VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta strojního inženýrství

Ústav automatizace a informatiky

Ing. Tomáš Pyrochta

NAVRHOVÁNÍ TESTOVACÍCH PROSTŘEDKŮ A METOD PRO APLIKACE INFORMAČNÍCH SYSTÉMŮ V PROSTŘEDÍ KLIENT/SERVER

DESIGN OF TESTING METHODS FOR INFORMATION SYSTEMS IN CLIENT/SERVER ENVIRONMENT

Zkrácená verze PhD Thesis

Obor:	26-15-9 Technická kybernetika
Školitel:	Doc. Ing. Branislav LACKO, CSc.
Oponenti:	Prof. Ing. Miloš Štěpánek, DrSc. Doc. Ing. Ladislav Buřita, Csc. Ing. Karel Bednář, Ph.D.
Datum obhajoby:	2. července 2002

KLÍČOVÁ SLOVA
Metriky, testování software, klient-server, informační systémy, Java, Navision, Axapta

KEYWORDS
Metrics, Software Testing, Client-Server, Information Systems, Java, Navision, Axapta

MÍSTO ULOŽENÍ PRÁCE
Ústav automatizace a informatiky,
Fakulta strojního inženýrství, VUT v Brně
Technická 2, Brno

ISBN 80-214-2231-9
ISSN 1213-4198

OBSAH

1 ÚVOD

Informace jsou dnes pro podnik pracujícím v tržním hospodářství jedním z rozhodujících faktorů vzrůstu a prosperity. Při snahách České republiky o proniknutí do evropských struktur je žádoucí, aby informační systémy pracující v českých firmách, které chtějí obstát ve světové konkurenci, byly jakostní a účinně podporovaly rozhodovací procesy. Informační systémy nejsou typickým výrobkem, tak jak je tradičně chápán. Ale jakost je atributem libovolného produktu a tedy i produktu z oblasti informačních technologií.

Jakost je tedy míra uspokojení stanovených, nebo daných potřeb uživatele při používání produktu za stanovených podmínek.[36]

Disertační práce se zaměřuje na v současné době aktuální oblasti testování složitých automatizovaných informačních a řídících systémů v prostředí klient/server, kde tento proces představuje problémy z několika hlavních důvodů:

- Informace představují jeden z hlavních faktorů pro rozvoj naší společnosti
- Pro zdárnou prosperitu podniku dnes představuje kvalitní informační systém rozhodující úlohu
- Navrhnout informační systém kvalitně je obtížné a představuje pro firmu velkou finanční a pracovní zátěž.
- Testování informačních systémů se stává čím dál složitější

Disertační práce se zabývá potřebou nové technologie automatizované podpory testování a jejím vlivem na systémy jakosti. Vzhledem ke složitosti a rychlosti vývoje současných aplikací jde ale odpovídající kvalitu zachovat obtížně především z následujících příčin:

- informační systém je složitě strukturován
- v průběhu vývoje se mění okolí systému – tím se mění i požadavky na systém
- na jeho návrhu spolupracuje řada pracovníků různých profesí
- současné systémy obsahují velké množství automatizovaných funkcí
- architektura klient/server přináší problém paralelního zpracování dat

V souvislosti s těmito charakteristikami dochází k růstu složitosti a důležitosti těchto systémů Proto dnes již nelze podchytit všechny stavy, události a jejich důsledky ani v rámci analýzy. Předtím bylo možné propracovanou analýzou, dobře zabezpečeným kódováním a inspekcí kódu přibližovat proces návrhu a tvorby SW ideálnímu stavu “produkce bez vad”.

Dochází tedy k posunu významu testování při zajištění jakosti v oblasti tvorby a údržby programových aplikací.

Až 50% nákladů vývoje současného programového vybavení připadá na testování.[42], 40% času vývoje velkých aplikací tvoří jejich validace.[41]

2 SOUČASNÝ STAV V RELEVANTNÍCH OBORECH K PŘEDMĚTU DISERTACE

Prostředkem k sjednocení hodnocení jakosti softwarových produktů je normalizace. První normou, která byla schválena je ISO/IEC 12119 Informační

technologie – Softwarové balíky – Požadavky na jakost a zkoušení. [36] Tato norma vymezuje pojem jakosti, přímo však neukládá dodavateli žádné povinnosti na dodržení určité její úrovně. Zavazuje jej pouze k tomu, aby poskytl potenciálnímu zákazníkovi všechny údaje předem. Norma obsahuje minimální povinnou osnovu tohoto dokumentu a pokyny pro to, jak lze splnění toho, co bylo v dokumentu deklarováno, ověřit zkouškou.

Norma ISO/IEC 9126 Informační technologie – Hodnocení softwarového produktu – Charakteristiky jakosti a jejich používání je platná již řadu let. Popisuje koncept hodnocení jakosti softwaru. V současné době bude postupně nahrazena soustavou norem a technických zpráv:

· Norma ISO/IEC 9126-1 Informační technologie – Jakost softwarového produktu – Model jakosti

· Technická zpráva ISO/IEC 9126-2 Informační technologie – Jakost softwarového produktu – Vnější metriky

· Technická zpráva ISO/IEC 9126-3 Informační technologie – Jakost softwarového produktu – Metriky pro jakost použití

· Technická zpráva ISO/IEC 9126-4 Informační technologie – Jakost softwarového produktu – Základní softwarové metriky

V naší práci se nezaměřujeme na Quality Assurance, ale pouze na určitou část testování SW (zjišťování úrovně jakosti). Proto si musíme uvědomit rozdíl mezi QA a testováním.

Jestliže vycházíme z definice podle ANSI/IEEE, pak zjednodušeně můžeme říct, že:

· Testování je kontrola kvality produktu. Testování se týká operací systému, nebo aplikace pod kontrolovanými podmínkami a vyhodnocuje výsledky.

· QA (QUALITY ASSURANCE) posuzuje kvalitu procesu použitého pro vytvoření kvalitního produktu. (Proto se softwaru pro QA týká úplný proces vývoje produktu, monitoring a zdokonalování procesu, zabezpečování vzniklých problémů, dodržení daných standardů a plánů atd.)[12]

Zaměřujeme se na testování produktu v rámci implementace. A tak by se mohlo zdát, že toto testování už nemá preventivní charakter, ale zvláště u velkých produktů má právě velký význam a vede ke značnému snížení nákladů, jestliže vady detekujeme a odstraníme před instalací a reálným provozem. Čím více chyb odstraníme v této fázi, tím se také sníží náklady na údržbu produktu, které dnes už představují velkou část z nákladů celého životního cyklu

Testováním téměř nikdy nelze dokázat korektnost produktu, neboť vždy může existovat ještě nějaký neotestovaný případ. Testováním lze odhalit přítomnost chyb, nelze dokázat jejich nepřítomnost. Proto za nejúspěšnější lze považovat ty, které odhalí nějakou chybu.

2.1 TESTOVÁNÍ C/S APLIKACÍ

2.1.1 Architektura klient/server

Prvotní myšlenkou systémů klient/server bylo to, že systém bude obsahovat centrální úložiště informací určitého typu dat, zpravidla databází. Jestliže je úložiště tedy umístěno centrálně, při jakýchkoliv změnách budou tyto změny předány všem spotřebitelům. Toto úložiště se nazývá server. Software na vzdáleném počítači komunikuje se serverem, zpřístupní informace, zpracuje je a nakonec zobrazí na vzdáleném počítači nazývaném klient.

Základní charakteristiky C/S systémů mohou být tedy tyto: prvních pět bodů může být považováno jako základní, povinné charakteristiky C/S systémů, zatímco dalších pět jsou nepovinné charakteristiky[5]:

- architektura klient/server se skládá z procesů klienta a serveru
- části klienta a serveru mohou pracovat na odlišných počítačových platformách
- buď platforma klienta, nebo platforma serveru mohou být zlepšeny, aniž by musela podstoupit upgrade druhá platforma
- server je schopen obsloužit více klientů současně; v určitém systému C/S, klient může přistupovat k více serverům
- systém klient/server obsahuje určitý typ síťového potenciálu (networking capability)
- významná část aplikace logicky spočívá na straně klienta
- akce je obvykle inicializována na straně klienta, ne na serveru
- uživatelsky přívětivé grafické prostředí GUI je obecně na straně klienta
- pro systémy klient/server je příznačný jazyk SQL
- databázový server umožňuje poskytovat chráněná a bezpečná data

První interaktivní IS byly realizovány na výkonném sálovém počítači se sítí jednoduchých terminálů schopných realizovat pouze znakovou komunikaci. S příchodem osobních počítačů se kromě využití grafiky (GUI) objevila tedy možnost využít na PC i část logiky. Při dělbě aplikace mezi serverem a klienty jsou možné následující varianty:

2.1.2 Testovací techniky

Základní princip testování je snažit najít co nejefektivnější způsob a dostatečnou sadu testů, které povedou k odhalení chyb. Následující čtyři testovací techniky jsou základní, které je možno použít a to jak pro hotový produkt, tak pro jeho jednotlivé části.[33]

- Testování podle struktury dat
- Inspekce produktu
- Statická analýza produktu
- Testování podle struktury programu

2.1.3 Přístupy k testování

Ruční Testování

Manuální provádění testování na multiuživatelském systému (client/server) představuje provádění namátkových technik. Navzdory veškeré dostupné automatizační technologii, více než polovina vývojářů vytvářející aplikaci na bázi C/S stále používá manuální testování. Navíc ruční testování neposkytuje opakovatelnost testů, není možnost analýzy testovacích dat, nevyhovuje požadavkům podle norem ISO 9000. To znamená, že vzrůstající složitost aplikací má za následek exp. růst času stráveného testováním (časté změny , které se musí otestovat). Proto se zavádí technologie ASQ (Automated software quality) – nástroje automatizovaného zabezpečení jakosti.[38]

Automatizované testování pomocí ASQ

Nástroje ASQ automatizují klíčové části procesu zabezpečení kvality a zahrnují široké soubory testování, simulace a monitoringu. Zatímco společnosti stále spoléhají na ruční testování softwaru, studie z roku 1994 oznámila, že 77 procent společností zjistily, že potřebují automatizované testování.[38] Jak se organizace stále více zaměřují na C/S prostředí, tak se dá očekávat další růst testování nástroji ASQ.

2.2 SOFTWAROVÁ FYZIKA

2.2.1 Princip softwarových metrik

Softwarové metriky můžeme podle různých hledisek rozdělit na několik skupin. Nejzákladnější dělení je na metriky teoreticky podložené, které se pokoušejí modelovat základní vlastnosti programů a na empirické , které se snaží změřit programování jako proces. Téměř všechny studie metrik podložených teorií vycházejí z Halsteadovy teorie z roku 1977. Halstead použil při hodnocení programů teorii informace, kdy je program chápán jako zpráva sestavená z operátorů a operandů. Halstead považuje programování za nedeterministický proces, při kterém vybíráme operátory a operandy z předem daného seznamu. Předpokládá, že všechny operátory jsou stejně kvalitní a zajímavé a že tedy budou vybírány ze stejnou pravděpodobností. Mezi empirické metriky můžeme zařadit hodnoty typu počet příkazů, počet modulů, průměrný počet řádků zdrojového kódu na modul, počet volání procedur atd.

Uvedené hodnoty se používají pro hrubé srovnání různých implementací, případně pro posouzení a ocenění jednotlivých programátorů. Mezi jedny z nejznámějších a nejhlouběji propracovaných empirických metrik patří McCabova cyklomatická složitost.[28]. Pro posouzení kvality softwaru nestačí pouze jednoduchá metrika typu počet řádků zdrojového kódu, ale že je vhodnější vyjít ze struktury programu charakterizované grafem toku dat v programu. Jiné empirické metriky posuzují styl programování. Například Berry-Meekinsova metrika [34] se používá jako nástroj v některých CASE systémech. Metrika je tvořena součtem vážených kvalit programu. Každá kvalita má přidělené určité bodové rozpětí

a bodování je potom orientováno pozitivním způsobem. Bohužel pokusy o použití této metriky například pro identifikaci programů s větším výskytem chyb však dosud nebyly podpořeny statisticky významnými daty. Úroveň určitého interního atributu ovlivňuje hodnotu určitého externího měřítka. Existuje tedy jak externí úroveň, tak interní aspekt většiny charakteristik. Například bezporuchovost může být měřena externě sledováním počtu chyb v daném časovém intervalu při testování softwaru a interně kontrolou specifikace a zdrojového kódu. Interní atributy mohou tedy být indikátory budoucích externích atributů. Jeden interní atribut může ovlivňovat více jak jednu charakteristiku a také jedna charakteristika může být ovlivňována jedním, nebo více atributy.

Interní metriky

Interní metriky se používají během fází návrhu a programování softwaru, který je v podobě např. specifikace, nebo zdrojového kódu. Měří vnitřní vlastnosti produktu, včetně těch, které vyplývají ze simulovaného chování. Účelem těchto metrik je tedy zjistit externí kvalitu. Interní metriky umožňují testerovi hodnotit výslednou kvalitu ještě před vznikem konečného produktu. Interní metriky měří interní atributy a udávají externí atributy pomocí analýzy statistických vlastností meziproduktu, nebo konečného produktu. Při měření se využívá počet či četnost prvků, ze kterých se software skládá.

Externí metriky

Měřítka externích metrik softwarového produktu jsou odvozena z měřítek chování systému, kterého je produkt částí. Zjišťují se testováním, provozováním a sledováním konečného softwarového produktu. Vycházejí tedy z požadavků na používání v určitém prostředí, v našem případě v prostředí C/S.

Jakost je definována jako míra splnění požadavků uživatele. Například budeme-li instalovat informační systém na vlakovém nádraží, budeme klást důraz na bezporuchovost, důležitá bude použitelnost, tzn. Že uživatel zvládne obsluhu bez dalšího zaškolování. Naproti tomu u informačního systému elektrárny bude kladen ještě větší důraz na bezporuchovost a bezpečnost, požadavek přenositelnosti a použitelnosti bude menší. Při definování požadavků na jakost se tedy vytváří seznam podstatných charakteristik a subcharakteristik. Při definování šesti základních charakteristik byla snaha o to, aby se co nejméně překrývaly. Potom se určí příslušné externí metriky a přípustné rozsahy. Tímto kvantifikujeme kritéria jakosti. Dále se definují interní atributy jakosti softwaru, čímž se při vývoji produktu předpokládá dosažení požadovaných hodnot externích charakteristik jakosti. Pro měření těchto hodnot zavedeme interní metriky a jejich přípustné rozsahy.

Základní normou týkající se zabezpečování jakosti softwaru, která byla v roce 1991 původně vydána v ISO je ČSN ISO/IEC 9126. Norma definuje šest základních charakteristik: [36]

· funkčnost (functionality) – schopnost informačního systému či softwarového produktu obsahovat funkce, které zabezpečují předpokládané, nebo stanovené potřeby uživatele při používáni systému za stanovených podmínek

· bezporuchovost (reliability) – schopnost informačního systému či softwarového produktu zachovat specifikovanou úroveň výkonu při používání systému za stanovených podmínek

· použitelnost (usability) - schopnost informačního systému či softwarového produktu být srozumitelný, se snadno naučitelnou obsluhou a atraktivní při používání za stanovených podmínek

· účinnost (efficiency) - schopnost informačního systému či softwarového produktu poskytovat potřebný výkon vzhledem k množství použitých zdrojů, při používání za stanovených podmínek

· udržovatelnost (maintainability) - schopnost informačního systému či softwarového produktu být modifikován. Modifikace zahrnují opravy nedostatků, vylepšování, adaptaci vzhledem ke změnám prostředí, změnám požadavků a změnám funkční specifikace

· přenositelnost(portability) - schopnost informačního systému či softwarového produktu být přenesen z jednoho prostředí do druhého

U jakéhokoliv typu softwaru, obzvláště u aplikací C/S je problémem, že se množství vzájemně souvisejících proměnných (metrik) zvětšuje natolik, že mohou být obtížně realisticky modelovány. Na rozdíl od hardwaru neexistuje jednoduchý opakovatelný model, který by zahrnoval míru selhání určitého prvku. Z dostupné literatury je zřejmé, že hodnocení jakosti softwaru-testování je problematické a ve stadiu rozpracovanosti.

3 CÍLE DISERTAČNÍ PRÁCE

Cílem disertační práce je navrhnout základ nového principu testování jakosti pro aplikace IS v prostředí klient/server, který by umožňoval posuzovat jakost softwaru z jeho měření v průběhu kódování a těsně po něm. Posuzování jakosti v prvních fázích životního cyklu aplikace je několikanásobně snadnější a lacinější, než kolik nákladů a úsilí stojí její posouzení později.

Cíle práce proto lze rozdělit do několika dílčích úloh:

· Předložit přehled postupů používaných při zajišťování kvality softwaru.

· Vybrat vhodné metriky měření a zdůvodnit jejich výběr s ohledem na prostředí C/S

· Pomocí měření reálných systémů zjistit, zda existují závislosti mezi vybranými metrikami složitosti softwaru

· Navrhnout a interpretovat faktory ovlivňující kvalitu systému s ohledem na prostředí klient/server.

Existují modely, jak popsat jakost softwaru pomocí charakteristik jako jsou funkčnost, bezporuchovost, použitelnost, atd.[36], které z různých hledisek jakost postihují. Problémem je, že u jakéhokoliv typu softwaru, obzvláště u aplikací C/S je problémem, že množství vzájemně souvisejících proměnných se zvětšuje natolik, že mohou být obtížně realisticky modelovány. Na rozdíl od hardwaru neexistuje jednoduchý opakovatelný model, který by zahrnoval míru selhání určitého prvku.

Z dostupné literatury je zřejmé, že hodnocení jakosti softwaru-testování je problematické a ve stadiu rozpracovanosti.

Významným indikátorem jakosti informačních systémů je jejich složitost, která je chápána jako míra úsilí, kterou člověk potřebuje vynaložit k tomu, aby systém navrhl, vyvinul, implementoval a udržoval. Složitost významně ovlivňuje i řadu dalších hledisek, jako například náklady. Jestliže se chceme zaměřit na posuzování jakosti v prvních fázích životního cyklu programu musíme se uchýlit ke statické analýza kódu. Modely založené na využití času se nechovají vždy dobře při zkoumání běhu kódu v budoucnosti. Data o sledovaných chybách nemusí osahovat relevantní informace pro požadované předpovědi.

Také testování softwaru pomocí statistických předpovědí selhání založených na extrapolaci testovaných dat je obtížné. Dalším důvodem je i skutečnost, že pro splnění zadání existují samozřejmě při tvorbě softwaru různé cesty.

Proto bylo zvoleno určit pomocí přesně definovaného a zdůvodnitelného postupu měřitelné parametry, metriky složitosti, které by charakterizovaly produkty C/S a z nich pak, pokud to je možné, izolovat faktory, které jsou potřeba, aby vysvětlily a reprodukovaly pozorované vztahy mezi proměnnými co nejpřesněji. Tyto faktory, pokud by existovaly by byly základem nového principu v posuzování jakosti aplikací informačních systémů v prostředí klient/server.

4 ZVOLENÉ METODY ZPRACOVÁNÍ

4.1 METRIKY

Metriky pro měření zdrojového kódu programu se dají rozdělit podle různých hledisek do různých kategorií. Jedny z často používaných [39] jsou tyto tři, které reprezentují rozdílné pohledy na zdrojový kód:

4.1.1 Procedurální metriky

Za jednotku se v této skupině metrik považuje procedura, nebo podprogram. Toto je zřejmě nejnižší pohled na zdrojový kód, pokládá se důraz na množství softwaru, který byl napsán a na jeho uspořádání na úrovni podprogramu.

- LOC
- COM
- N1, N2 ,U1, U2

4.1.2 Strukturální metriky

Tato skupina je orientována na pohled na software jako na komplex vzájemně se ovlivňujících modulů. Zaměření této skupiny metrik je spíše než na moduly samotné na síť vztahů mezi těmito moduly. Tento pohled představuje vyšší úroveň pojetí softwarového systému než procedurální pohled.

- MVG – McCabeovo cyklomatické číslo [28]
- FI - Fan-in (Fan-in)
- FO - Fan-out (Fan-out)

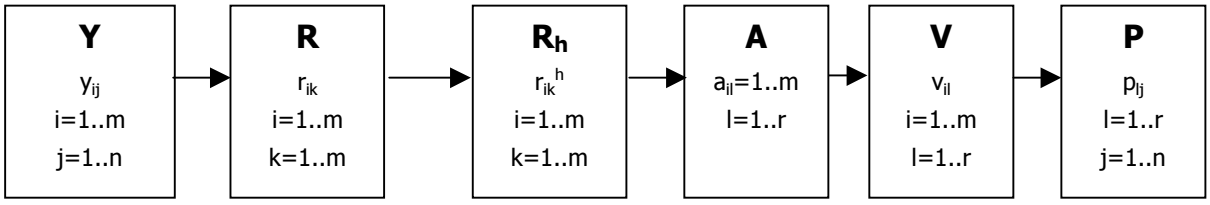
4.1.3 Objektově orientované metriky

Vzhledem k tomu, že architektura objektově orientovaných programů obsahuje nové prvky, není možno s úspěchem aplikovat všechny metriky navržené pro strukturální programy. Také proces návrhu OO programů je dosti odlišný (např. hranice mezi analýzou a návrhem nejsou tak striktně dané). Jedny z prvních OO metrik, které se staly de fakto standardem navrhli v roce 1991 Chidamber a Kemerer.

- NOC Počet dětí (Number of children)
- NOM – Počet metod ve třídě (Number Of Methods of per class)
- CBO – Spřažení mezi třídami (Coupling Between Object Classes)
- WMC – Váha metod pro třídu (Weighted Methods per Class)
- SP – Statický polymorfizmus (Static polymorphism)
- DP – Dynamický polymorfismus (Dynamic polymorphism)
- RFC – Odezva třídy (Response For a Class)
- LCOM - Nedostatečná soudržnost v metodách (Lack of Cohesion on Methods)
- DIT – Hloubka stromu dědičnosti (Depth of Inheritance Tree)

4.2 FAKTOROVÁ ANALÝZA

Při volbě faktorové analýzy jako statistické metody bylo vycházeno z toho, že při testování softwaru pomocí statistických předpovědí selhání založených na extrapolaci testovaných dat je obtížná. Modely založené na využití času se nechovají vždy dobře při zkoumání běhu kódu v budoucnosti. Data o sledovaných chybách nemusí osahovat relevantní informace pro požadované předpovědi. Proto byl zvolen postup určit měřitelné parametry – metriky, které by charakterizovaly produkty C/S a z nich pak izolovat faktory, které jsou potřeba, aby vysvětlily a reprodukovaly pozorované vztahy mezi proměnnými co nejpřesněji. Každá faktorová analýza vychází z matice dat Y. Z této vypočítáme korelační matici R. Do diagonály korelační matice dosadíme odhady komunalit a dostaneme Rh=(rhik). Tento problém komunalit pak spočívá v určení vhodných odhadových hodnot hi. Z matice Rh extrahujeme faktory a dostáváme jako výsledek matici A, tedy faktorovou matici. Sloupce této matice jsou navzájem ortogonální. Existuje mnoho rovnic, které jsou vhodné na reprodukci matice Rh. Proto musíme vybrat tu nejvhodnější, čímž vzniká problém rotace. Řešení tohoto problému vede k matici V. Posledním problémem je odhad faktorových skóre pro každý objekt, kde je výsledkem matice faktorových skóre P [10]:



Obr. 1 Schéma průběhu faktorové analýzy

4.3 ANALÝZA GQM

Při výběru metrik, kterých obecně existuje několik tisíc, je vhodné postupovat podle určité soustavy. V literatuře [4] je zmiňován tzv. GQM (Goal/Question/Metric) analýza, která nám umožní vybrat nejvhodnější metriky pro měření klient/serverových aplikací. Souvisí s modelem kvality softwarového produktu. Vycházíme z toho, že nevhodnější výchozí bod pro aplikaci metrik je analýza, která by měla obsahovat následující kroky:

- identifikace cílů
- položení otázek, které jsou v určitém vztahu ke každému atributu cíle
- volba a definice metrik, které jsou postupy, formulace, nebo

Výstupem z této analýzy je tedy tříúrovňová síť cílů, otázek a metrik. Síť obsahuje linky, které reprezentují vlastní výsledky analýzy, od vágního definování atributů cílů, odpovídajících otázek po identifikování použitelných metrik.

Na základě tohoto modelu kvality byly vybrány následující charakteristické atributy cílů:

- soběstačnost
- popisnost
- strukturovanost
- výstižnost
- čitelnost
- rozšiřitelnost

Model kvality podle Boema [4] byl vytvořen v ještě před tím, než se objektově orientovaná filozofie více rozšířila. V závislosti na tom, že vyšetřujeme C/S aplikace, používají OO techniky a nástroje jak v návrhu, tak ve vývoji jejich systémů. Bylo rozhodnuto přidat další atributy cílů nejnižší úrovně:

- abstraktnost
- znuvupoužitelnost
- sestavovatelnost

5 HLAVNÍ VÝSLEDKY PRÁCE

5.1 VÝBĚR DAT URČENÝCH K EXPERIMENTU A JEJICH ROZDĚLENÍ

Bylo vybráno několik reálných aplikací informačních systémů, které jsou, nebo budou v nejbližší době nasazeny ve firmách. Jako prostředí, které bylo používáno při experimentech autor zvolil systém Axapta 2.5 od společnosti Damgaard. Programovacím jazykem je X++. Tento jazyk využívá syntaxi jazyku Java (X++ je obchodní název společnosti Damgaard pro jazyk Java) a SQL, je zcela objektově orientovaný a je optimalizovaný pro psaní obchodních řešení. Bylo zajištěno to, že programy vytvářelo více programátorů, tím se odstranil individuální vliv každého programátora na výsledky měření. Byly měřeny klient/serverové systémy, které jsou navrženy na základě objektově orientovaného programování. Podle [22] je možno

měření těchto rozsáhlých systémů do tří úrovní a to na: systémovou, stromu dědičnosti, třídy.

Autor se ve své práci na nejnižší úroveň psaní kódu, na které je možno změřit všechny metriky vybrané podle metody GQM a která prezentuje měření na jednotlivých třídách jako základu celé hierarchie OO programu:

Poz.	Typ metriky	Tag	Popis
1	PROCEDURALNÍ	LOC	Řádky zdrojového kódu
2	PROCEDURALNÍ	COM	Řádky komentáře
3	PROCEDURALNÍ	L_C	Řádky kódu na řádek komentáře
4	STRUKTURÁLNÍ	MVG - v(G)	McCabeovo cyklomatické číslo
5	STRUKTURÁLNÍ	M_C	McCabeovo cyklomatické číslo na řádek komentáře
6	PROCEDURALNÍ	N_1	Halstead - celkový počet operátorů
7	PROCEDURALNÍ	N_2	Halstead - celkový počet operandů
8	PROCEDURALNÍ	U_1	Halstead - počet unikátních operátorů
9	PROCEDURALNÍ	U_2	Halstead - počet unikátních operandů
10	STRUKTURÁLNÍ	FI	Fan-in
11	STRUKTURÁLNÍ	FO	Fan-out
12	STRUKTURÁLNÍ	IF4	IF4
13	OBJEKTOVÁ	DIT	Hloubka stromu dědičnosti
14	OBJEKTOVÁ	NOC	Počet dětí
15	OBJEKTOVÁ	NOM	Počet metod ve třídě
16	OBJEKTOVÁ	WMC	Váha metod pro třídu
17	OBJEKTOVÁ	SP	Statický polymorfizmus
18	OBJEKTOVÁ	DP	Dynamický polymorfizmus
19	OBJEKTOVÁ	RFC	Odezva třídy
20	OBJEKTOVÁ	CBO	Spřažení mezi třídami
21	OBJEKTOVÁ	LCOM	Nedostatečná soudržnost tříd v metodách

Tabulka 1.: Seznam vybraných metrik

5.2 IMPLEMENTACE PROGRAMU

Pro zjištění základní matice dat, ze které se vycházelo v experimentu bylo nutno použít automatizované prostředky. Na trhu nebyl nalezen dostatečný programový prostředek, který by pokrýval celou škálu zvolených metrik. Z tohoto důvodu autor přistoupil k implementaci nové aplikace se snahou o co největší využití dostupných veřejných knihoven vhodných k použití. K tomuto účelu byla využita aplikace Jmetric (odkaz: www.csse.swin.edu.au/cotarr). Byl vyvíjen od roku 1998 na Swinburne University of Technology v Austrálii. Je k dispozici jeho zdrojový kód. Aplikace ClassMetric je implementována v jazyce Java. Program byl vyvíjen a provozován na operačním systému Windows 2000. Částečné testování probíhalo

také na operačním systému Linux (distribuce Red Hat). Volba jazyka byla dána těmito výhodami:

- Je objektově orientovaný
- Nezávislí na platformě
- Obsahuje grafiku nezávislou na zařízení zabudovanou přímo do jazyka
- Je zdarma
- Je relativně jednoduchý

Jako vývojové prostředí byl použit editor JCreator Pro 2.0 firmy Xinox. Byl využit vývojový balík Java Development Kit (JDK) verze 1.3.1. firmy Sun Microsystems. Program ClassMetric je určen pro zjišťování hodnot metrik na úrovni třídy. Při tvorbě algoritmů bylo využito zdrojového kódu knihoven aplikace Jmetric, zdrojový kód byl k dispozici. Funkčně se skládá ze čtyř modulů:

- Vstupní modul
- Parser (zdroj Jmetric)
- Metriky (zdroj masky Jmetric)
- Výstupní modul

5.3 POPIS EXPERIMENTU

Experiment umožnil analyzovat množinu programů v jazyku Java z komerčních aplikací v prostředí Axapta, což je C/S informační systém. Z této množiny programů byla naměřena matice základních dat. Prvním krokem tedy bylo určit dostatečně velkou množinu tříd, aby nasazení faktorové analýzy bylo statisticky odůvodnitelné. N Java programů, na kterých byla zjišťována M množina metrik, vytvořila matici N x M. Náhodně bylo tedy vybráno z každého systému několik tříd, kterých potom dohromady bylo 71. Na třídách bylo měřeno pomocí analýzy GQM 21 vybraných metrik. Pomocí programu ClassMetric byla určena matice 71 x 21 vstupních dat, neboli matici vstupních skór. S využitím programu Statgraphics Plus 5.0 jsem provedl předběžnou analýzu dat. Byla vypočtena korelační matice (její diagonální prvky jsou tedy rovny jedné).

Jak již byla řečeno, experiment zahrnuje analýzu dat metrik z množiny programů s využitím faktorové analýzy. Technika faktorové analýzy byla vybrána pro svou redukční schopnost. Hlavní využití technik faktorové analýzy je tedy redukovat počet proměnných a určit strukturu vztahů mezi proměnnými (metrikami). Na základě principů FA obdržíme z korelační matice množinu základních faktorů. Analýza hledá složku, která vyjadřuje v datech největší proměnlivost. Tato se pak stává prvním faktorem. Poté jsou určeny další faktory, kdy každý faktor reprezentuje jiný aspekt dat a je ortogonální k druhým.

5.3.1 Předběžná analýza dat

Předběžná analýza dat zjistila následující výsledky:

Tag	Průměr	S. odchylka	Minimum	Maximum
LOC	583,718	246,112	155,0	1000,0
COM	32,831	10,3302	10,0	52,0

L_C	17,6915	4,64468	5,3	27,8
MVG	57,0423	43,9566	1,0	170,0
M_C	1,92254	1,48797	0,0	6,3
N_1	28,7042	19,6312	7,0	77,0
N_2	80,507	30,7427	36,0	164,0
U_1	12,7746	7,21149	1,0	29,0
U_2	5,39437	3,57962	1,0	14,0
FI	3,97183	2,40221	0,0	9,0
FO	3,8169	2,46292	0,0	9,0
IF4	734,634	1067,84	0,0	3969,0
DIT	1,05634	1,21758	0,0	4,0
NOC	0,873239	1,22975	0,0	4,0
NOM	7,84507	5,91523	1,0	25,0
WMC	14,4085	9,56568	1,0	40,0
SP	6,16901	5,29956	0,0	30,0
DP	1,84507	3,55225	0,0	28,0
RFC	33,5493	19,3618	0,0	85,0
CBO	6,39437	4,21724	0,0	23,0
LCOM	12,3099	15,1286	0,0	78,0

Tabulka 2.: Statistické výsledky experimentu

Statistické výsledky ukazují, že procedurální metriky ukazují, že se jedná o rozsáhlejší třídy, které jsou ale podle L_C dostatečně okomentovány. Hodnoty MVG kolísají kolem hodnoty 50 což může znamenat, že by měl být program rozčleněn na menší a přehlednější celky, stukturovanost u některých tříd vykazuje velmi vysoké hodnoty. Podobně u Halsteadových metrik pro zjišťování počtu operátorů a operandů, kdy některé třídy mají hodnoty těchto metrik neúnosně velké. Větší počet metod ve třídě a jejich vyšší složitost má obvykle za následek větší aplikační „specifičnost“ třídy a tím menší pravděpodobnost, že mohou být dále děděny. To může být způsobeno například špatným návrhem třídy. Třídy, které mají hodnoty metriky NOM kolem 100 jsou toho důkazem. Podle [2] jsou ale naměřené hodnoty NOM odpovídající aplikacím velkých informačních systémů.

Vysoké čísla metriky fan-out značí kromě jiného vyšší spřažení mezi třídami, takže u tříd, kterým byla naměřena hodnota fan-out více než 5, spřažení již výrazně narušuje modularitu a ztěžuje jejich užití v jiné aplikaci. Na druhé straně vysoké hodnoty fan-in indikují, že objekty třídy jsou dobře navrženy.

Hloubka dědičnosti na měřených třídách, kterou indikuje metrika DIT je poněkud „mělká“, malé hodnoty znamenají menší pravděpodobnost existence chyby, protože čím je DIT větší, tím více metod může třída zdědit, tím obtížnější může být potom určení jejího chování.

Bohužel ale naproti tomu je vidět, že měřené třídy mají malý počet potomků (NOC), kdy čím vyšší hodnoty dosahuje tato metrika, tím je pravděpodobnost detekce chyb menší. DIT a NOC si navzájem konkurují, čím menší se snažíme mít

hodnotu DIT, tím vzrůstá hodnota NOC a naopak. Tyto výsledky korespondují s tím , co je obsaženo v práci [9]

Čím vyšší je CBO tím jsou třídy více náchylné k chybám a proto třídy s hodnotou metriky kolem 20 značí, že jsou špatně navrženy, protože vykazují malé hodnoty zapouzdření, což je cílem objektového návrhu. Podle [40] je metrika CBO více důležitá pro třídy UI (User Interface), než pro třídy DB (DataBase), což se zdá logické, protože u UI tříd jsou kladeny vyšší nároky na jejich opětovné použití v jiných aplikacích (musí mít CBO menší), než je tomu u tříd typu DB.

Byly naměřeny extrémě vysoké hodnoty metriky LCOM, což značí, že třídy mají velmi malou soudržnost. Třída s nízkou soudržností naznačuje, že je špatně navržena. Velké hodnoty RFC, kolem 100 značí, že u tříd je větší pravděpodobnost k nalezení chyb. Podobně třídy, které mají hodnoty metriky WMC větší, jsou více náchylné k chybám.

5.4 STATICKÁ ANALÝZA DAT

Většina postupů analýzy dat a z ní plynoucích závěrů jsou závislé na splnění základních předpokladů z nichž byly tyto postupy odvozeny. Jestliže nejsou tyto požadavky splněny, jsou další postupy jako je výpočet korelační matice a z ní vycházející výpočet faktorů snadno zpochybnitelné. Pro naši analýzu je hlavním předpokladem normalita. To znamená předpoklad, že data vycházejí z normálního rozdělení. Bohužel charakter měřených dat nedával ze své podstaty mnoho nadějí, že tento požadavek bude splněn. Dalším předpokladem je standardizace dat, protože měření metrik neprobíhalo na stejném rozsahu.

Statistické výsledky ukazují, že naměřená data vykazují vysoký stupeň nestability. Směrodatná odchylka v některých případech překročila hodnoty průměru, což znamená velký rozptyl naměřených hodnot. Pro ověření normality bylo použito testu třetího a čtvrtého centrálního momentu (šikmosti a špičatosti) a jeho porovnání z hodnotami pro normální rozdělení. U proměnných vykazujících nenormalitu byla zjištěna existence vybočujících měření, tzn. takových které se značně liší od ostatních hodnot. V jeho důsledku vycházejí vyšší hodnoty rozptylů, standardní odchylky atd. Bohužel charakter měření metrik na základě zdrojového kódu určuje systematicky sešikmené rozdělení, nebo rozdělení s vysokou špičatostí. V tomto případě tedy nemůžeme tyto hodnoty vypustit a násilně data „zesymetričtit“, protože bychom se dopustili přílišného zjednodušení. Připravili bychom se o důležité informace, které metriky v kterých třídách nabývají vysokých hodnot. Je také zřejmé, že použití aritmetického průměru jako metody odhadu střední hodnoty náhodné veličiny nedávalo odpovídající výsledky, protože udané metriky nejsou normálně rozděleny, ale aritmetický průměr je na něm založen.

Pro odstranění asymetrie dat zjištěných proměnných byla zvolena Box - Coxova metoda nelineární transformace dat. Jedním z cílů disertační práce dokázat, že existují závislosti mezi metrikami složitosti. To by znamenalo aplikovat faktorovou analýzu k redukci dat bez nároků na interpretovatelnost faktorů. Tento minimální cíl platí pro všechny analýzy hlavních komponent bez připojené rotace. Faktory jsou

potom matematickými konstrukcemi s určitými optimálními vlastnostmi, kterým ve skutečnosti nemusí nic odpovídat. Jestliže ale chceme také nalezené faktory nějakým způsobem interpretovat, tzn. odhadnout přímo neměřitelné veličiny, které by ovlivňovaly jakost softwaru je nutno jít o krok dále. Vycházíme tedy z předpokladu, že určitá veličina ovlivňující jakost softwaru a vycházející z metrik složitosti kódu klient/serverových aplikací je nepřímo měřitelná, ale že se musí projevovat ve vícerých proměnných, z kterými je korelativně spojená. Pak můžeme takovouto veličinu odhadnout pomocí klasické faktorové analýzy. Proto bylo zamítnuto použití metody hlavních komponent, kde jsou komunality rovny jedné, stejně jako diagonála korelační matice. Důvodem je také fakt, že bychom museli přijmout hypotézu, že celková variabilita proměnných je společná, tj. že je plně určená měřenými metrikami. Předpokládáme dále , že existují specifické faktory.

Bylo požadováno, aby faktory byly standardizované a navzájem nekorelovaly. Odhadem komunalit a extrakcí faktorů byla vypočtena výsledná matice A, (faktorové schéma) s jejími faktorovými saturacemi. Každý faktor je tedy charakterizovaný jedním sloupcem, každá proměnná jedním řádkem matice A. Při volbě typu rotace byly uvažovány rotace typu Quartimax a Varimax. Byla zamítnuta rotace typu Quartimax, která zjednodušuje faktorovou matici A maximalizováním rozptylu čtverců faktorových saturací v řádku. To nebere ohled na faktory a zjednodušuje popis pouze jedné dané proměnné (metriky). Varimax metoda definuje jednoduchost faktoru jako rozptyl čtverců jeho faktorových saturací. Při maximalizaci tohoto rozptylu se blíží faktorové saturace buď jedné, nebo nule a faktor je možno takto lépe interpretovat:

Metrika	Faktor 1	Faktor 2	Faktor 3	Faktor 4	Faktor 5	Komunalita
LOC	0,905986	0,094247	-0,03195	0,085411	0,085816	0,845373
COM	0,825842	0,021615	0,115036	-0,38053	0,055629	0,843616
L_C	0,500408	0,131286	-0,23138	0,469337	0,046476	0,543618
MVG	0,327114	-0,17091	0,219851	0,799395	0,021254	0,824031
M_C	-0,09182	-0,14044	0,18211	0,911454	-0,02359	0,892623
N1	0,879546	0,238364	0,151099	0,195588	0,024763	0,892118
N2	0,768583	-0,02977	-0,07091	0,163511	0,142739	0,643746
U1	0,881515	0,23774	0,00493	0,13432	-0,03622	0,852967
U2	0,868289	0,260449	-0,02325	0,0627	-0,0384	0,827706
FI	0,065575	0,084797	0,87382	0,098501	-0,04125	0,786457
FO	0,093803	0,014169	0,919592	0,004037	0,043035	0,856517
IF4	0,047402	0,04506	0,946701	0,027397	0,041557	0,902998
DIT	0,244955	0,814581	0,064528	-0,24046	-0,04336	0,78741
NOC	0,252537	0,53774	-0,25289	0,278124	0,398595	0,653123
NOM	-0,1005	-0,12065	0,259472	0,041173	-0,02431	0,094268
WMC	-0,10685	-0,33152	0,083331	-0,25929	-0,16749	0,223554
SP	0,460261	0,618115	-0,28798	0,054123	0,295757	0,767239
DP	0,239696	0,886766	-0,10359	-0,09804	0,067409	0,868694

RFC	-0,0946	0,763808	0,209042	-0,07671	-0,06861	0,646643
CBO	0,027224	-0,06488	-0,0284	-0,04602	0,869983	0,764745
LCOM	0,048406	0,171855	0,076321	0,05484	0,786682	0,659578

Tabulka 3.: Faktorové saturace po rotaci Varimax

Po rotaci odhadujeme faktorové skóre P pro každé měření. Jestliže je faktorová saturace značně vyšší, nebo menší jako nula, byla označena ve faktorovém schématu jako tučně.

Na měřených datech bylo zjištěno, že není možno nalézt nějaký všeobecný faktor, tedy faktor, kde by se všechny saturace proměnných značně lišili od nuly.

Bylo nalezeno 5 společných faktorů, tedy faktorů, které mají saturace od dvou až do sedmi proměnných společné. Na zvolených datech bylo zjištěno, že metriky vykazují vysokou saturaci vždy pouze v jednom faktoru. Komplexita proměnných je tedy jedna. Validita odhadu faktorových skóre závisí hlavně na velikosti korelace mezi proměnnými a faktory. Čím je tato korelace vyšší, tím je samozřejmě vyšší přesnost odhadu.

Komunalita je součet čtverců saturací společných faktorů. Reprezentuje tu část jednotkového rozptylu každé proměnné, která je společná se společnými faktory. Je tedy mírou determinovanosti ve vícenásobné regresi společných faktorů na danou proměnnou. Usuzujeme tedy, že počet faktorů pět je odpovídajícím kompromisem mezi přesností výpovědi faktorů a zjednodušením daného problému. Kromě metrik NOM a WMC mají všechny další metriky v celkovém jednotkovém rozptylu komunalitu zastoupenou více než v 50%. U těchto dvou metrik hraje významnější roli rozptyl jejich specifického faktoru.

Byla spočtena matice faktorových skóre, která obsahuje hodnoty daného měření pro každý faktor. V tomto případě byla vytvořena matice z 5-ti faktorů a 71 měření. Z této matice se vychází pro praktické použití celé faktorové analýzy. Tak je možno každý měřený program popsat několika základními veličinami a to právě lineární kombinací jeho faktorového skóre.

5.5 INTERPRETACE ZJIŠTĚNÝCH POZNATKŮ

Při výzkumu byly zjištěny následující poznatky:

· Výběr metrik je nutno provádět systematicky z ohledem na typ softwarové aplikace.

Původní předpoklad byl vybrat náhodně několik různých typů metrik a s těmito pracovat. Byla použita upravená analýza GQM (Goal/Question/Metric) [4], která nám umožnila zvolit nejvhodnější metriky pro měření klient/serverových aplikací podle těchto kroků:

· identifikace cílů (podle normy ČSN ISO/IEC 9126).
· položení otázek
· volba a definice metrik

· Experimenty byly potvrzeny poznatky, že měření rozsáhlých systému, které jsou objektově orientovány je nutno rozdělit do třech úrovní a to na: systémovou úroveň, na úroveň stromu dědičnosti a na úroveň třídy.[22]

Tradiční procedurální programy mohu být rozděleny na dvě odlišné úrovně: na systémovou (mezimodulární) úroveň a na úroveň modulární a na těchto dvou úrovních mohou být aplikovány statické metriky složitosti. Převzetím toho modelu na objektové aplikace docházelo ke špatné interpretaci měření.

· Porovnáním s dostupnou literaturou [2], [9], [40] byla kontrolována správnost aplikovaných algoritmů pro měření metrik.

Bohužel pro nedostatek srovnávacích dat nelze potvrdit, ani vyvrátit hypotézu, že hodnoty metrik změřených na aplikacích typu klient/server v programovacím jazyku Java odpovídají na určité hladině významnosti hodnotám naměřených na jiných typech softwaru, v jiném programovacím jazyku.

· Byly prokázány objektivní závislosti mezi vybranými metrikami složitosti softwaru.

Vzhledem k charakteru měřených dat byl předpoklad, že naměřená data budou vykazovat vysoký stupeň nestability. Bylo zjištěno, že směrodatná odchylka v některých případech překročila hodnoty aritmetického průměru, což znamená velký rozptyl naměřených hodnot. Pro ověření normality bylo použito testu třetího a čtvrtého centrálního momentu (šikmosti a špičatosti) a jeho porovnání z hodnotami pro normální rozdělení. Původní předpoklad nenormálního rozdělení byl potvrzen v několika případech. Proto pro odstranění asymetrie na zjištěných proměnných byla zvolena Box - Coxova metoda nelineární transformace dat.

Z vypočtené korelační matice bylo zjištěno, že „neobjektové“ charakteristiky korelují mezi sebou více, než objektové metriky.

· Byly navrženy a interpretovány faktory ovlivňující kvalitu

Bylo nalezeno 5 společných faktorů, tedy faktorů, které mají saturace od dvou až do sedmi proměnných společné. Na zvolených datech bylo zjištěno, že metriky vykazují vysokou saturaci vždy pouze v jednom faktoru.

Metrika	Objem	Abstrakce	Rozhraní	Řízení	Soudržnost
LOC	X				
COM	X				
L_C	X				
N1	X				
N2	X				
U1	X				
U2	X				
DIT		X			
NOC		X			
WMC		X			
SP		X			

DP		X			
RFC		X			
FI			X		
FO			X		
IF4			X		
NOM			X		
MVG				X	
M_C				X	
CBO					X
LCOM					X

Tabulka 4.: Interpretace faktorů

· Objem

Faktor významně souvisí s metrikami, které měří velikost zdrojového kódu, počítají operátory a operandy. Snaží se tedy zjistit určitou objemovou vlastnost programu.

· Abstrakce

Faktor je ovlivňován objektovými metrikami, které zjišťují úroveň modularity struktury programu, interpretuje hloubku stromu dědičnosti, počty volaných metod, aplikační specifičnost programu. Snaží se určit míru abstrakce programu.

Čím vyšší bude úroveň abstrakce, tím bude program lépe objektově navržen, tím větší bude náročnost na jeho pochopení a na testování

· Rozhraní

Souvisí se skupinou metrik, která je orientována na pohled na software jako na komplex vzájemně se ovlivňujících modulů. Zaměřuje se na síť vzájemných vztahů mezi těmito moduly. Faktor je mírou informačního toku, který do části programového kódu vstupuje, nebo s ní vystupuje přes jeho rozhraní.

· Řízení

Faktor je spojen s metrikami, které zjišťují strukturovanost programu s ohledem na jeho rozčlenění do funkčních celků.

· Soudržnost

Faktor zjišťuje stupeň závislosti jednotlivých tříd, jejich vzájemnou soudržnost.

6 ZÁVĚRY

V České republice softwarové firmy věnují pozornost výhradně vlastní technologii. Na druhé straně se podceňuje problematika řízení postupu tvorby softwaru. V tržním prostředí je velmi důležité, aby navržený program byl jakostní z hlediska potřeb zákazníka. Je však ale neméně důležité, aby již před zahájením vývoje bylo možno odpovědět na takové otázky, jako je: kolik bude stát vývoj, jak dlouho bude trvat vývoj, kolik bude nutno nasadit různých pracovních profesí, jaká jsou předpokládaná rizika, které vlastnosti softwaru budou rozhodující pro jeho úspěšné nasazení v praxi. Najít odpovědi na tyto otázky není v oblasti SW

jednoduché. V teoretické části předkládané práce jsou rozebrány jednotlivé teoretické aspekty testování softwaru, tato část se věnuje popisem typických klient/serverových architektur, vyzdvihuje, dokazuje výhody a přínosy automatického testování pomocí ASQ oproti ručnímu testování. Také se zaměřuje na softwarovou fyziku, speciálně na softwarové metriky, které jsou účinným a v disertační práci použitým prostředkem k zjištění kvality systémů. Je prezentován problém, že software, obzvláště aplikace C/S mohou být obtížně realisticky modelovány. Na rozdíl od hardwaru neexistuje jednoduchý opakovatelný model, který by zahrnoval míru selhání určitého prvku.

Cílem praktické části bylo zvolit pomocí přesně definovaného a zdůvodnitelného postupu měřitelné parametry softwaru – metriky, které by charakterizovaly klient/serverové systémy. Pomocí statistických metod zjistit, zda existují významné závislosti mezi metrikami složitosti jakosti softwaru a jestli ano, izolovat faktory, které by vysvětlili a reprodukovali vztahy mezi pozorovanými proměnnými co nejpřesněji. Autor narazil na fakt, že neexistuje vhodný použitelný nástroj vhodný ke shromáždění potřebných dat. Proto byl vytvořen originální nástroj ClassMetric, který využívá knihovny parseru vyvíjeného na Swinburne University of Technology v Austrálii. Nástroj je schopen zpracovávat zdrojové kódy programů Java a ukládat hodnoty metrik, které byly zvoleny z ohledem na charakter klient/serverových aplikací pomocí analýzy GQM. Pro testování byly vybrány reálné aplikace poskytnuté softwarovou firmou. Každý algoritmus implementovaný v aplikaci ClassMetric byl prakticky ověřen. Obecně je při měření softwaru nastává problém, kdy je obtížné shromáždit dostatečně velký soubor zdrojových kódů programů. V objektově orientovaných aplikacích je vhodné měřit metriky zdrojového kódu na třech úrovních. Takže při orientaci na nejnižší úroveň a na fakt , že každý zkoumaný projekt byl týmová práce vždy několika autorů se podařilo se docílit toho, že velikost matice naměřených dat je statisticky významná. Při kvalifikovaném použití statistické analýzy bylo zjištěno, že měřená data vykazují charakteristické vlastnosti jako je velký rozptyl dat, jejich nenormální rozložení.

Velký počet softwarových metrik komplikuje záměr kterým je určit a posoudit daný softwarový program. To znamená, že není možné objektivně posoudit absolutní složitost dvou odlišných programů. Cílem práce bylo zjistit, které aspekty softwaru mohou být hodnoceny, které metriky efektivně měří softwarovou složitost a tím ovlivňují jakost celé aplikace informačního systému. Originálně byla použita faktorová analýza a navzdory faktu, že FA je lineární model bylo zjištěno, že významný počet softwarových metrik navržených mnoha autory měří stejné aspekty jakosti softwaru, což je přínosem do teorie softwarového inženýrství.

Na základě podrobné znalosti problematiky autor přistoupil k interpretaci zjištěných faktorů a zjistil, že pouze pět faktorů dostatečně charakterizuje objektově orientovaný softwarový program. Jsou to: objem, abstrakce, rozhraní, řízení, soudržnost. Autor zjistil, že objektové informační systémy jsou především charakterizovány objektovými faktory nazvanými abstrakce, rozhraní, řízení a soudržnost. Pouze faktor objem reprezentuje neobjektovou podstatu aplikace.

Doposud v softwarovém inženýrství byl kladen velký důraz právě na strukturální metriky. Vzhledem k tomu, že více než 90 procent informačních systémů je v současné době realizováno objektově orientovanými jazyky je toto mylné. Autor ukázal na možnost aplikace statistické analýzy v prostředí softwarové fyziky, kde tento postup vede k novým poznatkům.

Bohužel pro nedostatek srovnávacích dat nelze potvrdit, ani vyvrátit hypotézu, že hodnoty metrik změřených na aplikacích typu klient/server v programovacím jazyku Java odpovídají na určité hladině významnosti hodnotám naměřených na jiných typech softwaru, v jiném programovacím jazyku. Proto by bylo vhodné provést měření na zdrojovém kódu programů jiného jazyka. Identifikované faktory ovlivňující kvalitu jsou základem nového principu testování jakosti. Jestliže by byly známy hodnoty faktorových skóre pro referenční aplikace, tedy aplikace, které by byly prohlášeny za jakostní, mohlo by se pomocí těchto pěti hodnot usuzovat na její jakost. Na základě poznatků zjištěných v disertační práci by bylo také vhodné odvodit novou metodu odhadu pracnosti softwarového projektu.

V předkládané práci byly naplněny hlavní stanovené cíle. Práce představuje další krok ke zkvalitnění procesu zjišťování jakosti softwarových aplikací. Dosažené výsledky byly prezentovány na konferencích a na pracovních seminářích v zahraničí. Výsledky budou též v dohledné době prezentovány v potvrzeném příspěvku na konferenci Objekty 2002 pod názvem „Význam objektově orientovaných metrik v měření informačních systémů“.

7 SUMMARY

Software testing is one of important approaches to assure the reliability and quality of software. Factor analysis has been used to analyze industrial software measures and is useful for identify the important factors that influence variability in measures. A lot of metrics suitable for C/S programs have been studied extensively. A statistical procedure for validating these metrics is presented.

The benefits of client/server architecture bring increased risk and complexity. The Yankee Group and Cooper & Lybrand have said that almost 75 percent of client/server projects have quality problems and only one percent complete on schedule and within budget. Research in software engineering in last decade has shown the importance of software metrics and the value of the quality assurance policy for project.[11] This thesis describes a project that investigates the use of software metrics based on measurement of source code in testing of client/server application. No relevant set of software metrics has been clearly identified that allows the construction of mathematical models to assess the quality of these applications.

We study the existence of relationships among current software measures based on a set of software metrics. They are measured on a set of java programs from commercial Axapta applications. To make metrics useful in a practical industry setting, it is necessary to identify the significant quality attributes of the current project, which can then be used to derive or select useful metrics. This process is

formalized in the goal/question/metric paradigm, proposed by Basili and Rombach (1988). The premise of goal/question/metric (or GQM) is that the most appropriate framework for application of metrics is in the context of an analysis that contains the following steps:

· Identification of 'goals' which are abstract attributes that the person undertaking the analysis regards as desirable in a software process or product. Goals are usually abstract nouns, e.g. testability, structuredness, portability, maintainability.

· Posing of 'questions' which express, as an interrogative sentence in a human language, the dichotomy between the presence and the absence of the goal attribute. An example question would be 'Has the system been decomposed into understandable procedural modules?'.

· Defining of 'metrics' which are procedures, formulae or algorithms that may be presented as evidence in reply to the question. Example metrics might include mean number of non-comment lines of code per function, mean number of parameters per function, etc.

The outcome of the GQM analysis is a three-layer network of goals, questions and metrics. For the current project, the goals were selected based on a hierarchical model of software development quality proposed by Boehm[4]. With regard to C/S characteristic were selected six low-level quality attributes: self-containedness; self-descriptiveness; structuredness; conciseness; legibility; and augmentability. The quality model was originally developed in the period before the widespread acceptance of the object-oriented paradigm. It was decided to add low-level quality attribute goals to reflect the fact that the reference team was using object-oriented techniques and tools in both the design and implementation of its system. The added attributes were: abstractness, reusability and buildability. For each of the low-level quality attributes, one or more questions were posed which are intended to express the way the attribute in question is seen as contributing to the quality goals of the project. A number of metrics were chosen for implementation based on the goals and questions described. The metrics implemented were grouped into three categories. The categories chosen were intended to assist in presenting the metrics as illustrations of different coherent views of the project's source code:

· Procedural metrics
· Structural metrics
· Object-oriented metrics

The main applications of factor analytic techniques are to reduce the number of variables and to detect structure in the relationships between variables, that is to classify variables. Therefore, factor analysis is applied as a data reduction or structure detection method (the term factor analysis was first introduced by Thurstone, 1931).

Technique of factor analysis was selected for its reduction capability. Factor analysis looks for a component that explains the greatest variability in the data. This becomes the first factor. Additional factors are identified for the remaining

variability. Each factor represents a differential aspect of the data and is orthogonal to the other factors. Factor analysis is used for determine the existence of some pattern of relationship among software metrics. This pattern has been used to reduce the data to a smaller set of factors.

The experiment has been involved analyzing the set of java programs from commercial Axapta applications. On this set of java programs was derived raw matrix. N java programs for witch M metrics have been computed generated an N * M matrix. Starting with a complete raw data set, correlation matrix is extracted. The initial set of factor is obtained using principal component method of correlation matrix. This set of initial factor has been rotated to obtain simpler and more easily interpreted factors. To facilitate the interpretation of the results, the following information has been provided with each factor analysis performed:

- mean and standard deviation of each metric
- matrix of correlation coefficients among the metric
- matrix of the rotated factors
- eigenvalues of the rotated factors

The diverse of programmers and the number of programs that has been analyzed involve provide a good basis for guaranteeing that bad influence was minimized. Experiments and result of this project confirmed that many of metrics suitable for C/S applications that have been proposed by many software researchers measure the same aspect of software quality and determine the existence of relationships among measures of metrics. The following five factors has been extracted and identified:

- Volume
- Abstraction
- Interface
- Control
- Cohesion

Work presented here provides a different viewpoint that can be used to further the understanding of software testing process a software measures.

8 SEZNAM LITERATURY

[1] Anděl, J.: Matematická statistika, Praha, SNTL 1985

[2] Basili, V., Briand, L.: A validation of object oriented design metrics as quality indicators. Technical Report, Univ. of Maryland, Dep. of Computer Science, April 1995.

[3] Berson, A.: Client/Server Architecture. New York, McGraw-Hill, Inc., 1992

[4] Boehm, B.W.: Software Engineering Economics. Prentice Hall, Englewood Cliffs, New Jersey, 1981

[5] Bochenski, B.: Implementing Production-Quality Client/Server Systems. John Wiley & Sons,Inc., 1994

[6] Booch, G.: Object – oriented analysis and design. Redwood City CA, Benjamin Cummings 1994 (article from web site)

[7] Capers Jones, SOFTWARE PRODUCTIVITY RESEARCH, "Woburn´s SQA: SQUISHING SOFTWARE BUGS" , The Boston Sunday Globe, 7. květen 1995.

[8] Chandor,A.-Graham,J.-Williamson,R.: Praktická systémová analýza. Bratislava, ALFA 1974

[9] Chidamber, S. R., Kemerer C. F.: A Metrics Suite for Object Oriented Design. IEEE Trans. Software Eng. Vol 20, no. 6, pp 476-493, June 1994

[10] Hebák, P.: Vícerozměrné statistické metody s aplikacemi

[11] Hetzel, B.: The Complete Guide to Software Testing. QED Information Sciences, Inc., 1988

[12] Hower R.: Software QA and Testing FAQ.In: Software QA/Test Resource Center, 1998

[13] Horáček, V.: Informace pro jakost. In: Sborník z konference Quality Management. VUT FS, Brno 1994, strana 1-8

[14] Hůlová, M.-Jarošová,E.: Statistické metody v managementu kvality. Praha,VŠE 1996

[15] Koubek, M.: Příčiny zaostávání ČR v informatizaci. CHIP č.7/1994, str.10-12

[16] Kaner, C.: Negotiating Testing Resources: A Collaborative Approach. In: Proceedings of the 9th International Quality Week, Software Research, San Francisco, CA 1996

[17] Kaner, C.: Software Negligence & Testing Coverage. In: Proceedings of STAR 96. Software Quality Engineering, Jacksonville, FL, 1996

[18] Koubek,M.: Příčiny zaostávání ČR v informatizaci. CHIP, 1994, č.7, str. 10-12

[19] Lacko, B.: Standardizace software. In: Sborník konference Standardizace a jakost software, VUT Brno 1997, strana 41-47

[20] Lacko, B.: Analýza dynamiky rozvoje informačního systému. In: Sborník konference Systémová integrace 95, VŠE KIT Praha 1995

[21] Lacko, B.: Informační strategie firmy. In: Bulletin CS SSADN USERS GROUP, č.3/1993, LBMS Praha

[22] Lake, A., and C. Cook, A Software Complexity Metric for C++, Annual Oregon Workshop on Software Metrics, March 1992.

[23] Langefors, B.: Theoretical Analysis of Information System. Auerbach Publishers Ins., Philadelphia, 1973
[24] Lin, Jin-Cherng.-Lin, Szu-Wen.- Ian-Ho.: Estimated method for software testability measurement. In: Proceedings of the 1997 8th IEEE International Workshop on Software Technology and Engineering Practice, London UK, 14,7,1997
[25] Nilaš, P.: Použití nástroje ARIS-TOOLSET pro dokumentaci systému řízení jakosti.In: Sborník konference Quality management "96", VUT Brno 1996, strana 41-46
[26] Marick, B.: Classic Testing Mistakes. Testing Foundations 1997
[27] Marick, B.: The Test Manager at the Project Status Meeting. In: Proceedings of the 10th International Quality week. Software San Francisco, CA, 1997
[28] McCabe T.J.: A Complexity Measure. IEEE Trans. on Software Engineering, SE-2, no.4, str. 308-320, 1976
[29] Molnár, Z.: Moderní metody řízení informačních systémů.GRADA, Praha 1992
[30] Mykiska, A.: Spolehlivost v systémech jakosti. Praha, ČVUT 1996
[31] Noskievičivá, D.:Statistické metody v řízení jakosti. Ostrava, VŠB 1996
[32] Pettichord, B.: Success with test Automation. In: Proceedings of the 9th International Quality Week, Software Research, San Francisco, CA 1996
[33] Richta, K.- Sochor, J.: Softwarové inženýrství I. Praha, Vydavatelství ČVUT 1996
[34] Salamon, W. J., Walace. D. R.: Quality characteristic and metrics for reusable software, National Institute of Standards and Technology, NISTIR 5459, 1994
[35] Vaníček, J.: Měření a hodnocení jakosti software z hlediska mezinárodní normalizace. In: Sborník konference Standardizace a jakost software, VUT Brno 1997, strana 1-21
[36] Vaníček, J.: Měření a hodnocení jakosti informačních systému. Praha, ČZU 2000
[37] Űberla,K.: Faktorová analýza. Bratislava, Alfa 1974
[38] Zohar, G.: Automated Software Quality Solutions-Managing Risk in Client/Server Systems. Mercury Interactive Corp., Sunnyvale CA, 1995
[39] Zuse, H.: A Framework of Software Measurement, Walter de Gruyer, Berlin – New York, 1998
[40] A Quantitative Approach to Software Management and Engineering. UFRJ/COPPE, CAPES-Brasil, PowerPoint presentation
[41] News@merc-int.com, European Edition, Fall 1997/98
[42] Sytem Testing & Quality Assurance Techniques, Advanced Information Technologies, březen 1995.

9 SEZNAM PUBLIKACÍ AUTORA

[1] Pyrochta, T.: Využití ARIS-TOOLSET pro modelování diskrétních systémů. In: Sborník konference Process Control 1997, Tatranske Matliare, June 8-11 1997, strana

[2] Pyrochta, T.: Zabezpečení kvality systémů klient/server. In: Jakost a informační systém. Duben 1999, str. 49-53

[3] Pyrochta,T.: Projektové řízení na FS. In: Výuka projektového řízení na vysokých školách v České republice. Brno, duben 1998, str. 32-34

[4] Pyrochta, T.: Software metrics in Client/Server Systems. In: Intergraph workshop, Swindon UK 2000.

[5] Pyrochta T.: How we can measure software systems. In: Workshop Weight – Info Project IST-2000-26251, Athens, Greece December 2001.

[6] Pyrochta T.: Software metrics selected for Weight-Info. In: Workshop Weight –Info Project IST-2000-26251, Paris, France January 2002.

[7] Pyrochta,T.: The Basics of Factor Analysis of Source Code of Client/Server applications. In: Rip 2002. Červen 2002.

10 AUTOROVO CURRICULUM VITAE

Osobní data:

PŘÍJMENÍ, JMÉNO: TOMÁŠ PYROCHTA

POHLAVÍ: muž
DATUM NAROZENÍ: 23. ČERVNA 1973
MÍSTO NAROZENÍ: Znojmo, Česká republika
NÁRODNOST: česká
STAV: SVOBODNÝ

Praxe:

2002 do dneška IT INTEGRATORS Brno, CZ, London, UK

Developer

1999 – 2002 TG NUMIC CZ, a.s. Brno, CZ

Product manager

V současné době práce na evropském projektu Weight-Info, odpovědnost za aplikace Web Service Management.

Odpovědnost za implementace IS Intools na MS SQL, Watcomu a Oracle8, troubleshooting, monitoring…

Programátor, DBA

Vývoj Axapta aplikací na MS SQL serveru, administrace MS SQL serveru, vytváření Visual Basic aplikací, web a ASP aplikace, T-SQ skripy, Infomaker, Access dotazy a reporty.

1998 Asyc, s.r. o. Brno, CZ

Programátor

Práce na vizualizaci technologických procesů pro PLC (Control Pan
Quick Control-OS/2), Pascal aplikace, SQL skripty.

1996 NICOM a.s. Brno, CZ

Lektor systému Win95 a MS Office95

Vzdělání:

1996 – 1999 VUT Brno, CZ

Postgraduální studium

Navrhování testovacích prostředků a metod pro aplikace informačních systémů v prostředí klient/server.

1991– 1996 VUT Brno, CZ

Fakulta strojního inženýrství

Ústav automatizace a informatiky

Kurzy, školeni, znalosti

- Informační systémy, advanced course Intools Integraph, Swindon UK
- Letní škola jakosti TU Vídeň/VUT Brno
- MS SQL SERVER – Microsoft Certified Professional
- Microsoft navision Attain – cert. Solution Reseler
- Microsoft Navision Attain – cert. Application builder
- Microsoft Axapta – cert. Programmer, Navison Dánsko

Operační systémy: Windows 98, NT 4.0 & 2000 (Workstation/Server), základy Linuxu . Databáze: MS SQL Server 7 2000, Oracle 8, MS Access, Watcom, FoxPro, DB IV.

Vybrané aplikace: Sybase Infomaker, Intools, Axapta, Navision Attain,

Control Panel, Quick Control, Adobe Photoshop, MS Office

Programovací Jazyky: Transact-SQL, X++, Java, Visual Basic,VB.NET, HTML, DHTML, ASP, Pascal

Řidičský průkaz skupiny B