aktivity. Jsou označeny názvem klasifikátoru a reprezentují jeho instance nebo instance jeho podtříd. Vstupní a výstupní hrany objektových uzlů se označují jako *cesty objektu*. Jde o další typ cesty znázorňující postup objektů uvnitř aktivity. Objekty mohou být vytvářeny a odebírány akčními uzly. [12] Na obrázku 1.7 je vyobrazen proces využívající jeden object node.

1.2.2.4 Edges

Hrany znázorňují cesty uvnitř aktivit. V activity diagramech jsou definovány dva typy hran.

1. Řídící hrany, které reprezentují možné přechody tokenů mezi uzly.



Obrázek 1.7: Ukázka UML Object Node

2. Objektové hrany, které reprezentují cesty objektů v rámci aktivity.

1.2.3 Formát exportu UML Activity diagramů

Následující fragmenty XML kódu zachycují export UML Activity diagramu ve formátu XMI. XMI u každého uzlu zachycuje jak jeho vlastní identifikátor tak identifikátory vstupních i výstupních hran. Podobná situace je i u hran. Každá hrana má vlastní identifikátor i identifikátory počátečního a cílového uzle. Identifikátory mají v původním exportovaném souboru délku přibližně 28 znaků, v ukázce jsou pro přehlednost zkráceny na pět znaků. Z ukázky vidíme, že všechny uzly jsou reprezentovány elementem *node* a teprve atribut *type* určuje jejich typ. To je rozdílem od BPMN, kde konkrétní typ uzlu určuje již název elementu. Dalším rozdílem oproti BPMN je přítomnost atributů *outgoing* a *ingoing* u elementu node. Ty to atributy jsou u tohoto elementu zbytečné, protože pouze duplikují informace, které již jsou obsaženy elementu *edge*.

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1"
 xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
 xmlns:uml="http://schema.omg.org/spec/UML/2.1">
...
  <!-- Initial node -->
  <node xmi:type="uml:InitialNode" xmi:id="nxwKk"
    name="Initial Node" outgoing="gOOgk"/>
  <!-- Actvity Final Node -->
  <node xmi:type="uml:ActivityFinalNode" xmi:id="fnzgK"
    name="Activity Final Node" incoming="gO2AK"/>
  <!-- Action -->
```

```
  <node xmi:type="uml:OpaqueAction" xmi:id="fnyAK"
   name="Action1" outgoing="gO1AK" incoming="gO2gK">
  <!-- Edge -->
  <edge xmi:type="uml:ControlFlow" xmi:id="gO1gK" name="CtrlFlow"
    source="fnyQK" target="fnzAK"/>
  <!-- Fork Node -->
  <node xmi:type="uml:ForkNode" xmi:id="fnywK" name="Fork/Join"
   outgoing="gO2gK gO3AK" incoming="gOOAK"/>
  <!-- JoinNode -->
  <node xmi:type="uml:JoinNode" xmi:id="fnzAK" name="Fork/Join1"
   outgoing="gO3gK" incoming="gO1AK 1gKkE">
...
</uml:Model>
```

1.3 XML Process Definition Language

XML Process Definition Language (XPDL) [14] je standardizovaný formát pro přenos modelů obchodních procesů navržený skupinou Workflow Management Coalition (WfMC). XPDL definuje různé koncepty jako jsou činnosti v rámci procesu, řídící prvky, práce s daty rolemi, událostmi a vyjímkami. Protože stále větší části organizací pracují denně s procesními modely, bylo nutné navrhnout standardizovaný formát, který umožní uživatelům pracovat s modely procesů v nástrojích od různých výrobců nebo každému uživateli dát možnost vybrat si takový nástroj, který mu pro jeho práci nejvíce vyhovuje. Může se jednat o nástroje pro modelování, optimalizaci nebo simulaci obchodních procesů. [14] [15]

Hlavním účelem XPDL je poskytnout formát pro serializaci BPMN modelů. Na rozdíl od BPMN XPDL řeší i zpětnou kompatibilitu mezi diagramy uloženými v různých verzích notace BPMN. Jednou z klíčových vlastností XPDL je jeho rozšiřitelnost o informace, které používají nástroje různých výrobců. [16] Tato vlastnost ale podle Richarda Macha vede k překážkám při praktické analýze modelů uložených v tomto formátu. [17]

Kapitola 2

Míry kvality BPM

Problematikou měr kvality business procesů se zabývá pouze velmi málo vědeckých publikací. Proto lze tuto považovat za ne zcela probádanou oblast. Z literární rešerše vyplývá, že některé míry kvality business procesů budou fungovat na podobných principech, jako jejich protějšci ze světa měr softwarové kvality a složitosti. Oblastí měr softwarové kvality a složitosti se zabývá mnohonásobně více vědeckých publikaci než mírami kvality business procesů. Proto je vhodné se před studiem měr kvality business procesů seznámit alespoň se základními měrami softwarové kvality.

Podobně jako se software skládá z částí jako třída, metoda nebo proměnná tak business procesy se skládájí z procesů aktivit a datových objektů. Mapování entit, ze kterých se skládájí softwarové programy na entity vytvářející business procesy lze popsat následující tabulkou 2.1. [4]

Tabulka 2.1: Podobnost entit BPMN a OOP [4]

OOP	**BPMN**
Třída/balíček	proces, subproces
Proměnná/konstanta	Datový objekt
Komentář	Anotace
Rozhraní třídy	Rozhraní procesu/subprocesu: množina aktivit procesu, které přijímají nebo posílají zprávu
Lokální proměnná	Datové objekty procesu: datové objeky přiřazené k procesu anotací
Třídní proměnná	Datové objekty používané aktivitami procesu: datové objekty přiřazené zprávám, které prochází aktivitami procesu
Volání metody	Přijetí sekvence nebo zprávy aktivitou procesu

2.1 Význam metrik kvality BPM

V dnešní době jsou procesy součástí každé větší organizace. Procesy velkou měrou ovlivňují chod celé organizace a kvalitu služeb, které poskytuje. Správným nastavením procesů lze v organizaci ušetřit nemalé finanční částky tím, že se minimalizuje komunikace mezi zaměstnanci, zvýší se vytížení jednotlivých zaměstnanců nebo oddělení. Dalšími výhodami správně nastavených procesů může být zvýšení kvality služeb nebo zkrácení doby dodání. Díky tomu může organizace získat konkurenční výhodu. V některých společnostech se mohou vyskytovat i velice sofistikované procesy, které jsou náročné na analýzu, porozumění i opravu chyb. Business procesy často slouží také jako komunikační nástroj mezi všemi účastníky projektu jako jsou experti na příslušnou doménu, analytici obchodních procesů, architekti a vývojáři softwaru. Z výše uvedených důvodů je zřejmé, že v zájmu každé společnosti je mít procesy co nejjednodušší, tak aby je všichni zúčastnění snadno pochopili. Prvním krokem k tomu, abychom mohli modelovat snadno pochopitelné business procesy je definicice toho, co vlastně fráze „složitý model“ znamená. Podle IEEE Standard Computer dictionary výraz *složitý* znamená: „The degree to which a system or component has a design or implementation that is difficult to understand and verify“. IEEE definuje výraz *složitý* jako stupeň složitosti pochopení a ověření daného modelu.[18] [19]

Metriky kvality BPM procesů nám mohou dávat návod, jak procesy modelovat tak, aby jejich složitost byla co nejnižší. Dále nám tyto metriky umožňují vybrat v každé organizace nejsložitější procesy, které by bylo vhodné zjednodušit. [20]

2.2 Kritéria metrik kvality BPM

Následující část textu se zabývá různými požadavky na procesní míry, které jsou popsány v současné literatuře. Protože tyto metriky splňují tato kritéria, která byla diskutována v seriózní vědecké literatuře, mají jejich výsledky větší význam jak z vědeckého, tak z praktického pohledu.

Jorge Cardoso ve své práci Business Process Control-Flow Complexity: Metric, Evaluation, and Validation [1] stanovuje, že kvalitní metrika by měla dodržovat množinu jednoduchých ale důležitých vlastností. Tyto vlastnosti jsou:

- **Jednoduchost** - Metrika by měla být snadno pochopitelná jejími koncovými uživateli tj. procesními analytiky a návrháři.
- **Konzistentnost** - Metrika by měla vždy vracet stejný výsledek pokud dva nezávislí uživatelé aplikují metriku na shodné procesy.
- **Automatizace** - Musí být možné automatizovat výpočet dané metriky.

- **Aditivnost** - Pokud dva nezávislé procesy zřetězíme, tak složitost výsledného procesu musí být vyšší nebo shodná jako součet původních procesů.
- **Interoperabilita** - Díky tomu, že BPM diagramy lze popsat v mnoha jazycích je důležité, aby metrika nebyla závislá na jazyce.

Dalším autorem, který zavádí kritéria pro metriky kvalit BPM je Latva-Koivisto. Podle Latva-Koivista je důležité aby metriky splňovaly následující vlastnosti:

- **Validita** - metrika měří pouze to, co má měřit.
- **Spolehlivost** - výsledky měření získané různými měřeními stejného procesu jsou shodné
- **Vyčíslitelnost** - pro metriku existuje počítačový program, který umí tuto metriku vyhodnotit v konečném čase.
- **Snadnost implementace** - složitost implementace metriky je v rozumných mezích
- **Intuitivnost** - Pochopení definice metriky je snadné a je jednoduše poznat jak metrika souvisí se složitostí.
- **Nezávislost** - v ideálním případě je metrika nezávislá na ostatních vlastnostech procesu, které mohou být předmětem měření ostatních metrik. Toto platí alespoň pro velikost procesu a jeho grafickou reprezentaci.

Vidíme, že kritéria popsaná oběma autory jsou si velice podobná. Dalším autorem, který se zabývá metrikami kvality je Elaine Weyuker. Weyukerová formalizovala devět vlastností, které by měly splňovat míry kvality softwaru. Tyto vlastnosti dávají základ i metrikám kvalit BPM. Vlastnosti metrik podle Weyukerové by měly odhalovat silné a slabé stránky již existujících měr kvality a pomoci při návrhu nových měr. Práce Weyukerové je známá pro svůj formální přístup a je to jedna z nejcitovanějších prací v oblasti problematiky měr složitosti. Práce byla seriózně diskutována v odborné literatuře [21]. Někteří autoři práci ale kritizují, protože nikdy nebylo jasné, jestli by všechny metriky měly splňovat všechny vlastnosti podle Weyukerové. [22] [23]

Vlastnost 1 Dobrá metrika je taková metrika, která dokáže rozlišit dva procesy tak, že je nehodnotí stejnými naměřenými hodnotami.

Vlastnost 2 Změna procesu musí způsobit změnu jeho složitosti. Dobrá metrika by měla být schopna tuto změnu detekovat.

Vlastnost 3 Pokud existují dva různé procesy, které mají shodné datové typy a hodnoty, ale jména proměnných jsou různá, tak dobrá metrika by měla vrátit stejné hodnoty složitosti.

Vlastnost 4 Dva procesy mohou vypadat stejně, ale může se lišit jejich vnitřní struktura. Dobrá metrika by měla umět tyto dva procesy rozlišit na základě jejich interní struktury.

Vlastnost 5 Pokud je proces složen ze dvou interagujících procesů musí být složitost výsledného procesu větší nebo rovna součtu složitosti interagujících procesů.

Vlastnost 6 Je možné mít dva identické procesy, ale pokud tyto dva procesy spojíme do třetího procesu složitosti výsledných procesů nebudou shodné. Spojením dvou procesů vzniká potenciální možnost zvýšení složitosti nad rámec originálních procesů. Dobrá metrika by měla být schopna odlišit tyto procesy.

Vlastnost 7 Výsledná složitost procesu je závislá na pořadí jeho prvků. Dva shodné procesy mohou mít rozdílnou složitost pokud změníme pořadí jejich prvků. Dobrá metrika by měla být schopna tyto změny detekovat.

Vlastnost 8 Pokud se dva procesy liší pouze ve výběru názvů pro různé elementy, tak tyto dva procesy jsou shodné. Dobrá metrika by měla pro oba procesy vracet stejnou hodnotu složitosti.

Vlastnost 9 Pokud části dvou procesů společně interagují, musí tato skutečnost způsobit navýšení složitosti. Dobrá metrika by měla tuto změnu detekovat.

2.3 Základní metriky kvality softwaru

Před započetím studia metrik kvality business diagramů je vhodné se seznámit alespoň se základními metrikami kvality softwaru. Mezi metrikami složitost softwaru a business diagramů je silná analogie. Tuto skutečnost dokumentuje tabulka 2.1. Problematikou měření kvality softwaru se však odborníci zabývali již od konce 70. let 20. století [24], a proto je tato oblast lépe probádaná.

Současná softwarová díla v praxi jsou velice rozsáhlá a obsahují mnoho stavů a větvení. Díky metrikám složitosti softwaru, lze tyto vlastnosti měřit a následně pracovat na zjednodušení softwaru a významně tak snížit riziko vzniku chyby. Podle Gregora Polančiče a Blaže Cegnara [25] největší počet měr složitosi procesů vychází ze softwarových měr *Lines of code* a *Cyclomatic complexity*.

2.3.1 Lines of code

Metrika lines of code, zkráceně LOC, je jednou z nejjednodušších metrik pro měření velikosti softwaru. Jak již z jejího názvu vyplývá tato metrika měří počet řádků zdrojového kódu. Význam hodnot naměřených touto metrikou

je velice diskutabilní, protože do výsledku jsou zahrnuty i prázdné řádky, komentáře, deklarace proměnných.

$$LOC = počet\ řádků\ zdrojových\ kódů\ softwarového\ díla$$

Protože započítávání komentářů a prázdných řádků je u metriky LOC velký problém, byla navržena metrika Non-Commented LOC, zkráceně NCLOC, která již nezapočítává komentáře a prázdné řádky. Proto mají výsledky metriky NCLOC větší vypovídající hodnotu než LOC.[26]

$$NCLOC = LOC - počet\ prázdných\ řádků - počet\ řádků\ s\ komentáři$$

2.3.2 Cyclomatic complexity

V roce 1985 navrhl Thomas J. McCabe metriku složitosti softwaru zvanou cyklomatická složitost. Jeho základní myšlenkou bylo, že na míře složitosti softwaru se určitou měrou podílejí podmíněné příkazy. Pokud budeme mít dva programy o stejné délce, tak program s větším počtem podmíněných příkazů by měl být složitější. Cyklomatická složitost měří počet lineárně nezávislých cest zdrojovým kódem, proto jeho nízká hodnota značí, že program je snadno pochopitelný a modifikovatelný. Cyklomatická složitost nachází svoje využití i v oblasti testování softwaru, protože udává nejnižší možný počet testů, abychom pokryli všechny cesty programem.[18][27]

McCabe definuje cyklomatickou složitost jako:

$$v = e - n + 2p$$

V tomto výrazu e zastupuje počet hran grafu, n počet uzlů grafu a p je počet komponent grafu.[28]

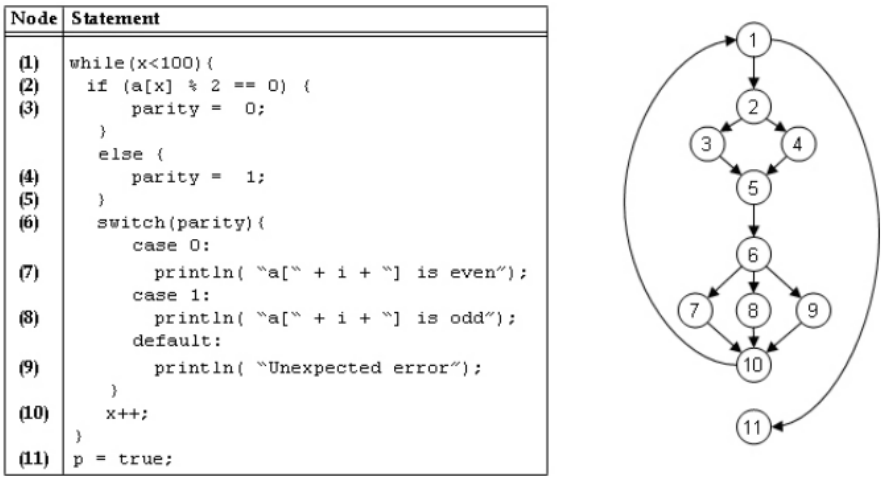
Výpočet cyklomatické složitosti softwarového programu se provádí pomocí takzvaného control-flow grafu, který popisuje logickou strukturu programu. V tomto grafu uzly reprezentují části programu, kde nedochází k žádnému větvení a hrany reprezentují přechod mezi těmito stavy. Protože control-flow graf obsahuje vždy jednu komponentu můžeme pro výpočet MCC používat následující zjednodušení.

$$MCC = e - n + 2$$

Obrázek 2.1 zachycuje převod zdrojového kódu programu na control-flow graf. Control-flow graf tohoto programu obsahuje 14 hran a 11 uzlů, tudíž cyklomatická složitost tohoto programu je rovna 5.[1]

2.4 Přehled měr kvality BPM

V této kapitole se seznámíme s několika vybranými metrikami složitosti, které jsou popsány v současné odborné literatuře. Podle Irene Vanderfeesten, Jorge Cardosa a kolektivu [29] můžeme kvalitu návrhu rozdělit na pět principů. Pro každý z těchto principů se budeme zabývat jeho metrikami.



Node	Statement
(1)	while(x<100){
(2)	if (a[x] % 2 == 0) {
(3)	parity = 0;
	}
	else {
(4)	parity = 1;
(5)	}
(6)	switch(parity){
	case 0:
(7)	println("a[" + i + "] is even");
	case 1:
(8)	println("a[" + i + "] is odd");
	default:
(9)	println("Unexpected error");
	}
(10)	x++;
	}
(11)	p = true;

Obrázek 2.1: Převod programu na control-flow graf [1]

- Size
- Complexity
- Modularity
- Coupling
- Cohesion

Slovinští výzkumníci Polančič a Cegnar ve své rozsáhlé literární rešerši [25] identifikovali celkem 66 různých metrik složitosti procesních modelů. Tento výzkum je velice aktuální, byl vydán v prosinci 2016. V dalších částech práce jej budu využívat jako přehled existujících metrik.

2.4.1 Size

Size metriky se zabývající velikostí díla. Tyto metriky si zakládájí na hypotéze, že větší model bude obsahovat větší počet chyb a budou náročnější na pochopení. Přesto, že tyto metriky jsou velice jednoduché, jsou i důležité. Představme si proces, který má 50 lineárně zřetězených aktivit. Tento proces by měl Control-flow complexity rovnou 0, ale na pochopení by jednoduchý rozhodně být nemusel.

Number of activities Metrika Number of activities, zkráceně NOA, vychází ze softwarové míry Lines of code. Je důležité poznamenat, že metrika NOA vychází pouze z jednoho úhlu pohledu na model a tím je jeho

velikost. NOA nebere v úvahu funkčnost nebo složitost modelu.[30]. Někdy může být tato metrika označována jako Number of tasks, zkráceně NOT. [25]

$$NOA = počet\ aktivit\ procesu$$

Number of activities and control-flow elements NOAC je další metrikou, která je odvozená z LOC. NOAC přidává k počtu aktivit modelu procesu také počet řídících prvků modelu. [4] NOAC lze aplikovat pouze na procesy, které jsou *well-structured*.[30] Pojem *well-structured* definoval van der Aalst takto: „A model is well-structured if the split/join constructions are properly nested“.[31]

$$NOAC = počet\ aktivit\ procesu\ a\ počet\ splitů\ procesu$$

Number of activities splits and joins Bohužel, ne všechny procesy splňují definici *well-structured*. Proto byla zavedena další metrika Number of activities splits and joins, zkráceně NOA. Tato metrika měří počet aktivit, splitů a joinů.[30]

$$NOAJS = počet\ aktivit\ procesu\ a\ počet\ splitů,\ joinů\ procesu \tag{2.1}$$

2.4.2 Complexity

Complexity metriky se zabývají složitostí modelu. Model by měl být tak jednoduchý, jak jen je to možné. Složitost roste přímo úměrně s počtem různých cest modelem. Complexity metriky si zakládají na hypotéze, že modely větším počtem cest budou obsahovat větší počet chyb a budou náročnější na pochopení.

Control-flow complexity Metrika Control-flow complexity, zkráceně CFC, vychází z metriky složitosti softwaru MCC. Metrika MCC již byla popsána v kapitole „Základní metriky softwaru“.
Základem pro výpočet CFC jsou větvící prvky, které mají více než jeden výstupní přechod. To jsou AND-splits, OR-splits a XOR-splits. Všechny tyto prvky jsou charakterizovány počtem výstupních cest. V následující části textu budeme označovat písmenem s větvící prvek a dále jako *fan-out(s)* nebo n budeme označovat počet výstupních cest větvícího prvku.

- AND-split aktivuje všechny výstupní cesty paralelně. Protože již před vykonáním AND-splitu je vždy jasné jaké další aktivity budou vykonány, platí:

$$CFC_{AND-split}(s) = 1$$

- OR-split aktivuje jednu nebo více výstupních cest. Protože po vykonání OR-splitu může být aktivován libovolný počet pro jeho control-flow complexity platí:

$$CFC_{OR-split}(s) = 2^n - 1$$

  Z výše uvedeného vzorce jasně vidíme, že právě OR-splits budou s rostoucím *fan-out(s)* přispívat ke složitosti modelu nejvyšší měrou.

- XOR-splits aktivuje právě jednu výstupní cestu. Protože po vykonání XOR-splitu je vždy n možností jak bude proces pokračovat, tak platí, že:

$$CFC_{OR-split}(s) = n$$

Tímto jsme definovali výpočet složitost pro všechny typy splitů, takže nyní již můžeme definovat výpočet control-flow complexity jako:

$$\begin{aligned} CFC(p) = & \sum_{\forall s \in p \ \wedge \ s \ is \ ANDsplit} CFC_{AND-split}(s) + \\ & \sum_{\forall s \in p \ \wedge \ s \ is \ ORsplit} CFC_{OR-split}(s) + \\ & \sum_{\forall s \in p \ \wedge \ s \ is \ XORsplit} CFC_{XOR-split}(s) + \end{aligned}$$

kde značí proces, u kterého chce control-flow complexity měřit.

Halstead-based Process Complexity Metriky navržené Maurice Halsteadem [32] jsou jedny z nejznámějších a nejprostudovanějších metrik softwarové složitosti. Tyto metriky byly navrženy jako prostředek k určení složitosti programu na základě jeho operandů (proměnné, konstanty) a operátorů (aritmetické operátory, klíčová slova jazyka). Halsteadovi metriky jsou založeny na několika jednoduchých pozorování zdrojového kódu.

- n_1 = počet unikátních operátorů (if, while, =, ECHO, atd.)
- n_2 = počet unikátních operandů (proměnné, konstanty)
- N_1 = celkový počet výskytu operátorů
- N_2 = celkový počet výskytu operandů

Pro potřeby měr kvality odvodil Jorge Cardoso [30] následující mapování umožňující výše uvedené hodnoty stanovit pro model procesu.

- n_1 = *počet* unikátních aktivit, splitů, joinů a řídících operátorů
- n_2 = *počet* unikátních datových proměnných se kterými proces pracuje

- $N_1 = celkový$ počet aktivit, splitů, joinů a řídících operátorů
- $N_2 = celkový$ počet datových proměnných se kterými proces pracuje

Těmito jednoduchými definicemi Cardoso zavedl metriku Halstead-based Proces Complexity, zkráceně HPC. HPC slouží pro odhadování délky, velikosti a složitosti procesu. Tyto vlastnosti lze vyčíslit podle následujících pravidel:

$$N = n_1 * \log_2 n_1 + n_2 * \log_2 n_2$$
$$V = (N_1 + N_2) * \log_2(n_1 + n_2)$$
$$D = (n_1/2) * (N_2/n_2)$$

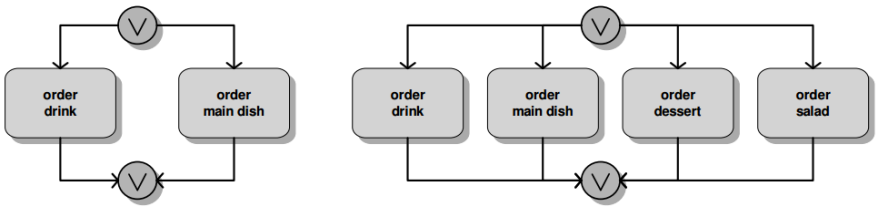
Kde N značí process lenght, V process volume a D process difficulty. Původní Halsteadovi metriky obsahují dále výpočty pro odhad složitosti implementace, času potřebného pro implementaci a odhad pro počet chyb. Cardoso tyto metriky pro modely procesů neodvozoval, protože by bylo nutné je kalibrovat empirickými pokusy.

Metriky HPC mají několik výhod. Nepotřebují analyzovat strukturu procesu, mohou předpovídat chybovost modelu, náročnost na údržbu a jejich výpočet je velice snadný.

Cognitive complexity Shao and Wang [33] definovali cognitive weight jako metriku pro měření potřebného úsilí pro porozumění softwaru. Na základě empirického výzkumu stanovili kognitivní váhy pro řídící struktury programů, které jsou metrikou pro jejich pochopení člověkem. Cognitive weight je definována jako součet kognitivních vah všech řídících struktur softwaru podle tabulky 2.2.

Na obrázku 2.2 vidíme dva modely, které obsahují OR-split, tudíž po vykonání OR-splitu může následovat paralelní spuštění jedné nebo více aktivit. Pokud budeme složitost definovat jako „obtížnost na otestování“ bude nejlepší výsledky podávat metrika CFC. Budeme-li se ale držet definice, že složitost je „obtížnost pochopení“ tak počet výstupů OR-splitu nemá až takový vliv na pochopení této řídící struktury jako mu přikládá CFC. Protože pokud se někdo bude snažit pochopit tento model nebude mu záležet na stupni OR-splitu, ale bude se jej snažit pochopit jako celek.[2]

Volker Gruhn [3] definoval Cognitive weight pro BPM následovně: „The cognitive weight of a BPM is the sum of the cognitive weights of its elements“. Definice cognitive weight pro software a pro BPM jsou tedy velice analogické. Konkrétní hodnoty kongnitivních vah pro BPM jsou definovány v tabulce 2.2.



Obrázek 2.2: Modely s rozdílnou CFC[2]

Tabulka 2.2: Kognitivní váhy BPM elementů [3]

BPM struktura	**Softwarová struktura**	**Kognitivní váha**
navazující kroky procesu	sekvence	1
XOR-split (volba ze dvou větví) s korespondujícím XOR-join	if-then	2
XOR-split (volba alespoň ze tří větví) s korespondujícím XOR-join	switch-case s libovolným počtem zvolených větví	3
AND-split s korespondujícím AND-join	paralelní vykonání programu	4
OR-split s korespondujícím OR-join	swith-case s následným paralelní vykonání programu	7
Subtask	volání funkce	2
Ukončení aktivity (aktivování aktivity ukončí jinou aktivitu)		1
Ukonční větve (aktivování aktivity ukončí ukončí ostatní aktivity ve větvi)	Porovnatelné s voláním funkce	2-3

2.4.3 Modularity

Stupeň modularizace má dopad na kvality návrhu modelu. Vysoká modularizace je nežádoucí stejně jako nízká. Modularity metriky si zakládají na hypotéze, že modely s nízkou modularitou budou obsahovat více chyb.

Interface complexity V roce 1981 publikovali Henry a Kafura[34] metriku složitosti softwaru, která je založena na měření toku informací mezi jednotlivými komponentami. Tato metrika byla nazvána „The procedure complexity“, zkráceně PC. Henry a Kafura pro validaci svoji metriky

využívali zdrojové kódy operačního systému UNIX a přišli na to, že hodnoty naměřené jejich metrikou korelují s počtem změn systému.

Metrika PC byla definována jako:

$$PC = lenght * (fan_{in} * fan_{out})^2$$

Kde fan_{in} je označení pro počet lokálních informací vstupujících do procedury a fan_{out} značí počet informací vystupující z procedury. Proměnná *lenght* značí počet řádků procedury. Alternativně lze místo počtu řádku procedury použít i McCabovu cyklomatickou složitost nebo délku z metrik navržených Maurice Halsteadem. Henry a Kafura tvrdí, že tyto alternativní varianty mají lepší korelaci s počtem chyb v proceduře a časem její implementace.[34]

Jorge Cardoso navrhl adaptaci této metriky pro potřeby BPM následujícím způsobem:

- Pro výpočet délky musíme aktivity procesu rozdělit na white-box a black-box. U black-box aktivit známe pouze jejich interface. Takže není možné stanovit jejich délku. Proto budeme u black-box aktivit předpokládat délku 1. V případě white-box aktivit je jejich délka rovna počtu úloh.
- Hodnoty fan_{in} a fan_{out} mohou být mapovány přímo jako počet vstupů a výstupů aktivity. Aktivita je spuštěna ve chvíli, kdy jsou k dispozici její vstupy (fan_{in}) a je určena ke spuštění. Ve chvíli, kdy je aktivita ukončena jsou její výstupní data (fan_{out}) přenesena k dalším aktivitám.

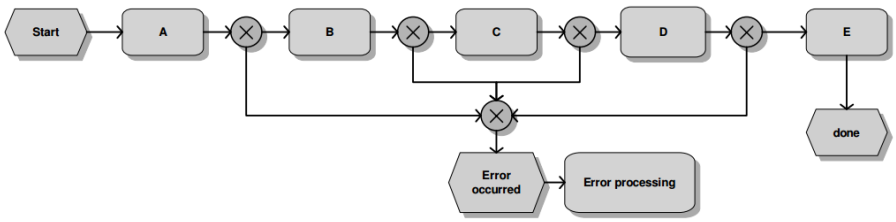
Metriku interface complexity lze vypočítat následujícím způsobem:

$$IC = délka * (počet\ vstupů * počet\ výstupů)^2$$

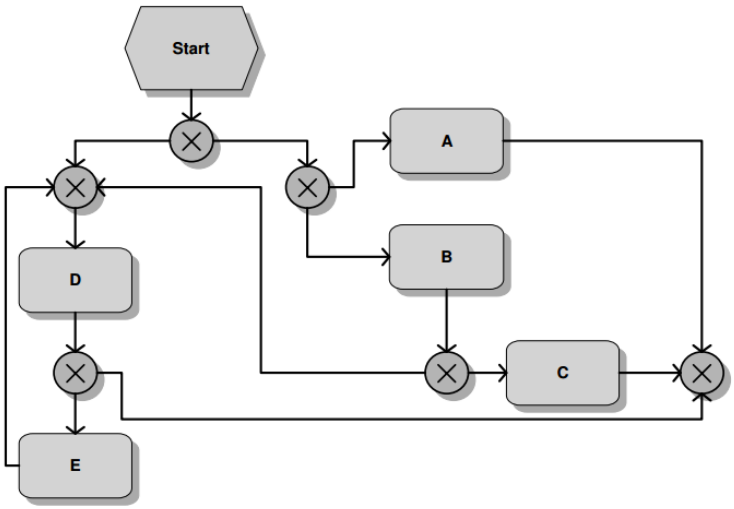
Výhodou metriky interface complexity je, že bere v úvahu data-driven procesy a může být stanovena již při návrhu, dříve než je implementována. Nevýhodou je, že pokud má aktivita fan_{in} nebo fan_{out} roven nule tak je její složitost nulová. Tato situace typicky nastává na konci procesu.

Nesting depth Gruhn [3] zavádí metriku nesting depth, která je odvozena z prací [35, 36] a vychází z faktu, že s rostoucím zanořením roste složitost softwaru i procesních modelů. Nesting depth udává počet rozhodnutí, které je třeba učinit, abychom modelem procesu došli k nejhlouběji zanořenému elementu.

$$MaxND = maximální\ zanoření\ elementu$$



Obrázek 2.3: Model procesu s nesting depth 1. [3]



Obrázek 2.4: Model procesu s nesting depth 3. [3]

Dále Gruhn uvádí, že nesting depth je vhodným doplněním k CFC. Může totiž nastat situace, kterou vidíme na modelech z obrázků 2.3 a 2.4. Protože oba modely obsahují 4 XOR-splity se dvěma výstupy, bude hodnota CFC pro oba modely rovna 8. Model na obrázku 2.4 je ale složitější na pochopení něž model na obrázku 2.3, který je téměř lineární. Tento případ dokáže metrika nesting depth odlišit. Zatím co nesting depth procesu na obrázku 2.3 je pouze 1 (každá chyba vede ke spuštění procesu pro ošetření chyb), tak proces na obrázku 2.4 má nesting depth rovnou 3. Musíme se třikrát rozhodnout než dojdeme k aktivitě C. [3]

Další autoři jako Kluza[37] nebo Mendling[38] zavádí mean nesting depth, která měří průměrnou hloubku zanoření.

$$MeanND = průměrné\ zanoření\ elementu$$

2.4.4 Coupling

Výraz coupling je v IEEE Standard Computer dictionary [19] popsán jako „The manner degree of interdependence between software modules“. Dále tento slovník uvádí, že opačným výrazem k *coupling* je *cohesion*. Coupling metriky měří míru provázanosti mezi jednotlivými moduly.

Coupling u modelů obchodních procesů se zaměřuje na sílu spojení mezi jednotlivými aktivitami.

Výzkumu coupling metrik se prozatím věnoval pouze malý počet výzkumníků. Současná literatura nabízí o coupling metrikách pouze málo informací. [29] Tyto metriky si zakládají na hypotéze, že model se silně provázanými moduly, by měl obsahovat více chyb.

Weighted coupling metric Irene Vanderfeesten, Jorge Cardoso a kolektiv [29] navrhli metriku, která byla inspirována již existujícími coupling metrikami a metrikami z oblasti software. Tato metrika je založena na sčítání vah spojení u všech aktivit a je definována následovně:

$$CP = \frac{\sum_{t_1,t_2 \in T} connected(t_1, t_2)}{|T| * (|T| - 1)}$$

kde $connected(t_1, t_2)$ je definováno jako:

$$\begin{cases} 1 & \text{pokud } (t_1 \to t_2) \wedge (t_1 \neq t_2) \\ 1 & \text{pokud } (t_1 \to AND \to t_2) \wedge (t_1 \neq t_2) \\ connected_{OR}(t_1, t_2) & \text{pokud } (t_1 \to OR \to t_2) \wedge (t_1 \neq t_2) \\ \frac{1}{m*n} & \text{pokud } (t_1 \to XOR \to t_2) \wedge (t_1 \neq t_2) \\ 0 & \text{pokud } (t_1 = t_2) \end{cases}$$

$$connected_{OR}(t_1, t_2) = \frac{1}{(2^m - 1) * (2^n - 1)} + \frac{(2^m - 1) * (2^n - 1) - 1}{(2^m - 1) * (2^n - 1)} * \frac{1}{m * n}$$

Ve výrazech popsaných výše, m zastupuje počet vstupujících přechodů, n počet výstupních přechodů. Symboly t_1 a t_2 značí aktivity procesu T.

Každý přechod mezi aktivitami je ohodnocen váhou. Váha každého přechodu je rovna pravděpodobnosti, že tento přechod bude aktivován. Často není dopředu jasné s jakou pravděpodobností budou přechody aktivovány, proto tato metrika předpokládá rovnoměrné rozdělení. Určení vah pro jednotlivé typy přechodů:

- Nejvyšší váhy mají přechody mezi aktivitami, které jsou rozděleny AND-splitem. Protože pravděpodobnost, že tento přechod bude využit je rovna 1.
- Naopak nejnižší váhu mají přechody, které jsou rozděleny XOR-splitem. Pravděpodobnost, že přechod bude vykonán je rovna $\frac{1}{m*n}$,

kde $\frac{1}{m}$ je pravděpodobnost využití části přechodu $t_1 \rightarrow XOR$ a $\frac{1}{n}$ je pravděpodobnost využití části přechodu $t_2 \rightarrow XOR$.

- Dále musíme definovat váhy přechodů, které jsou rozděleny OR-splitem. U OR-splitu víme jen, že počet využitých výstupů bude mezi 1 a n. Proto váha přechodů, které rozděluje OR-split musí být nižší než u AND-splitu a vyšší než OR- splitu. Proto je váha přechodů přes OR-split určena výrazem pro $connected_{XOR}(t_1, t_2)$.

2.4.5 Cohesion

Výraz cohesion je ve slovníku [19] popsán jako „The manner and degree to which the tasks performed by sigle software module are related to one other".

Cohesion metriky určují míru soudržnosti mezi elementy v jednom modulu. Tyto metriky si zakládají na hypotéze, že moduly s nízkou soudržností obsahují více chyb než moduly s vysokou soudržností.

Oblast metrik kohoze business process modelů je pouze velmi málo prozkoumaná. Pro tuto problematiku existuje velmi omezené množství literatury.

Process cohesion Metrika proces cohesion byla navržena Irene Vanderfeesten a kolektivem.[39] Podobně jako dříve popsané metriky vychází z již existujících měr koheze software. V případě metriky process cohesion nebylo odvození ani z daleka tak přímočaré jako u metrik NOA nebo Halstead-based process complexity.

Pro výpočet metriky autoři zavedli výraz *Activity Cohesion*, který je definovaný jako:

$$c(t) = \lambda(t) * \mu(t)$$

Kde λ označuje *relační kohezi*. Relační koheze určuje jak moc jsou související jednotlivé operace v rámci jedné aktivity.

Dále μ označuje *informační kohezi*, která se zaměřuje na informace, které jsou použity buď na vstupu nebo výstupu nějaké operace aktivity.

Pojmy *relační koheze* a *informační koheze* jsou formálně definovány v [39]. Metrika process cohesion je definována následujícím výrazem:

$$c = \frac{\sum_{t \in T} c(t)}{|T|}$$

Kde T označuje množinu aktivit procesu a t jsou aktivity z T.

Kapitola 3

Nástroje pro měření kvality BPM

V předchozích částech textu jsme se zabývali formáty exportu BPM diagramů a metrikami jejich kvalit. V této části textu se budeme zabývat již existujícími nástroji pro výpočet metrik kvality. Teoreticky definované metriky popisují návod pro pochopení a vyčíslení metriky. Manuální vyčíslení některých metrik, zvláště u digramů s mnoho elementy, může být složité a při ručních výpočtech by velice pravděpodobně docházelo k chybám. Z tohoto důvodu je vhodné výpočet metrik automatizovat.

3.1 CoCoFlow

CoCoFlow[3] je nástroj vyvinutý Irene Vaderfeesten a kolektivem v rámci práce *Evaluating workflow process designs using cohesion and coupling metrics* [39]. CoCoFlow je akronymem slov *COhesion COupling metrics for workFLOW models*. Již z názvu nástroje je patrné, že se bude zabývat pouze Cohesion a Coupling metrikami. Nástroj pracuje se soubory ve formátu XML, u kterých provádí jejich vstupní validaci. V základním distribučním balíku aplikace jsou tři ukázkové soubory pro demonstraci funkčnosti. Uživatelské rozhraní aplikace je rozděleno do tří částí, které se zabývají výpočtem metrik, vizualizací procesu a zobrazením importovaného XML souboru. Nástroj implementuje metriky *Coupling*, *Cohesion* a *Coupling/Cohesion ratio*. CoCoFlow je distribuován jako *exe* soubor. Zdrojové kódy nejsou veřejně dostupné.

[3] `http://is.tm.tue.nl/staff/ivanderfeesten/CoCoFlow/`

3.2 ProM

ProM[4] je rozšiřitelný framework, který podporuje širokou škálu technik disciplíny *proces mining*. Proces mining je disciplína zaměřující se na získávání informací o procesech analýzou log souborů. Jejím cílem je získat různé informace o procesu jako např. výkonost, data, organizace. Funkcionalita ProM se skládá ze zásuvných modulů, které lze rozdělit do kategorií: [40]

Moduly pro import umožňují importovat modely procesů z několika různých formátů jako např. BPMN nebo PNML (Petriho sítě).

Moduly pro dolování implementují funkcionalitu pro process mining, pro který je ProM primárně vyvíjen.

Moduly pro analýzu slouží pro analýzu procesů z mnoha různých pohledů jako např. testování správnosti Petriho sítí nebo kontroly MXML modelů. Tato kategorie zahrnuje i modul pro výpočet měr kvality procesů, který implementuje míry NOA, CFC a density.

Moduly pro konverzi lze využí pro převod modelů mezi různými formáty.

Moduly pro export lze využít pro export objektů z ProM pro použití jinými nástroji.

Nástroj ProM je implementován v jazyce Java a jeho zdrojové kódy jsou veřejně dostupné.

3.3 BPMN Measures

BPMN Measures je nástroj vyvinutý Richardem Machem v rámci jeho diplomové práce. [17] BPMN Measures pracuje se soubory ve formátech XPDL 2.0 a XPDL 2.2 vyexportovanými z nástroje Bizagi Modeler. Importované soubory umožňuje validovat. Tento nástroj implementuje 10 různých metrik kvalit business procesů jako jsou NOA, CFC, MaxND nebo MeanND. Nástroj je implementovan v jazyce Java a jeho funkčnost je rozdělena do třech tříd. Tyto třídy řeší validaci vstupního souboru, napojení na webové služby a vlastní výpočet metrik složitosti.

3.4 BPMN Quality Tool

BPMN Quality Tool je nástroj vyvinutý v rámci práce *A Tool for Evaluating the Quality of Business Process Models*. [41] Nástroj je implementovan v jazyce Java a jeho funkčnost je rozdělena do čtyřech modulů. První modul zvaný *extractor* přijme na vstupu model procesu ve formátu XMI. Tímto je

[4] `http://www.promtools.org/`

zajištěno, že nástroj lze integrovat do ostatních nástrojů používající tento standard. Další modul se nazývá *constructor*. Constructor bere v úvahu perspektivu vybranou uživatelem a generuje strom, který obsahuje všechny elementy modelu procesu, které náleží do vybrané perspektivy. Poslední dva moduly se nazývají *calculator* a *interpret*. Modul calculator implementuje výpočty metrik NOA, CFC, CP, Density a Coeficient of Network Complexity. Interpret zobrazuje výsledky uživateli, umožňuje nastavit priority metrik a prahy optimálních hodnot jednotlivých metrik. Tyto prahy lze nalézt v literatuře jako výsledky empirických výzkumů.[41]

3.5 Vyhodnocení dostupných nástrojů

Při rešerši byly nalezeny celkem čtyři nástroje pro měření kvality BPM. Z těchto čtyř nástrojů jsou pouze dva veřejně dostupné. Jsou to nástroje CoCoFlow a ProM.

CoCoFlow implementuje pouze tři metriky, kterými jsou *Coupling*, *Cohesion* a *Coupling/Cohesion ratio*. Umí pracovat pouze se soubory ve formátu XML. Podrobnější specifikace formátu vstupních souborů není k dispozici. Do nástroje nelze importovat modely exportované z Enterprise Architect ani Modelio a nepodařilo se ani najít modelovací nástroj, ze kterého by bylo možné import provést.

ProM je nástroj, který je zaměřen hlavně na proces mining a výpočtům metrik kvality BPM se věnuje pouze okrajově. Jeho funkčnost je rozdělena do zásuvných modulů. Některé moduly se věnují výpočtu metrik kvality BPM. ProM umožňuje provést výpočet měr NOA, CFC a Density. Díky architektuře založené na zásuvných modulech umožňuje import souborů v několika formátech. Jako jediný z popsaných nástrojů má veřejně dostupné zdrojové kódy.

BPMN Measures sice implementuje 10 měr složitosti ale umožňuje načíst pouze jeden formát, a to XPDL ve verzích 2.0 a 2.2. Zdrojové kódy ani spustitelná forma programu nejsou veřejně dostupné na internetu.

BPMN Quality Tool implementuje metriky NOA, CFC, CP, Density a Coeficient of Network Complexity. Umožňuje pracovat se soubory ve formátu XMI. Zdrojové kódy ani spustitelná forma nástroje BPMN Quality Tool nejsou volně dostupné na internetu.

Výše uvedené nástroje umožňují automatizovat výpočty měr kvality procesních diagramů, ale pouze dva z nich jsou volně dostupné. Pokud se omezíme pouze na veřejně dostupné nástroje můžeme pracovat pouze s CoCoFlow a ProM. CoCoFlow umí vypočítat alespoň dvě metriky kvality procesů ale nebyl nalezen nástroj, který by umožňoval procesní modely exportovat tak, aby je bylo možné importovat do CoCoFlow. Tato skutečnost činí nástroj CoCoFlow prakticky nevyužitelným. ProM umožňuje provést výpočet tří me-

trik kvalit procesu a umí pracovat s více vstupními formáty, ale výpočet měr kvality procesů je pro něj pouze okrajová záležitost.

Klára Jelínková ve své diplomové práci[40] konstatuje, že nenalezla vhodný nástroj pro automatizaci výpočtu procesních měr, který by usnadnil její výzkum. V následujících částech práce budeme zabývat vývojem nástroje, který by měl odstranit nevýhody současných řešení:

- Některé ze současných nástrojů nejsou veřejně dostupné.
- Většina současných nástrojů podporuje pouze jeden vstupní formát.
- Nástroje často implementují pouze metriky snadné na manuální výpočet jako jsou NOA nebo CFC.

Základní obecné požadavky na nově vzniklý nástroj jsou:

- Nástroj bude umožňovat výpočet alespoň několika základních metrik.
- Nástroj bude umožňovat práci s více formáty procesních modelů.
- Nástroj bude snadno rozšiřitelný o další vstupní formát i o implementaci nové metriky.

Kapitola 4

Analýza a návrh nástroje pro měření kvality procesů

Předtím, než započneme tvořit jakékoli softwarové dílo, je důležité co nejpřesněji definovat účel nově vzniklého softwaru proto, aby po celou dobu vývoje bylo jasně definováno, *co* se má vyvinout. Tímto se budeme zabývat v kapitole analýza.

Pokud máme definovaný účel softwarového díla a víme co chceme vyvinout, můžeme se zabývat tím, *jak* toto dílo vyvinout. Tímto se budeme zabývat v kapitole návrh.

Nově vytvořený nástroj pro měření kvality business procesů bude pojmenován Q4BPM, což je akronym slov Quality for Business Process Models.

4.1 Analýza

V této části textu se budeme zabývat tím, jaký nástroj je potřeba vyvinout, aby měl co nejširší využití. Proto by tato kapitola měla odpovědět na následující otázky:

- Jaké jsou funkční a nefunkční požadavky pro nově vzniklý nástroj?
- Jaké metriky bude nově vzniklý nástroj implementovat?

4.1.1 Požadavky na aplikaci

Sběr požadavků na aplikaci je jedním z prvních kroků při vývoji nového softwarového díla. Hlavním jeho smyslem je získat přehled o tom, jakou funkčnost bude budoucí aplikace poskytovat. Důležitost sběru požadavků potvrzují i studie [42, 43], podle kterých jsou chyby při sběru požadavků častým faktorem selhání projektu. Požadavek je definován jako „specifikace toho, co má být implementováno“. Rozlišujeme dva typy požadavků: [12]

- **Funkční** určují, jaké chování bude zamýšlená aplikace nabízet
- **Nefunkční** specifikují vlastnosti nebo omezující podmínky aplikace

4.1.1.1 Funkční požadavky

Soupis funkčních požadavků pro nástroj Q4BPM je následující:

F01 Výpočet měr procesu
Nástroj bude umožňovat automatizovat výpočet měr kvalit procesu definovaného v souboru ve formátu BPMN, XMI, XPDL.

F02 Výpočet měr procesu včetně subprocesů
Nástroj bude umožňovat automatizovat výpočet měr kvalit procesu včetně jeho podprocesů.

F03 Rozpoznání vstupního formátu
Nástroj sám rozpozná formát vstupního souboru.

4.1.1.2 Nefunkční požadavky

Soupis nefunkčních požadavků pro nástroj Q4BPM je následující:

N01 Rozhraní aplikace
Nástroj bude poskytovat API takové, aby jej bylo snadné integrovat do dalších aplikací.

4.1.2 Podporované metriky

V rešeršní části práce je popsáno několik vybraných metrik kvalit procesních modelů a také kritéria, díky kterým tyto metriky získávají věrohodnost jak po formální tak po praktické stránce. Například Jorge Cardoso[1] zavádí pro metriky následující kritéria: jednoduchost, konzistentnost, automatizace, aditivnost, interoperabilita. Další kritéria zavedl Latva-Koivisto[20] nebo Elaine Weyuker[22].

Metriky dle jejich zaměření můžeme rozdělit do pěti kategorií: size, complexity, modularity, coupling, cohesion. Cílem nástroje je pokrýt co největší počet metrik zaměřujících se na velikost modelu. Proto bude pro tuto kategorii implementováno nejvíce metrik. Q4BPM bude implementovat následující metriky:

- Size
  - Number of actions
  - Number of actions and control-flow elements
  - Number of activities split and joins

  - Number of end events
  - Number of exclusive gateways
  - Number of gateways
  - Number of inclusive gateways
  - Number of parallel gateways
  - Number of pools
  - Number of sequence flows
  - Number of start events
  - Number of swimlanes
- Complexity
  - Cognitive complexity
  - Control-flow complexity
- Modularity
  - Maximum nesting depth
  - Mean nesting depth
- Coupling
  - Weighted Coupling Metric

Q4BPM bude tedy implementovat celkem 17 metrik, a alespoň jednou metrikou bude pokrývat čtyři kategorie z pěti.

4.1.3 Podporované modelovací nástroje

Podle funkčního požadavku *F01* bude nástroj podporovat formáty BPMN, XMI a XPDL. Existuje řada modelovacích nástrojů, které umožňují modely procesů exportovat do těchto formátů. Bohužel pokud shodný proces vyexportujeme z několika modelovacích nástrojů, nebudou nikdy výsledné soubory shodné. Proto nelze zaručit 100% kompatibilitu nástroje Q4BPM s každým modelovacím nástrojem. Je potřeba určit, se kterými modelovacími nástroji bude Q4BPM kompatibilní. Při návrhu nástroje bude kladen důraz na to, aby byl nástroj co nejrobustnější, proto je možné, že nástroj bude podporovat i další modelovací nástroje, ale pro exporty z dále popsaných nástrojů budou vytvořeny automatizované testy, tak aby byla ověřena jejich kompatibilita.

Zvláštní pozornost se bude upínat k notaci BPMN, která je v poslední době nejpoužívanější.

- Podporované nástroje pro BPMN:
  - Enterprise Architect
  - bpmn.io
  - Bizagi Modeler
  - Bonita BPM
- Podporované nástroje pro XMI:
  - Enterprise Architect
  - Modelio
- Podporované nástroje pro XPDL:
  - Enterprise Architect
  - Bizagi Modeler

4.2 Návrh

V této části textu se budeme zabývat tím, jak bude nástroj Q4BPM navržen a jak bude vypadat jeho API. Správný návrh Q4BPM zajistí, že nástroj bude do budoucna snadno rozšiřitelný a bude snadné jej rozšířit o další metriky nebo vstupní formát.

- Jaké bude vypadat API Q4BPM?
- Jaká bude architektura nově vzniklého nástroje?

4.2.1 Q4BPM API

Návrh API nástroje Q4BPM je velice důležitý. API bude prostředníkem mezi Q4BPM a dalším softwarem, který jej bude využívat. API definuje jak mezi sebou budou softwarové komponenty spolupracovat.

API Q4BPM se bude skládat z několika rozhraní a tříd.

BPMMetric je rozhraní, které budou implementovat všechny metriky. Toto rozhraní definuje dvě metody:

- *double calculate(Process process)* vypočítá metriku pro daný proces
- *String name()* vrací jméno metriky

MetricResult je třída reprezentující výsledek výpočtu metriky. Tato třída má dva atributy:

- *double result* výsledek výpočtu metriky
- *String metricName* jméno metriky

ProcessParser je rozhraní, které implementují všechny parsery vstupních souborů. Metody rozhraní Q4BPM, které využívají toto rozhraní nabízí uživateli možnost vybrat si implementace parseru a vyhnout se tak automatickému rozpoznání formátu. Rozhraní obsahuje několik metod, které budou popsány v další části textu.

Q4BPM Rozhraní Q4BPM je hlavní částí celého API. Definuje veškerou funkčnost, kterou nástroj Q4BPM nabízí. Rozhraní se skládá z následujících metod:

- *Set<BPMMetric> availableMetrics()* metoda vracející implementace všech metrik, které jsou implementovány
- *List<MetricResult> calculateAllMetric(File file)* metoda, která vypočítá všechny dostupné metriky pro daný soubor. Využívá automatické rozpoznání formátu a výběr parseru. Rozhraní nabízí tuto metodu i pro argument typu *InputStream*.
- *List<MetricResult> calculateAllMetric(File file, ProcessParser parser)* metoda, která vypočítá všechny dostupné metriky pro daný soubor. Implementace parseru je vybrána uživatelem. Podobně jako v předchozím případě rozhraní nabízí tuto metodu i pro argumenty *InputStream, ProcessParser*.
- *MetricResult calculateMetric(BPMMetric metric, File file)* metoda, která vypočítá jednu konkrétní metriku pro daný soubor. Nástroj automaticky rozpozná formát a vybere implementaci parseru. Rozhraní nabízí tuto metodu i s argumenty *BPMMetric, InputStream*.
- *MetricResult calculateMetric(BPMMetric metric, File file, ProcessParser parser)* metoda, která vypočítá jednu konkrétní metriku. Implementace parseru je definována uživatelem. API nabízí tuto metodu i s argumenty BPMMetric, InputStream, ProcessParser.
- *List<MetricResult> calculateWithSubprocesses(File file, ProcessParser parser, String rootFileName, String mappingFileName)* metoda, která umožňuje vypočítat všechny dostupné metriky pro proces i se subprocesy.
  - *File file* zip archív, který obsahuje model hlavního procesu i modely jeho podprocesů.
  - *ProcessParser* implemetace parseru, která bude parsovat modely procesu.
  - *String rootFileName* jméno procesu, který považujeme za kořenový.

– *String mappingFileName* jméno mapovacího souboru. Tento soubor mapuje jména podprocesů na jména souborů s modely podprocesů. Data jsou v souboru uložena ve formátu *klíč=hodnota*, kde klíč je jméno podprocesu a hodnota jméno souboru, ve kterém je uložen model tohoto podprocesu.

U souborů umístěných v zip archívu nezáleží na jejich umístění, ale pouze na jejich jméně.

Pro přehlednost následuje kompletní soupis metod a jejich argumentů rozhraní Q4BPM.

```
public interface Q4BPM {
Set<BPMMetric> availableMetrics();
List<MetricResult> calculateAllMetric(File file);
List<MetricResult> calculateAllMetric(InputStream stream);
List<MetricResult> calculateAllMetric(
       File file, ProcessParser parser);
List<MetricResult> calculateAllMetric(
       InputStream stream, ProcessParser parser);
MetricResult calculateMetric(BPMMetric metric, File file);
MetricResult calculateMetric(
       BPMMetric metric, InputStream stream);
MetricResult calculateMetric(
       BPMMetric metric, File file, ProcessParser parser);
MetricResult calculateMetric(
       BPMMetric metric, InputStream stream,
       ProcessParser parser);
List<MetricResult> calculateWithSubprocesses(
       File file, ProcessParser parser,
       String rootFileName, String mappingFileName);
List<MetricResult> calculateWithSubprocesses(
       File file, String rootFileName, String mappingFileName);
}
```

4.2.2 Architektura

Funkcionalita nástroje Q4BPM je rozdělena do čtyř částí.

Parser Třídy patřící do části parser zodpovídají za získání informací ze vstupních souborů. Tyto třídy implementují rozhraní ProcessParser. Každá z těchto tříd zodpovídá za parsovaní právě jednoho vstupního formátu. Konkrétní implementace rozhraní ProcessParser jsou BPMNProcessParser, XMIProcessParser, XPDLProcessParser. Rozhraní ProcessParser obsahuje následující metody:

- Set<StartEvent> parseStartEvents(XQueryProcessor xqp)
- Set<EndEvent> parseEndEvents(Processor xqp)
- Set<Task> parseTasks(XQueryProcessor xqp)
- Set<Flow> parseFlows(XQueryProcessor xqp)
- Set<Gateway> parseGateways(XQueryProcessor xqp)
- Set<Participant> parseParticipants(XQueryProcessor xqp)
- Set<Lane> parseLanes(XQueryProcessor xqp)
- Set<String> supportedNamespaces()
- InputStream preprocess(InputStream is)

Metody začínající prefixem *parse* vrací všechny výskyty elementů požadovaného typu. Metoda *supportedNamespaces* vrací všechny XML namespaces, pro které je ten to parser otestovaný automatizovanými testy. Metoda *preprocess* umožňuje parseru na inicializovat potřebné vnitřní proměnné nebo upravit vstup tak, aby jej bylo možné zpracovat.

Model Model jsou třídy, které definují interní reprezentaci procesu. Hlavním účelem těchto tříd je odstínění výpočtu metrik od formátu vstupního souboru. Model zachycuje celkem osm typů elementů.

Hlavní nejdůležitější třídou modelu je třída proces, která zastřešuje celý model. Atributy třídy proces jsou:

- *String name* - jméno procesu. Tento atribut je důležitý zejména pro mapování podprocesů.
- *List<Process> subprocesses* - seznam podprocesů
- Dalšími atributy jsou seznamy elementů obsažených v procesu. Tyto elementy jsou startEvents, endEvents, tasks, flows, gateways, participants, lanes.

Dalšími třídami modelu jsou třídy StartEvent a EndEvent, které reprezentují počáteční a koncové události. Tyto třídy mají jediný atribut typu String se jménem *id*.

Třída Flow reprezentuje elementy sekvenčního toku v modelech procesů. Flow má tři atributy *String id, String source, String target*. Atributy source a target obsahují identifikátory počátečního respektive cílového elementu sekvenčního toku.

Třída Gateway reprezentuje brány v procesech. Jejímy atributy jsou *String id, GatewayType type* a množiny výstupních a výstupních hran s názvy incomeFlowIds, outcomeFlowIds. GatewayType označuje typ brány a je to enumerace s hodnotami EXCLUSIVE, INCLUSIVE, PARALLEL.

Třídy Participant a Lane reprezentují bazény a plavecké dráhy, které přiřazují zodpovědnosti jednotlivým účastníkům procesu. Obě tyto třídy mají pouze jeden atribut a to *String id*.

Metriky V části metriky jsou třídy, ve kterých je implementován samotný výpočet metrik. Všechny tyto třídy implementují rozhraní BPMMetric.

```
public interface BPMMetric {
        public double calculate(Process process);
        public String name();
}
```

Na ukázce kódu vidíme, že rozhraní BPMMetric je jednoduché a obsahuje pouze dvě metody. Metoda *calculate* vrací hodnotu metriky pro zadaný proces. Metoda *name* vrací jméno metriky.

4.2.3 Výpočet metrik pro procesy obsahující podprocesy

Nástroj Q4BPM bude umožňovat vypočítat metriky pro procesy včetně jejich podprocesů. Proto je ovšem nutné, aby měl nástroj k dispozici modely procesu i jeho podprocesů, jinak by nebylo možné výpočet realizovat. Modely všech procesů i podprocesů budou nástroji dodány ve formě zip archivu. Tento archiv musí obsahovat:

- Soubor s kořenovým procesem
- Soubory s modely podprocesů
- Mapovací soubor

U souborů obsažených v archivu záleží pouze na jejich jméně nikoli na jejich umístění v archivu. Jak již bylo popsáno dříve, mapovací soubor má formát *klíč=hodnota*. Na obrázcích 4.1 a 4.2 vidíme modely procesů určených pro ilustrativní vysvětlení výpočtu metrik pro procesy obsahující podprocesy. Předpokládejme, že proces z obrázku 4.1 je exportován do souboru *proces.bpmn* a proces z obrázku 4.2 do souboru *podproces.bpmn*. Pokud budeme chtít, aby ve výsledcích metrik byl zahrnut i podproces, musíme vytvořit zip archív s následujícím obsahem:

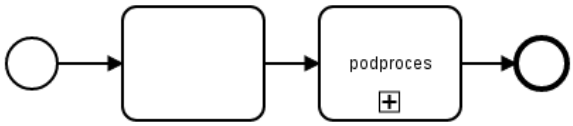
```
proces.zip
├── mappings.properties
├── podproces.bpmn
└── proces.bpmn
```

Protože chceme mapovat podproces s názvem *podproces* na soubor se jménem *podproces.bpmn*, soubor mappings. properties by měl mít tento obsah:

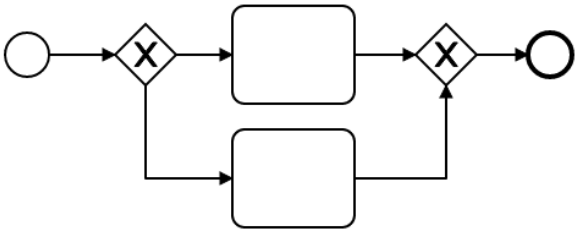
```
podproces=podproces.bpmn
```

Pro finální výpočet využijeme metodu *calculateWithSubprocesses* rozhraní Q4BPM s argumenty:

- file = "proces.zip"
- rootFileName = "proces.bpmn"
- mappingFile = "subproces.bpmn"



Obrázek 4.1: Proces pro ilustraci výpočtu metrik pro procesy s podprocesy



Obrázek 4.2: Podproces pro ilustraci výpočtu metrik pro procesy s podprocesy

Kapitola 5

Implementace nástroje pro měření kvality procesů

V předchozí kapitole jsme se zabývali analýzou a návrhem. Zjistili jsme, jaký nástroj budeme implementovat a máme i představu, jak ho budeme implementovat. V této kapitole se budeme zabývat samotnou implementací nástroje a problémy, které při implementaci nastaly.

5.1 Parsování elementů

Prvním problémem, který bylo potřeba vyřešit bylo parsování jednotlivý elementů procesu z formátů BPMN, XMI a XPDL. Jak již víme z předchozích kapitol, všechny tyto formáty jsou založeny na XML. Nabízí se tedy možnost využití jazyků pro vyhledávání elementů a atributů v XML dokumentech jako jsou XPath nebo XQuery. Protože platí, že XPath je pouze podmnožinou XQuery, budeme při implementaci využívat XQuery z důvodu komplexnější funkcionality. Dále bylo potřeba vybrat vhodný engine pro zpracování XQuery dotazů. Hlavními kandidáty byly BaseX XQuery Processor a Saxon konkrétně ve verzi home-edition. Pro implementaci byla zvolena knihovna Saxon hlavně pro její snadné použití a úzké zaměření pouze pro XSLT a XQuery.

Funkčnost knihovny Saxon je zapouzdřená ve třídě XQueryProcessor. Tato třída má jedinou veřejnou metodu *executeQuery*, která zajišťuje spušťení XQuery dotazu.

```
public XdmValue executeQuery(String xquery) {
  XQueryExecutable exec =
              saxon.newXQueryCompiler().compile(xquery);
  XQueryEvaluator query = exec.load();
  query.setContextItem(document);
  return query.evaluate();
}
```

Jazyk XQuery je založen na FLWOR výrazu [44]. FLWOR je zkratka slov for, let, where, order by, return.

- **for** vybere sekvenci uzlů
- **let** přiřadí sekvenci do proměnné
- **where** umožňuje filtraci uzlů
- **order by** seřadí uzly
- **return** definuje formát výsledku dotazu. Tato klauzule se vyhodnotí pro každý uzel právě jednou.

Všechny XQuery dotazy mají return klauzuli naprogramovanou tak, že jejich výsledkem je opět XML v podobě, kterou lze snadno mapovat na objekty programovacího jazyka Java. XQuery dotaz pro získání všech hran z procesu ve formátu BPMN vypadá takto:

```
query version "1.0";
for $element in
//*[(namespace-uri()='http://www.omg.org/spec/BPMN/20100524/MODEL')
       and local-name()='sequenceFlow']
where (//*:BPMNShape[data(@bpmnElement)=$element/data(@id)]
       or //*:BPMNEdge[data(@bpmnElement)=$element/data(@id)])
       and string-length($element/data(@sourceRef))>0
       and string-length($element/data(@targetRef))>0
return concat('<flow',' ','id=', '"',$element/data(@id), '"',' ',
       'source=', '"',$element/data(@sourceRef), '"',' ',
       'target=', '"',$element/data(@targetRef), '"',' ','/>')
```

Dotaz vybere všechny elementy se jménem *sequenceFlow*, které patří do specifikovaného namespacu. Where klauzule řeší odlišnosti mezi jednotlivými modeláři. Return klauzule formátuje výsledek do podoby vhodné pro mapování na objekty programovacího jazyka. Tyto odlišnosti budou podrobněji popsány v dalších částech textu. Výsledek tohoto XQuery dotazu může vypadat například takto:

```
<flow id="Id_4662d217-cf19-4139-b3af-685ef06850b7"
       source="Id_918cbceb-4592-4661-ac52-87dea86faa52"
       target="Id_23ecea1a-fc7d-492e-bc5a-3585684e4275"
/>
```

Protože Q4BPM obsahuje přibližně 30 poměrně komplexních XQuery dotazů, které ale řeší podobné problémy, byla vytvořena třída XQueryBuilder. Tato třída výrazně zjednodušuje vytváření dotazů určených pro tuto problematiku.

Na ukázce vidíme použití třídy XQueryBuilder pro vytvoření XQuery dotazu pro získání všech elementů EndEvent. Implementace metody pochází ze třídy *BPMNProcessParser*. Metoda nejprve vytvoří XQuery dotaz a následně volá privátní metodu *processQuery*, která zajistí zpracování dotazu včetně mapování výsledků na instance třídy EndEvent.

```
@Override
public Set<EndEvent> parseEndEvents(XQueryProcessor xqp) {
       String xquery = new XQueryBuilder()
       .addNamespace(supportedNamespaces())
       .setElementName("endEvent")
       .setOutputXmlElement("endEvent")
       .setWhereExpression(WHERE_EXPRESSION)
       .returnAttribute("id", "id").build();
       return processQuery(xquery, xqp, EndEvent.class);
}
```

5.2 Mapování XML na objekty

Elementy získané prostřednictvím XQuery dotazů je nutné mapovat na objekty. K tomutu účelu byla implementována třída XQueryResultMapper, která obsahuje generickou metodu *map*, která mapuje objekt třídy XdmValue reprezentující výsledek XQuery dotazu na objekty tříd patřících do modelu Q4BPM. Metoda *map* má dva argumenty. Prvním je *XdmValue result* a druhým je *Class<T> clazz*, kterým určujeme na objekty jaké třídy budeme výsledky mapovat. Mapování využívá standardní knihovnu programovacího jazyka Java pojmenovanou Java Architecture for XML Binding, zkráceně JAXB. Pro použití této knihovny je nutné anotovat třídy modelu anotací *@XmlRootElement* a jejich atributy anotací *@XmlAttribute*. Q4BPM dále využívá anotací *@XmlEnum* pro mapování výčtového typu a *@XmlList* pro mapování kolekcí.

5.3 Problémy exportovanými procesy

Pokud budeme z několika modelářů exportovat stejný proces do stejného formátu může se výstup lišit. Další odlišnosti nastávaly u chování jednotlivých modelářů. Z této skutečnosti pramenilo při implementaci několik problémů, které bylo třeba řešit. Mezi nejproblémovější modeláře patří Enterprise Architect. Nejproblémovějším formátem pro parsování bylo XMI, naopak XPDL a BPMN způsobavaly pouze minimální problémy.

5.3.1 Enterprise Architect

Při implementaci bylo zjištěno, že modelovací nástroj Enterprise Architect do exportu zahrnuje i elementy, které uživatel smazal. Pokud chce uživatel element odstranit i z exportu, musí tento element odstranit z přehledu elementů modelu procesu. Toto chování nástroje Enterprise Architect bylo vyhodnoceno jako matoucí pro uživatele, a proto do výpočtu metrik jsou zahrnuty pouze elementy, které jsou pro uživatele viditelné v modeláři. Zdrojem informací o viditelnosti jednotlivých elementů jsou části exportů, které jsou určeny pro zachycení grafické podoby procesu.

5.3.2 Odlišnosti XMI

Export do formátu XMI z modelářů Enterprice Architect a Modelio byl na tolik odlišný, že bylo nutné definovat rozhraní *XMIQueryFactory*, které obsahuje metody pro vytvoření XQuery dotazů pro parsování všech elementů a následně vytvořit zvláštní implementace pro oba modeláře. Rozdíly byly jak ve jménech některých elementů tak v tom, zda informace byly exportovány jako XML elementy nebo atributy.

Gateway exportovaná z nástroje Enterprice Architect:

```
<node xmi:type="uml:DecisionNode" xmi:id="EAID_E8DF798B">
  <incoming xmi:idref="EAID_F9D7FDF2"/>
  <incoming xmi:idref="EAID_DC3C4A3A"/>
  <outgoing xmi:idref="EAID_E5C9F967"/>
</node>
```

Gateway exportovaná z nástroje Modelio:

```
<node xmi:type="uml:DecisionNode"
   xmi:id="_dxC8xgs0Eee7r8OKvvGkXQ"
   outgoing="_dxC81As0Eee7r8OKvvGkXQ _dxC81gs0Eee7r8OKvvGkXQ"
   incoming="_dxC80As0Eee7r8OKvvGkXQ"
/>
```

Z ukázek exportů je vidět, že stejný typ brány exportovaný ze dvou různých modelářů má v jednom případě vstupní a výstupní hrany jako XML subelementy a ve druhém případě jsou hrany zachyceny jako atribut. V XQuery by bylo možné napsat dotaz, který by tento problém eliminoval, ale z důvodu udržení složitosti implementace dotazů v rozumných mezích bylo rozhodnuto pro odělení implementací.

5.3.3 Bizagi Modeler

Bizagi Modeler umožňuje exportovat elementy sequenceFlow i pokud nemají definovaný jejich počáteční element. Tyto elementy mohou způsobit chyby

při výpočtech některých metrik, jako je například NumberOfSequenceFlows. Proto jsou ty elementy filtrovány.

5.4 Načtení procesu

Přesto, že je již vyřešeno parsovaní elementů i jejich mapování na objekty programovacího jazyka, není inicializace objektů třídy Process zcela hotová. Pro načtení procesu slouží rozhraní *ProcessReader*, které má jedinou metodu *public Process readProcess()*. Objektový návrh načítání procesu byl inspirován návrhovým vzorem Builder. Metoda *readProcess* zajišťuje vše, co je potřeba vyřešit pro načtení modelu procesu ze souboru. Rozhraní *ProcessReader* má dvě implementace. První a jednodušší implementací je *SimpleProcessReader*. Tato implementace načítá procesy z jednotlivých souborů. Druhá implementace se nazývá *ZipProcessReader* a jejím úkolem je načítání procesů ze zip archívů.

Při načítání procesů se vyskytly další problémy způsobené rozdíly mezi formáty nebo modeláři. U formátů BPMN a XMI je informace o vstupních a výstupních hranách bran součástí elementu příslušné brány, ale u XPDL nikoli. Třída Gateway patřící do modelu Q4BPM má atributy, které reprezentují vstupní a výstupní hrany. Proto je nutné v metodě *unifyProcess* třídy *AbstractProcessReader* tyto atributy naplnit správnými daty. Další problém, který řeší metoda *unifyProcess*, je způsoben modelářem Bizagi Modeler. Bizagi exportuje i elementy sekvenčního toku, které nemají definován počáteční nebo cílový element. Tyto elementy jsou filtrovány již na úrovni XQuery dotazů. Problém nastává ve chvíli, kdy odfiltrujeme neplatný element sekvenčního toku, který byl vstupní nebo výstupní hranou některé brány. V tuto chvíli nastává situace, kdy graficky má brána o jednu vstupní nebo výstupní hranu méně, než v exportovaném souboru. Tato skutečnost má vliv na metriky ControlFlowComplexity nebo CognitiveComplexity. Proto je nutné upravit množiny vstupních a výstupních hran u všech bran tak, aby obsahovaly pouze hrany, které jsou zobrazeny v grafické podobě modelu.

Na ukázce můžeme vidět implementaci metody *readProcess* třídy *SimpleProcessReader*. Metoda využívá dva privátní objekty *parser* typu *ProcessParser* a *xqp* typu *XQueryProcessor*. Oba tyto objekty jsou inicializovány v konstruktoru třídy *SimpleProcessReader*. Metoda nejprve vytvoří novou instanci třídy Process a následně nastaví všechny její atributy. Před vrácením objektu process z metody je volána metoda *unifyProcess*, která vyřeší problémy popsané v přechozím odstavci.

```
@Override
public Process readProcess() {
  Process process = new Process();
  process.setTasks(parser.parseTasks(xqp));
  process.setFlows(parser.parseFlows(xqp));
```

```
  process.setEndEvents(parser.parseEndEvents(xqp));
  process.setStartEvents(parser.parseStartEvents(xqp));
  process.setGateways(parser.parseGateways(xqp));
  process.setLanes(parser.parseLanes(xqp));
  process.setParticipants(parser.parseParticipants(xqp));
  process.setSubprocesses(Collections.EMPTY_LIST);
  return unifyProcess(process);
}
```

5.5 Rozšířitelnost nástroje

Nástroj Q4BPM byl navržen tak, aby byl snadno rozšiřitelný. Za rozšiřitelnost nástroje se považuje implementace nové metriky nebo přidání dalšího vstupního formátu.

5.5.1 Implementace nové metriky

Pro implementaci nové metriky je nutné implementovat rozhraní BPMMetric, které bylo již popsáno v předchozích částech textu. Výpočet metriky se provádí v metodě calculate, která má jediný argument objekt třídy Process. Tento objekt obsahuje všechny důležité informace o procesu, které jsou nutné k výpočtu metriky. Nově vzniklou implementaci je nutné přidat do balíčku *cz.cvut.fit.brabepa1.q4bpm.metrics.impl* nebo jeho podbalíčků. Vše ostatní již vyřeší Q4BPM samostatně pomocí Java reflexí.

5.5.2 Přidání nového formátu

U přidání nového formátu je podobná situace jako u implementace nové metriky. Jediné co je potřeba, je implementovat rozhraní ProcessParser, které již bylo popsáno v předchozích částech textu. Nově vzniklou implementaci je nutné přidat do balíčku *cz.cvut.fit.brabepa1.q4bpm.parser* nebo jeho podbalíčků. Vše ostatní, včetně automatického rozpoznání formátu, již vyřeší Q4BPM samostatně pomocí Java reflexí.

Kapitola 6

Testování a dokumentace

Testování a dokumentace jsou důležitou částí vývoje každého netriviálního softwarového díla. Dokumentace může mít zásadní vliv na rychlost a kvalitu pochopení rozsáhlých zdrojových kódů. Úkolem testování je odhalit co nejvyšší počet chyb v softwaru.

6.1 Testování

Vývoj softwaru je činnost, při které často dochází k chybám. Chyby mohou vznikat ve všech fázích vývoje softwaru. Úkolem testování je tyto chyby odhalit a přispět tak ke zvýšení kvality softwaru. Chyby je nutné odhalit co nejdříve, protože s pokročilejší fází vývoje výrazně stoupá cena opravy chyby.

Pro vytvoření jednotkových testů nástroje Q4BPM byl použit framework JUnit. Celkem bylo vytvořeno více než 70 testů. Hlavním předmětem testování je parsování z různých formátů do společného modelu a výpočet metrik. Další část testů se soustředí na testování Q4BPM API. Tyto testy testují jak parsování modelů procesů tak výpočet metrik.

6.1.1 Testování vstupních formátů

Při testování vstupních formátů je potřeba prověřit velké množství asercí pro každý formát a každého modeláře. Proto, aby bylo možné testy vytvářet co nejefektivněji a bez duplicit testovacího kódu, vznikla pro každý formát abstraktní testovací třída, díky které je možné test implementovat pouze jednou, ale otestujeme všechny modeláře podporující daný formát. Aby tento postup fungoval je nutné proces, který chceme otestovat namodelovat v každém modeláři a dodržovat konvence pojmenování exportovaných modelů.

```
public abstract class AbstractXPDLProcessParserTest {
 @Test
 public void testBizagi() {
  doTest(TestFileFactory.getXPDLTestFile(
       TestFileFactory.BIZAGI, testName()));
 }
 @Test
 public void testEA() {
  doTest(TestFileFactory.getXPDLTestFile(
       TestFileFactory.EA, testName()));
 }
 protected abstract String testName();
 protected abstract void doTest(File file);
}
```

JUnit považuje za testy všechny metody anotované anotací *@Test*. Na ukázce to jsou metody testBizagi a testEA, které volají abstraktní metodu doTest s argumentem reprezentujícím vstupní soubor pro test. Pokud tyto soubory obsahují shodný proces, který je pouze namodelovaný v jiném modeláři, musí být výsledky všech asercí stejné a samotný test je tak možné implementovat pouze jednou. Samotné testy jsou implementovány v potomcích třídy AbstractXPDLProcessParserTest v metodě doTest. Potomci musí dále implementovat další abstraktní metodu testName, která vrací jméno testu, podle kterého je dohledán vstupní soubor pro test.

6.1.2 Testování API

Testy API jsou ze všech testů nejkomplexnější, protože testují veškerou funkcionalitu včetně parsování metrik. Pro každou netriviální metriku jsou vytvořeny ještě zvláštní testy. Testování API probíhá se dvěma odlišnými modely, které se snaží pokrýt co nejširší funkcionalitu Q4BPM. V tabulce 6.1 jsou vypočteny předpokládané výsledky metrik pro model z přílohy C.

6.2 Dokumentace

S rozsahem softwarového díla se význam dokumentace zvyšuje. Správně napsaná dokumentace může ušetřit programátorům mnoho hodin při hledání chyb nebo zavádění změn do softwaru. Dokumentace softwarového díla může mít mnoho forem jako například: analytickou dokumentaci, programátorskou dokumentaci, nebo uživatelskou příručku.

Analytickou dokumentací se částečně zabývá kapitola Analýza a návrh nástroje pro měření kvality procesů. Programátorskou dokumentaci lze automaticky generovat ze zdrojových kódů pomocí nástrojů jako *javadoc* nebo

Tabulka 6.1: Hodnoty metrik pro model procesu z přílohy C

Metrika	Hodnota
CognitiveComplexity	33
ControlFlowComplexity	6
MaxNestingDepth	3
MeanNestingDepth	1.65
NumberOfActions	8
NumberOfActionsAndControlFlowElements	11
NumberOfActivitiesSplitsAndJoins	14
NumberOfEndEvents	1
NumberOfExclusiveGateways	2
NumberOfGateways	6
NumberOfInclusiveGateways	2
NumberOfParallelGateways	2
NumberOfPools	1
NumberOfSequenceFlows	18
NumberOfStartEvents	1
NumberOfSwimlanes	2
WeightedCouplingMetric	0.22

Doxygen. Javadoc je určený pouze pro programovací jazyk Java. Doxygen lze použít pro programovací jazyky, jako jsou C++, C, Objective-C, C#, PHP, Java, Python. Oba tyto nástroje používají speciální dokumentační komentáře, ze kterých je následně dokumentace vygenerována. Programátorská dokumentace Q4BPM je generována nástrojem *javadoc* a je součástí přiloženého DVD.

6.2.1 Uživatelská příručka

Důležitou součástí dokumentace je také uživatelská příručka nebo návod jak software používat. U nástrojů jako je Q4BPM je vhodné mít v uživatelské příručce příklady, ze kterých se mohou uživatelé rychle seznámit s funkcionalitou nástroje.

Pro použití nástroje Q4BPM je důležité znát následující třídy:

- *Q4BPM* - rozhraní, přes které lze přistupovat k funkcionalitě nástroje
- *Q4BPMImpl* - implementace rozhraní *Q4BPMN*
- *MetricResult* - třída reprezentující výsledek metriky. *MetricResult* má dva atributy vypočtenou hodnotu a jméno metriky.
- *BPMMetric* - rozhraní, které implementují všechny metriky

- *ProcessParser* - rozhraní, které implementují parsery vstupních souborů

API Q4BPM nabízí metody, které umožňují vypočítat všechny dostupné metriky nebo pouze jednu konkrétní. Metody, jejichž argumenty neobsahují instanci ProcessParser rozpoznají formát vstupního souboru automaticky a zvolí příslušný parser. Nástroj Q4BPM umožňuje vypočítat metriky i pro procesy, které obsahují podprocesy. Do tohoto výpočtu je schopen zahrnout i modely podprocesů, pokud jsou všechny jejich modely součástí zip archivu, jehož struktura je popsána v kapitole Analýza a návrh nástroje pro měření kvality procesů.

Výpočet všech dostupných metrik s automatickým rozpoznáním formátu:

```
Q4BPM q4bpm = new Q4BPMImpl();
File file = new File("process.bpmn");
List<MetricResult> res = q4bpm.calculateAllMetric(file);
```

Výpočet všech dostupných metrik s ručním určením formátu souboru:

```
Q4BPM q4bpm = new Q4BPMImpl();
File file = new File("process.bpmn");
List<MetricResult> res = q4bpm.calculateAllMetric(file,
   new BPMNProcessParser());
```

Výpočet jedné metriky s automatickým rozpoznáním formátu:

```
Q4BPM q4bpm = new Q4BPMImpl();
File file = new File("process.bpmn");
MetricResult res = q4bpm.calculateMetric(new NumberOfActions(),
   file, new BPMNProcessParser());
```

Výpočet metrik pro procesy obsahující podprocesy, u kterých známe jejich modely.

```
Q4BPM q4bpm = new Q4BPMImpl();
File file = new File("process.zip");
List<MetricResult> res = q4bpm.calculateWithSubprocesses(file,
   "rootFile.bpmn", "mapping.properties");
```

Závěr

Hlavním cílem této práce bylo vytvoření nástroje pro výpočet metrik obchodních procesů, který by řešil nedostatky současných řešení. Jako nejzávažnější nedostatky současných řešení byly identifikovány problémy jako nedostatečný počet podporovaných vstupních formátů a nebo nízký počet podporovaných metrik.

Teoretická část čtenáře seznamuje s notacemi, ve kterých lze vytvářet modely obchodních procesů a s formáty, do kterých lze tyto modely exportovat. Dále jsou zde zavedena kritéria, které by metriky měly splňovat, abychom jejich výsledky mohli považovat za věrohodné. Popsána jsou jednoduchá kritéria zavedená Jorge Cardosem nebo Latva-Koivistem. Dále se práce částečně zabývá i kritérii navrženými Elaine Weyuker, které jsou známé pro svůj formalizmus. Výsledky metrik procesních modelů mají vyšší teoretickou i praktickou hodnotu díky tomu, že tato kritéria byla diskutována v seriozní literatuře. Poslední část teoretické části práce se zabývá samotnými metrikami, které rozděluje do pěti kategorií. Tato část textu také ukazuje na souvislosti mezi metrikami složitosti softwaru a procesních modelů.

Praktická část práce se zaměřuje na samotný vývoj nástroje pro automatizaci výpočtů metrik složitosti modelů obchodních procesů. Prvním krokem je analýza současných řešení. Výstupem tohoto kroku jsou základní požadavky na nově vzniklý nástroj tak, aby byl co nejpoužitelnější a řešil problémy současných implementací. Dalším krokem je analýza, ve které jsou stanoveny funkční a nefunkční požadavky na nástroj a dále vybrány metriky, které budou v nástroji implementovány. Výstupem kapitoly návrh je popis API Q4BPM, návrh vnitřního formátu BPM, ze kterého jsou vypočítávány hodnoty metrik. Předposlední kapitola popisuje implementaci nástroje. Protože formáty BPMN, XPDL i XMI jsou založeny na XML, tak Q4BPM pro jejich zpracování používá XQuery v kombinaci s JAXB, které mapuje výsledky XQuery dotazů na objekty programovacího jazyka Java. Toto řešení je velice robustní, jednoduché na implementaci a zároveň snadné na údržbu. Další části této kapitoly popisují hlavně problémy, které způsobily rozdíly mezi chováním jednotlivých

modelářů nebo rozdíly v jednotlivých formátech. Poslední kapitola práce se zaměřuje na testování a dokumentaci Q4BPM. Pro testování Q4BPM bylo navrženo přibližně 70 testů, které byly implementovány pomocí frameworku JUnit. Tyto testy pokrývají hlavně parsování vstupních souborů na společný model a výpočet metrik.

Hlavní cíl práce byl naplněn. Vznikl nástroj Q4BPM, který podporuje tři vstupní formáty a implementuje 17 metrik složitosti modelů obchodních procesů. Mezi podporované formáty patří BPMN, XMI, XPDL. V obou těchto směrech převyšuje všechna existující řešení.

Literatura

[1] Cardoso, J.: Business Process Control-Flow Complexity: Metric, Evaluation, and Validation. *International Journal of Web Services Research*, 2008, ISSN 15457362. Dostupné z: http://jorge-cardoso.github.io/publications/Papers/JA-2008-017-JWSR-Business-Process-Control-Flow-Complexity-proof.pdf

[2] Abramowicz, W.; Mayr, H. C.: *Technologies for business information systems*. Dordrecht: Springer, c2007, ISBN 1402056338.

[3] Gruhn, V.; Laue, R.: Approaches for business process model complexity metrics. In *Technologies for Business Information Systems*, 2007. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6354395

[4] Khlif, W.; Makni, L.; Zaaboub, N.; aj.: Quality Metrics for Business Process Modeling. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2009, ISBN 978-960-474-127-4. Dostupné z: https://www.researchgate.net/profile/Lobna_Makni/publication/228943162_Quality_metrics_for_business_process_modeling/links/0fcfd50c64383132e9000000/Quality-metrics-for-business-process-modeling.pdf

[5] Object Management Group: *Business Process Model And Notation [software]*. 2011, [cit. 2017-03-01]. Dostupné z: http://www.omg.org/spec/BPMN/

[6] Shapiro, R.: *BPMN 2.0 handbook*. Lighthouse Point, Fla: Future Strategies, druhé vydání, 2012, ISBN 9780984976409.

[7] Object Management Group: *Business Process Model And Notation - Referenční příručka*. 2011, [cit. 2017-03-01]. Dostupné z: http://www.omg.org/spec/BPMN/2.0/PDF/

[8] Object Management Group: *XML Metadata Interchange [software]*. 2015, [cit. 2017-03-08]. Dostupné z: http://www.omg.org/spec/XMI/

[9] Object Management Group: *MetaObject Facility [software]*. 2016, [cit. 2017-03-08]. Dostupné z: http://www.omg.org/spec/MOF/

[10] Object Management Group: *XML Metadata Interchange - Referenční příručka*. 2016, [cit. 2017-03-08]. Dostupné z: http://doc.omg.org/formal/2014-04-06.pdf

[11] Object Management Group: *Unified Modeling Language [software]*. 2015, [cit. 2017-03-09]. Dostupné z: http://www.omg.org/spec/UML/2.5/

[12] Arlow, J.; Neustadt, I.: *UML 2 a unifikovaný proces vývoje aplikací*. Brno: Computer Press, vyd. 2. vydání, 2007, ISBN 978-80-251-1503-9.

[13] What is UML. 2015. Dostupné z: http://www.uml.org/what-is-uml.htm

[14] Workflow Management Coalition: *XML Process Definition Language [software]*. 2016, [cit. 2017-03-08]. Dostupné z: http://www.xpdl.org

[15] Mendling, J.; Neumann, G.: A Comparison of XML Interchange Formats for Business Process Modelling. Dostupné z: http://nm.wu-wien.ac.at/research/publications/b455.pdf

[16] Workflow Management Coalition: *XPDL 2.2 Complete Specification [software]*. 2012, [cit. 2017-03-10]. Dostupné z: http://www.xpdl.org/documents.html

[17] Mach, R.: Návrh a tvorba nástroje pro optimalizaci procesů na základě analýzy BPM modelů. 2015.

[18] Gruhn, V.; Laue, R.: Complexity metrics for business process models. In *in: W. Abramowicz, H.C. Mayr (Eds.), 9th International Conference on Business Information Systems (BIS 2006), Lecture Notes in Informatics*, 2006. Dostupné z: https://www.researchgate.net/publication/221281564_Complexity_Metrics_for_business_Process_Models

[19] *IEEE standard computer dictionary*. New York, NY, USA: Institute of Electrical and Electronics Engineers, c1990, ISBN 1559370793.

[20] Latva-Koivisto, A.: Finding a complexity measure for business process models. Helsinki University of technology, 2001. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.2991&rep=rep1&type=pdf

[21] Cardoso, J.: Control-flow Complexity Measurement of Processes and Weyuker's Properties. In *6th International Conference on Enformatika*, Budapest, Hungary: International Academy of Sciences, 2005, ISBN 975-98458-7-3. Dostupné z: https://github.com/jorge-cardoso/jorge-cardoso.github.io/blob/master/publications/Papers/CP-2005-022-ICInformatika-Control-flow-complexity-measurement-of-processes-and-weyker-properties.pdf

[22] Weyuker, E. J.: Evaluating Software Complexity Measures. *IEEE Trans. Softw. Eng.*, Září 1988, ISSN 0098-5589. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6178

[23] Muketha, G.; Ghani, A.; Selamat, M.; aj.: A survey of business process complexity metrics. *Information Technology Journal*, 2010, ISSN 1812-5638. Dostupné z: http://scialert.net/qredirect.php?doi=itj.2010.1336.1344&linkid=pdf

[24] Cardoso, J.; Vanderfeesten, I.; Mendling, J.; aj.: Quality metrics for business process models. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.6133&rep=rep1&type=pdf

[25] Polančič, G.; Cegnar, B.: Complexity metrics for process models–A systematic literature review. *Computer Standards & Interfaces*, 2017. Dostupné z: http://www.sciencedirect.com.ezproxy.techlib.cz/science/article/pii/S0920548916302276

[26] Al-Hajjaji, M.; Alsmadi, I.; Samarah, S.: Evaluating Software Complexity Based on Decision Coverage. 2012. Dostupné z: https://goo.gl/tRXRmt

[27] Kaur, K.; Minhas, K.; Mehan, N.; aj.: Static and Dynamic Complexity Analysis of Software Metrics. 2009.

[28] McCabe, T. J.: A Complexity Measure. 1976.

[29] Vanderfeesten, I. T.; Cardoso, J.; Reijers, H. A.: A weighted coupling metric for business process models. In *CAiSE Forum*, ročník 247, 2007. Dostupné z: http://jorge-cardoso.github.io/publications/Papers/WP-2007-039-CAISE-FORUM-A-weighted-coupling-metric.pdf

[30] Cardoso, J.; Mendling, J.; Neumann, G.; aj.: A Discourse on Complexity of Process Models. In *BPI06 - Second International Workshop on Business Process Intelligence, In conjunction with BPM 2006*, Vienna, Austria: Springer-Verlag, Berlin, Heidelberg, 2006. Dostupné z: https://jorge-cardoso.github.io/publications/Papers/WP-2006-029-BPI-A-discourse-on-complexity-of-process-models.pdf

[31] Van der Aalst, W. M.: The application of Petri nets to workflow management. *Journal of circuits, systems, and computers*, 1998. Dostupné z: http://martinfowler.workflowpatterns.com/documentation/documents/vanderaalst98application.pdf

[32] Halstead, M. H.: *Elements of software science*. New York: Elsevier, první vydání, 1977.

[33] Shao, J.; Wang, Y.: A new measure of software complexity based on cognitive weights. *Canadian Journal of Electrical and Computer Engineering*, April 2003, ISSN 0840-8688, doi:10.1109/CJECE.2003.1532511. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.7800&rep=rep1&type=pdf

[34] Henry, S.; Kafura, D.: Software Structure Metrics Based on Information Flow. *IEEE Transactions on Software Engineering*, Sept 1981, ISSN 0098-5589, doi:10.1109/TSE.1981.231113. Dostupné z: http://ieeexplore.ieee.org/document/1702877/

[35] Piwowarski, P.: A Nesting Level Complexity Measure. *SIGPLAN Not.*, Září 1982, ISSN 0362-1340.

[36] Harrison, W. A.; Magel, K. I.: A Complexity Measure Based on Nesting Level. *SIGPLAN Not.*, Březen 1981, ISSN 0362-1340, doi:10.1145/947825.947829.

[37] Kluza, K.; Nalepa, G. J.: Proposal of square metrics for measuring Business Process Model complexity. In *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sept 2012, ISBN 9788360810514. Dostupné z: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6354395

[38] Mendling, J.: *Metrics for process models empirical foundations of verification, error prediction, and guidelines for correctness*. Berlin: Springer, 2008, ISBN 9783540892243. Dostupné z: http://www.springer.com/cn/book/9783540892236

[39] Vanderfeesten, I.; Reijers, H. A.; van der Aalst, W. M.: Evaluating workflow process designs using cohesion and coupling metrics. *Computers in Industry*, 2008, ISSN 0166-3615, doi:http://dx.doi.org/10.1016/j.compind.2007.12.007. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0166361507001807

[40] Jelínková, K.: Návrh měr kvality obchodních procesních modelů. 2017.

[41] Makni, L.; Khlif, W.; Haddar, N. Z.; aj.: A Tool for Evaluating the Quality of Business Process Models. 2010. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.407.5056

[42] Hofmann, H.; Lehner, F.: *Requirements Engineering as a Success Factor in Software Projects [online]*. 2001, [cit. 2017-04-05]. Dostupné z: http://www.ics.uci.edu/~wscacchi/Software-Process/Readings/Req-Engr-SuccessFactors-Software-July01.pdf

[43] Krigsman, M.: *CIO analysis: Why 37 percent of projects fail [online]*. 2011, [cit. 2017-04-05]. Dostupné z: http://www.zdnet.com/article/cio-analysis-why-37-percent-of-projects-fail/

[44] Holubová, I.: XQuery. Dostupné z: http://www.ksi.mff.cuni.cz/~svoboda/courses/2015-2-MIE-PDB/lectures/Lecture-06-XQuery.pdf

Příloha **A**

Seznam použitých zkratek

BPM Business Process Model

BPMN Business Process Model Notation

OOP Objektově orientované programování

Q4BPM Quality for Business Process Models

UML Unified Modeling Language

API Application Programming Interface

OMG Object Management Group

XML Extensible Markup Language

XMI XML Metadata Interchange

MOF MetaObject Facility

CASE Computer-aided software engineering

WfMC Workflow Management Coalition

XPDL XML Process Definition Language

IEEE Institute of Electrical and Electronics Engineers

LOC Lines of code

MCC McCabe cyclomatic complexity

NOA Number of activities

NOAC Number of activities and control-flow elements

NOAJS Number of activities splits and joins

CFC Control-flow complexity

IC Interface complexity

MaxND Maximum nesting depth

MeanND Mean nesting depth

JAXB Java Architecture for XML Binding

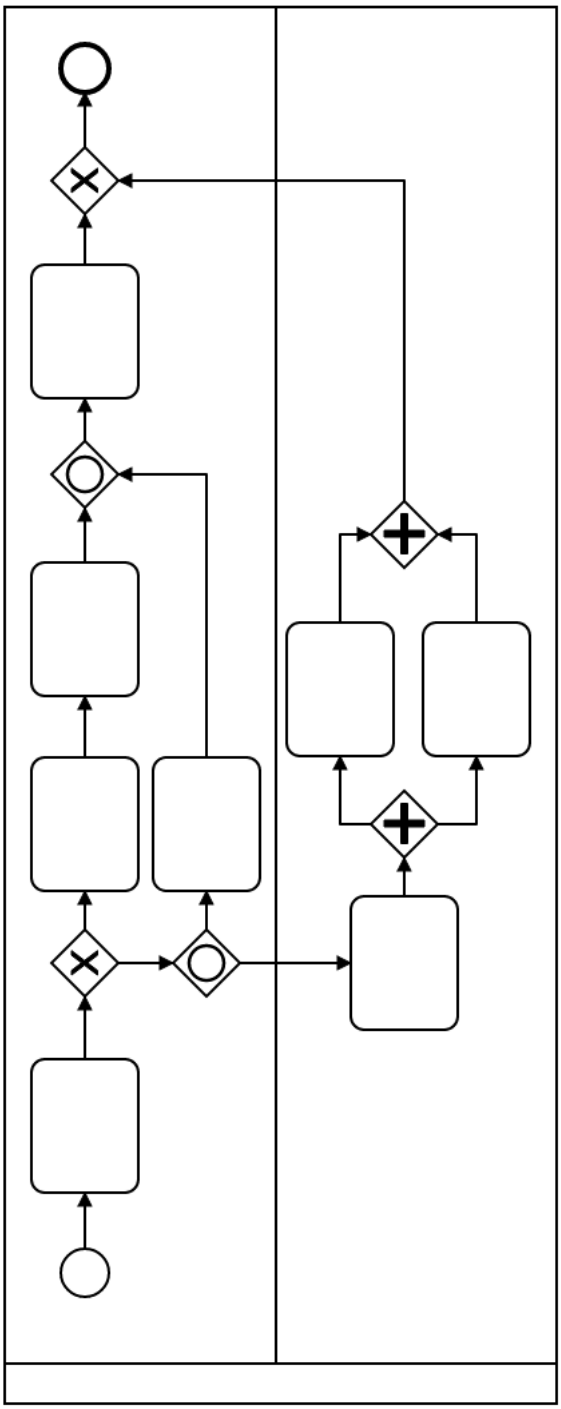
Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `bin`
  - `Q4BPM-1.0.jar`
- `javadoc` ..... dokumentace zdrojových kódů
- `Q4BPM` ..... zdrojové kódy Q4BPM
  - `readme.txt` ..... popis sestavení Q4BPM
- `thesis`
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojová forma práce ve formátu LaTeX

Příloha C

Model procesu pro testování Q4BPM API



Obrázek C.1: Model procesu pro testování Q4BPM API