Softwarové metriky a jejich význam

Jitka Kreslíková

Motto: "Nemůžete řídit to, co nemůžete měřit"
Tom DeMarco

1. Úvod

Současné pojetí tvorby software, jako průmyslového produktu, vytváří nutnost jeho plánování, oceňování, normování a hodnocení kvality. Současně jsou získávány údaje o realizovaných programových produktech. Tyto údaje mohou být použity jako podklad pro analýzu zákonitostí realizace softwaru a pro testování hypotéz o těchto zákonitostech. Výsledkem analýzy údajů jsou empirické zákonitosti, které mohou být použity jako základ pro stanovení technicko-ekonomických podkladů pro řízení prací při tvorbě softwaru. Mohou být rovněž použity jako podklad pro hledání takových metod realizace softwarových produktů, které by přinesly podstatné snížení pracnosti vývoje a hlavně údržby software.

Velkým problémem dat o realizaci softwaru je velký rozptyl pozorovaných hodnot. Důvody rozptylu dat jsou následující:

- Produktivita práce programátorů (měřeno v jednotkách délky programu na normojednotku času) silně závisí na typu realizovaného softwaru.
- Všechny kvantitativní charakteristiky programů silně závisí na kvalitě programátorů:
- Mezi programátory jsou velké rozdíly v produktivitě práce.
- Výsledné programy mají poměr efektivnosti 1 : 10; kratší programy píší obvykle lepší programátoři a tyto programy bývají i rychlejší.
- Programátoři jsou schopni vědomě ovlivnit různé charakteristiky programů (délka programů, počet proměnných, rychlost práce programu), pokud mohou libovolně měnit ostatní vlastnosti programů.

Průměrné hodnoty různých charakteristik programů však bývají relativně stálé. Je např. známo, že u programů střední složitosti bývá produktivita asi 3000 řádků na programátora a rok. Při opakovaných realizacích podobných projektů bývá kvalita odhadu různých charakteristik programů poměrně vysoká. V nových oblastech, kde nejsou nalezeny optimální postupy řešení, však bývá odhad značně nejistý. Tento fakt je třeba mít na paměti při hodnocení kvality univerzálně platných metod odhadu.

Pro vysvětlení řady jevů v programování a k odhadu budoucího vývoje se odvozují statistické zákonitosti z empirických dat. Zákonitosti se považují za platné, pokud nejsou ve sporu s pozorovanými daty (a také ve sporu mezi sebou vzájemně) a pokud mají schopnost vysvětlovat další jevy.

2. Co jsou softwarové metriky?

Proč při řízení softwarových prací je důležitá metrika a měření?

Důvod je jednoduchý: značná část řídicích činností zahrnuje vyhodnocování procesu tvorby software (nebo jeho plánování), oceňování stavu a kvality výsledků dosažených fází a rozhodování o proveditelnosti a implementovatelnosti. Všechno toto nevyžaduje jen měření, je to také součástí plánování a/nebo řízení vývoje softwaru. Je velmi obtížné zhodnotit stav nebo kvalitu projektu softwarového díla a objektivně rozhodovat bez přesného a spolehlivého měření. Toto je základem spolehlivého plánování, odhadu výkonu a produktivity, řízení a ocenění výsledků.

Měření vyžaduje používání vhodných jednotek, škál a postupů, tedy používání vhodné **metriky**.

Pomocí metrik jsou měřeny kvantitativní atributy. V softwaru se termín metriky týká disciplíny sběru a analyzování softwarových dat, softwarových atributů i měřicích škál.

Atributy jsou významné vlastnosti softwarového produktu nebo procesu, které poskytují užitečné informace o stavu produktu nebo o postupech v projektu. V tomto kontextu je produktem míněna specifikace, návrh a kódování. Jedním z atributů produktu je "rozměr", který může být měřen v rozličných jednotkách, nejčastěji se používá počet řádků kódu (LOC).

Hodnoty atributů nejsou získávány jenom měřením. Některé hodnoty metrik je možno považovat za cílové, odvozené z požadavků zákazníka, a proto působí jako omezení projektu (např cena projektu). Jiné hodnoty jsou odhady budoucího stavu projektu nebo produktu založené na odhadovacích rovnicích nebo mínění expertů. Systémy metrik pracují se všemi uvedenými typy hodnot. Cíle definují základní projektové požadavky, predikce indikují dosažitelnost cílů, skutečné naměřené hodnoty mohou být srovnávány s cíly při zjišťování pokroku v projektu.

Skutečné hodnoty umožňují monitorovat fáze projektu tím, že jsou porovnávány s plánovanými hodnotami. Navíc poskytují informace potřebné k určování místních prediktivních modelů. Preferuje se používání skutečných hodnot z objektivních měření, avšak některé "skutečné" hodnoty jsou založeny na "odhadech", kde hodnota je spíše odhadnuta než měřena a ostatní skutečné hodnoty jsou založeny na subjektivních měřítcích (např. složitost se často měří pomocí škály: velmi složité, nadprůměrné, průměrné, pod průměrem, jednoduché). Měření založená na subjektivních měřítcích musí být uvažována s opatrností. Je extrémně obtížné zajistit, aby různí lidé používali subjektivní škály konzistentním a srovnatelným způsobem. Jsou však atributy pro které je obtížné stanovit objektivní metriky a musí být tedy použity metriky subjektivní.

V kontextu se softwarovými metrikami se často vyskytují dvě koncepce. První koncepcí jsou standardní hodnoty pro standardní úlohu (např. dvě chyby při 100 LOC během testování jednotky). Tomuto se někdy říká **normy**. Značný problém vzniká chceme-li použít normy pro jedno prostředí v jiném prostředí. **Všechny dostupné zkušenosti indikují, že normy jsou platné pouze v jednom prostředí.** Druhá koncepce se týká **komplexních metrik** u kterých se pomocí matematických formulí kombinuje řada jednoduchých (primitivních metrik). U většiny komplexních metrik je problémem to, že není vždy jasné, který atribut chtějí měřit.

3. Výběr vhodných atributů a metrik

Problém výběru vhodných atributů a metrik nelze řešit jednoznačně. Např. na univerzitě v Marylandu byla vyvinuta metoda GQM (Goul-Question-Metric), která vychází z cílů dané organizace. Předpokládá se rozložení každého cíle organizace do množiny kvantifikovatelných otázek specifikujících metriky, pomocí kterých by otázky mohly být zodpovězeny.

Např. u problému údržby softwaru byly identifikovány čtyři cíle:

1. Maximalizace uspokojení zákazníka.
2. Zlepšování procesu údržby a rozšiřování systému.
3. Zvýšení predikce údržby.
4. Minimalizace ceny údržby.

Např. 2. cíl kromě jiného navozuje otázku:

Které jsou klíčové indikátory "zdravého" procesu?

Tato otázka navozuje metriky:

- úroveň plnění plánu
- trendy výskytu závad
- vážné a kritické závady

První uvedená metrika by měla být dále dekomponována na:
- skutečné plnění plánu každého projektu
- cílový plán každého projektu

Dále vzniká např. problém co je to plnění plánu, atd. Proto přístup GQM nelze považovat za uzavřený.

Měření používaná pro řízení projektů a jeho kvality lze rozčlenit na měření produktů a měření procesů.

Metriky atributů produktu

Měření **velikosti** (size) jsou základem pro většinu modelů odhadujících cenu a jsou také používány pro monitorování a predikci úrovní kvality, založenou na počtu chyb.

Existuje řada alternativních velikostních metrik. Mnohé z nich závisí na úrovni abstrakce při popisu produktu. Nejčastěji používaná velikostní metrika je počet řádků kódu. Jejich alternativou jsou metriky měřící počty bytů, počty operátorů a operandů.

Strukturální metriky vyjadřují vazby mezi softwarovými složkami na úrovni kódu. Obyčejně popisují řídicí tok do a z programových fragmentů (např. McCabe). Na úrovni specifikace požadavků a návrhu mají strukturní metriky snahu měřit **vzájemné vazby** (coupling).

Velikostní a strukturální metriky jsou často nazývány metrikami **složitosti**.

Částí produktu je také dokumentace. Je proto účelné zabývat se zjišťováním užitečnosti a použitelnosti dokumentace a odpovídajícími metrikami.

Kvality finálních produktů se projevují zejména v jejich předpokládaném prostředí (environment). Zahrnují výkon, spolehlivost, použitelnost atd. Měly by být definovány kvantitativně, aby mohly být verifikovány při přejímání produktu nebo v počátečních fázích jeho činnosti.

Je řada metrik, které usilují o měření **stability** produktu. Rozmezí při kterém produkt jako celek a jeho složky jsou nestabilní lze odhadnout pomocí počtu chyb a požadavků na změny. Nacházení chyb a požadavky na změny indikují potenciální problémové oblasti produktu.

Metriky atributů procesu

Řízení projektů se obvykle uskutečňuje **přidělováním zdrojů**. Vyžaduje měření úsilí, spotřebovaného času, počtu pracovníků a ostatních faktorů jako např. ceny strojového času. Při získávání hodnoty úsilí a spotřebovaného času je problémem stanovit vhodnou úroveň detailu. Aby se monitorovaly projekty efektivně, mělo by být úsilí měřeno na úrovni úkolů každého pracovníka. Mělo by být zřetelně odlišeno úsilí při opravách chyb od normálního úsilí a testování. Testování musí být plánováno, prováděno a vyhodnocováno. Metriky musí pokrýt všechny tyto aktivity.

Metrika **rozsahu testu** indikuje jaká část testu byla realizována. Je odvozena z velikostních a strukturálních metrik. Indikuje např. procento řádků kódu nebo programových větví, které byly prověřeny.

Metrika **výsledku testů** obvykle vyjadřuje počty chyb. Chyby by měly být klasifikovány tak, aby se získalo maximum informace o stavu projektu. Informace o chybě by měla identifikovat, který proces chybu způsobil a který proces ji detekoval.

V procesu softwarové tvorby dochází k transformaci požadavků specifikací na program. **Velikost transformace** může být posuzována pomocí **expanzních poměrů** (např. poměru počtu stránek návrhu vůči jedné straně specifikací). Neobvyklý poměr pro dílčí složku nebo úlohu může indikovat potenciální problém (nekonzistence, nekompletnost).

4. Monitorování fází projektu

Dříve než je možné měřit proces tvorby softwaru je nutné ho definovat. Toto znamená, že všechny normální fáze tvorby, úkoly a aktivity musí být identifikovány společně s výstupy (produkty) každého úkolu. Je také důležité stanovit milníky, ve kterých budou k dispozici plánované produkty.

Kvantitativní projektové monitorování
Zahrnuje následující kroky:
1. Stanovení kvantitativních cílů:
   - pro všechny atributy/metriky
   - přijatelný rozsah hodnot
2. Kontrola skutečnosti vůči cílům
3. Odezvy na odchylky od cílů

Monitorováni projektu zahrnuje srovnávání skutečných hodnot s plánovanými cíli a reakce na významnější odchylky. Může být prováděno na vyšší úrovni jednotlivých fází nebo na úrovni individuálních úkolů.

Řízeni kvality
Zahrnuje identifikaci složek s neobvyklými hodnotami:
1. Definování přijatelných rozsahů pro hodnoty složek
2. Identifikování složek s hodnotami mimo rozsahy
3. Přepracování/úpravy v případě nutnosti

Podobně jako v jiných oblastech lze konstatovat, že kvalitativní kontroly až na úrovni finálních produktů nejsou nejefektivnější. Měření je třeba provádět tak, aby potenciální problémy jednotlivých složek byly odhaleny co nejdříve.

Velkým problémem softwarových měření je, že neexistují objektivní standardy pro stanovení tolerancí hodnot metrik. Získané hodnoty mohou signalizovat celou řadu příčin. Uvažujme např. modul s nízkým počtem chyb. Příčinami mohou být: vysoká kvalita modulu, jednoduchost modulu, opakované použití ověřeného kódu. Další příčinou může být neadekvátní testování. Správná interpretace hodnot získaných metrikou je do značné míry ovlivněna zkušeností manažera.

5. Metriky softwarových produktů

Měří rozsah a strukturu. Měření rozsahu je důležité pro modely odhadů, úsilí, časových plánů a počtu chyb. Strukturální metriky indikují strukturální složitost softwaru.

Metriky specifikací
Metoda funkčních bodů

Tuto metodu publikoval Albrecht [1]. Je založena na zjištění počtu a rozsahu funkcí, které má výsledný systém provádět. V úvahu jsou brány funkce, které mohou výrazně ovlivnit délku programu. Metoda funkčních bodů počítá funkce zabezpečované softwarem, výsledné hodnoty jsou korigovány koeficientem složitosti, obtížnosti. Na základě součtu vážených funkčních bodů (FP) je proveden vlastní odhad. Hodnoty jsou získány ve fázi návrhu zjištěním následujících údajů:
- počet vnějších vstupů, IN
- počet vnějších výstupů, OUT
- počet logických souborů, FILES

- počet souborů na rozhraních, INT
- počet vnějších dotazů, INQ

Zjištěný počet funkčních bodů FP je dále násoben faktorem PCA zohledňujícím *složitost* projektu (PC), která je dána součtem ohodnocení následujících faktorů:
- datové přenosy
- distribuované funkce
- výkonnost
- intenzita využití konfigurace
- rychlost transakce
- přímý vstup dat
- míra využití uživatelem
- přímé opravy dat
- složitost zpracování
- opětné využití výsledků
- snadná instalace
- snadné použití
- nasazení na více místech
- závazek dalších úprav

Hodnoty pro vyhodnocení PC:
0 - není, nemá vliv, 1 - nepodstatný vliv, 2 - mírný vliv, 3 - průměrný vliv, 4 - významný vliv, 5 - velmi silný vliv

$PCA = 0.65 + (0.01 \times PC)$

$FPA = FP \times PCA$

Výsledné funkční obodování FPA slouží pro odhad práce, času a velikosti. Albrecht hodnotil během pětiletého období 22 projektů a stanovil aproximační vztahy podle použitého programovacího jazyka:

Pro obecný model $LOC = 66 \times FP$

Vynaložená práce (WH) v pracovních hodinách:

$WH = 54 \times FP - 13.390$

Z definice elementů funkčních bodů vyplývá, že tato metoda je vhodná pro měření standardních informačních systémů a ne pro aplikace v reálném čase nebo vědecké aplikace. Výpočty metodou funkčních bodů jsou dosti složité. Každá dílčí počítaná vlastnost je vážena podle své složitosti. Metody pro odhad složitosti se liší pro každou vlastnost, atd.

Metriky návrhu

Většina úvah o metrikách návrhu vychází z diagramů modulové struktury. Některé metriky strukturálního návrhu se týkají návrhu jako celku (např. počet modulů návrhu, maximální hloubka úrovně volání v hierarchické struktuře). Nejčastěji jsou metriky návrhové struktury založeny na principu párování:
- počet modulů, které volají daný modul
- počet modulů, které daný modul volá

Někteří autoři rozšiřují uvedený princip takto:

Informační tok do modulu (IFI) = počet modulů volajících daný modul + počet datových struktur, ze kterých daný modul získává data

Informační tok z modulu (IFO) = počet volaných modulů + počet datových struktur, do kterých modul zapisuje data + počet výstupních parametrů modulového rozhraní

Metrika složitosti informačního toku (IFC)

$IFC = (IFI * IFO)^2$

Většina dřívějších prací v oblasti metrik návrhu se týkala konvenčního strukturovaného návrhu, současné se týkají objektově orientovaného návrhu.

Metriky kódu

Těmto metrikám se věnuje největší pozornost. Známé jsou metriky od Halsteada [7] a McCabe [5].

Halstead použil při hodnocení programů teorii informace. Program je chápán jako zpráva sestavená ze základních symbolů - operátorů a operandů. Další charakteristiky programu, jako je větvení, obtížnost řešeného problému atd. jsou ignorovány. Ze zdrojového textu programu jsou vypočteny čtyři parametry:

N_1 - celkový počet operátorů v modulu

N_2 - celkový počet operandů v modulu

n_1 - počet různých operátorů v modulu

n_2 - počet různých operandů v modulu

Délku programu (N) definuje jako součet počtu operátorů a operandů: $N = N_1 + N_2$

Obdobně definuje *velikost programového slovníku* (n) jako součet všech různých operátorů a operandů. $n = n_1 + n_2$

Objem programu (V) charakterizuje celkový počet rozhodnutí, které potřebujeme během psaní programu při volbě každého z použitých operátorů a operandů: $V = N \times \log_2 n$

Úroveň programu (LEV) je dána jako poměr objemu nejúspornější implementace v určitém programovacím jazyku k objemu implementovaného programu: $LEV = (2/n_1) \times (n_2/N_2)$

Duševní úsilí (E) je úsilí potřebné na vytvoření určitého programu s úrovní LEV: $E = V/LEV$

Obtížnost programu (D) je nepřímo úměrná úrovni LEV: $D = 1/LEV$

Metrika složitosti od McCabe je založena na teorii grafů.

S rozvojem CASE nástrojů lze předpokládat, že hodnoty atributů software budou stále více získávány automaticky. Hlavním zdrojem budou data flow diagramy, ER diagramy, diagramy tříd, apod.

6. Cenové modely

Cenové modely poskytují přímé odhady úsilí nebo doby trvání. Mají jeden hlavní vstup (obvykle rozměr produktu) a řadu dalších přizpůsobovacích faktorů, které ovlivňují produktivitu. Nejznámějším cenovým modelem je COCOMO.

Odhad COCOMO (Constructive Cost Model)

Boehm [2] publikoval empirický model pro určení ceny projektu na základě známého nebo dobře odhadnutého rozsahu programu *(Del)*. Je založen na souboru významných veličin, které ovlivňují cenu a dobu řešení projektu. Model vyhodnocuje cenu a dobu trvání vývojové fáze softwarového díla, tj. dobu od zadání požadavků do příchodu programu na trh. Má tři úrovně - základní, střední a rozšířený model. COCOMO nepokrývá uživatelské školení, instalaci konečného produktu ani konverzi na nový výrobek.

Odhad COCOMO vychází ze vztahu:

$Doba = 2{,}5 \times (Del)^d$,

$Prac = c \times (Del)^b \times P$

Prac je udávána v člověkoměsicich, *Del* v tisících řádků, *Doba* v měsicich. Hodnoty *b* a *d* závisí na typu softwaru:

	b	*d*	*c*
pro jednoduché systémy	1,05	0,38	2,4
pro středně složité	1,12	0,35	3,0
pro složité systémy	1,2	0,32	3,6

c je nutno zjišťovat statistickými metodami (analýzou regrese) s použitím dat z dříve realizovaných projektů.

Model bere v úvahu široké spektrum faktorů, které mohou mít vliv na cenu a dobu řešeni. Střední odhad COCOMO používá 15 atributů:

Atributy produktu

RELY	míra požadavků na spolehlivost
DATA	míra rozsahu datové základny
CPLX	složitost produktu
	- řízení programu
	- složitost výpočtů
	- řízení vstupů a výstupů
	- práce s daty

Atributy počítače

TIME	míra požadavků na využití času
STOR	míra využití paměti
VIRT	míra proměnlivosti OS počítače
TURN	odezva počítače při vývoji

Atributy týmu

ACAP	znalosti a zkušenosti
AEXP	míra zkušenosti programátorů s podobnými aplikacemi
PCAP	míra kvality programátorů
VEXP	znalost virtuálního počítače
LEXP	znalost programovacího jazyka

Atributy projekce

MODP	míra použití moderních programovacích metod
TOOL	míra použití moderních prostředků vývoje software
SCED	"ostrost" požadavků na dobu realizace

Věrohodnost cenových odhadů

Boehm ukázal, že střední model COCOMO je středně přesný. 68% odhadů se lišilo méně než o 20% od skutečné ceny a času. Doporučuje použit více různých modelů pro odhad. Pokud se získané odhady výrazně odlišují je nutné znovu pečlivě zvážit všechny faktory a provést nový odhad. Upřesněni je často nezbytné, neboť:

- Vstupní údaje samy byly pouze odhadnuty a jsou postupně zpřesňovány.
- Modely pouze aproximuji skutečnost, známe-li jejich nedostatky, můžeme je kompenzovat.
- Některé projekty jsou natolik unikátní, že pro ně neexistují vhodné modely.
- Projekty sami podléhají změnám - zkrácený čas, méně nebo více řešitelů, změny v požadavcích na již hotový projekt.
- Modely jsou kalibrovány podle projektů, které se odlišují svým charakterem od připravovaného projektu.

7. Metriky v OO prostředí

7.1. Projektové (plánovací) metriky

Na rozdíl od metrik pro navrhování (design), projektové metriky neměří kvalitu vyvíjeného software. Jsou více fuzzy ale důležitější z hlediska celkové perspektivy projektů. Mohou být použity např. při predikci potřeb pracovníků a určování pokroku v procesu kompletace vyvíjeného systému.

Proces vývoje

Proces vývoje softwaru lze charakterizovat jako iterativní. Nové poznatky odkryté jedním modelem se zpětně zapracovávají do ostatních modelů. Každá iterace má tři části: plánování, tvorbu a zhodnocení.

Rozměr aplikace

V této části jsou metriky pro měření množství práce na projektu.

Počet scénářů (scenario script)

Scénář je sekvence kroků, které udělá uživatel a systém, aby realizoval určitý úkol (úlohu). Každý krok zahrnuje iniciátora, akci a účastníka. Scénář se vytváří pro každou větší funkci z hlediska konečného uživatele.

Počet scénářů indikuje rozměr vyvíjené aplikace. Předpokládá se minimálně jeden scénář pro každý kontrahovaný subsystém

Počet klíčových tříd

Klíčové třídy jsou centrálními prvky v problémové doméně. Jsou to takové, bez kterých by vznikaly velké potíže při tvorbě a údržbě systému.

	NSS	NKC	NSC	NOS	PDC	CPD	NMI	NCC
Rozměr aplikace								
Počet scénářů (NSS)								
Počet klíčových tříd (NKC)			x		x			
Počet podpůrných tříd (NSC)		x						
Počet subsystémů (NOS)								
Počet pracovníků								
Průměrný počet člověkodnů na třídu (CPD)		x				x		
Prům. počet tříd na vývojáře (CPD)					x			
Plánování								
Počet větších iterací (NMI)								
Počet zkompletovaných kontraktů (NCC)								

Tab. 7.1. Projektové metriky a jejich vztahy

Počet klíčových tříd indikuje objem práce potřebné k vývoji aplikace. Podle literatury, zkušenosti z objektově orientovaných projektů ukazují, že (20 - 40) % tříd lze charakterizovat

jako klíčové. Méně než 20% klíčových tříd signalizuje nedostatečnou analýzu problémové domény.

Počet podpůrných tříd

Podpůrné třídy nejsou centrální v problémové doméně, realizují však základní služby nebo interface pro klíčové třídy.

Počet podpůrných tříd indikuje objem práce potřebné k vývoji aplikace. Objevují se v pozdější fázi vývoje procesu.

Jejich počet se mění mezi jedno až tří násobkem klíčových tříd. Je to zejména závislé na uživatelském rozhraní.

Průměrný počet podpůrných tříd na klíčovou třídu

Vztah mezi klíčovými a podpůrnými třídami není jednoduchý, je ovlivněn řadou faktorů včetně složitosti uživatelského rozhraní. Průměrný počet se týká celkového počtu tříd v projektu. Minimální počet podpůrných tříd odpovídá počtu klíčových tříd.

Počet subsystémů

Subsystém je kolekce tříd, které společně realizují skupinu funkcí pro konečného uživatele. Minimální počet subsystémů se odhaduje na tři.

Počet pracovníků

Průměrný počet člověko-dnů na třídu

Průměrné množství úsilí na jednu třídu je nejlepším indikátorem pro určení množství práce na novém projektu, pokud již byl odhadnut počet tříd.

Průměrný počet člověko-dnů je závislý na kategorii třídy. Zejména záleží na tom, zda jde o třídu uživatelského rozhraní (UI) nebo modelovou, o třídu abstraktní nebo konkrétní, o třídu klíčovou nebo podpůrnou. Záleží také na hierarchii dědičnosti, programovém prostředí, knihovně tříd a zkušenosti pracovníka.

U komerčně použitelných tříd se uvažuje 10 - 15 člověko-dnů na třídu včetně testování a dokumentace.

Průměrný počet tříd na vývojáře (v rámci jednoho projektu)

Průměrný počet tříd je užitečný pro odhad počtu vývojářů na projekt. Vývojáři se mohou současně zabývat i více třídami. V závěru práce na jednotlivých třídách je vhodná určitá doba na "dozrávání".

U kratších projektů (pod 18 měsíců) lze v průměru uvažovat 20 tříd na vývojáře, u dlouhodobých 40 tříd.

Plánování

Počet větších iterací

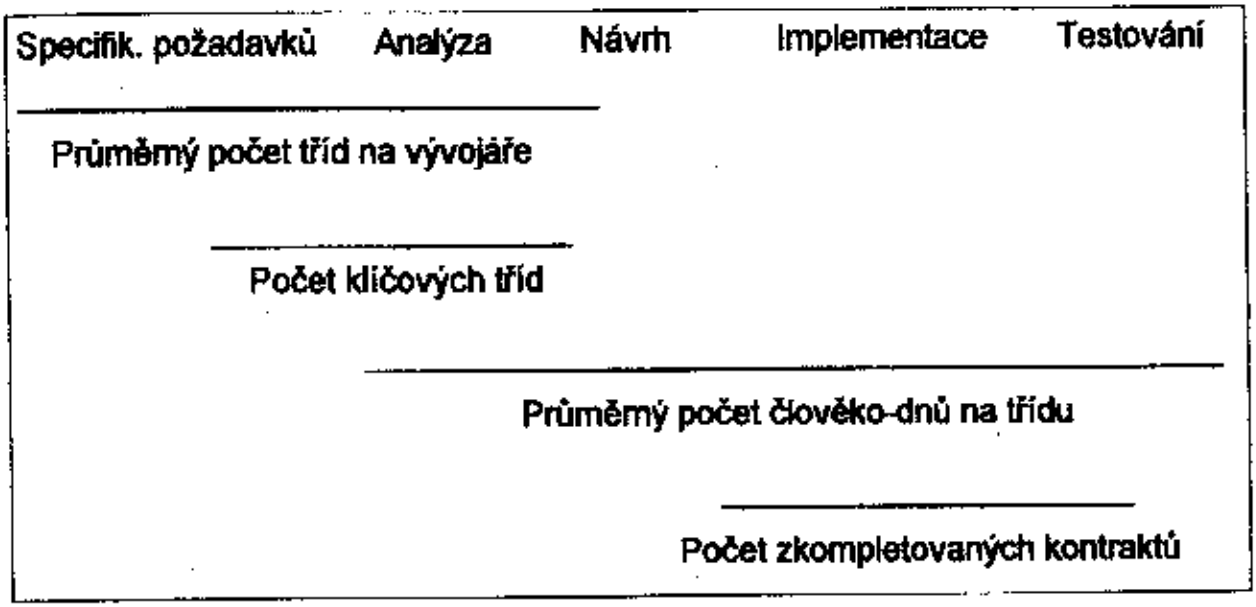
Za větší iterace považujeme takové, které vyžadují několikaměsíční úsilí. Počet iterací samozřejmě ovlivňuje dobu tvorby produktu.

Za optimální můžeme považovat 3 až 6 větších iterací.

Počet zkompletovaných kontraktů

Za kontrakt považujeme zjednodušující abstrakci skupiny veřejných činností, které jsou realizovány subsystémy a třídami pro jejich klienty. Dokončení důležitějších rozhraní předpokládané "veřejné služby" v systému. Je nejlepším celkovým indikátorem stavu při kompletování systému.

Vždy by mělo být během libovolné plánované iterace ukončeno několik kontraktů.



Tab. 7.2. Vztah proj. metrik k fázím životního cyklu projektu

	Odhadování	Plánování	Personál
Rozměr aplikace			
Počet scénářů (NSS)	V		
Počet klíčových tříd (NKC)	Z		
Počet podpůrných tříd (NSC)	V	V	V
Počet subsystémů (NOS)	V	V	
Počet pracovníků			
Průměrný počet člověko-dnů na třídu (PDC)	V		Z
Prům. počet tříd na vývojáře (CPD)	V		Z
Plánování			
Počet větších iterací (NMI)		V	
Počet zkompletovaných kontraktů (NCC)	Z		

Označení: V - metrika je vhodná k použití Z - metrika je zvláště doporučená

Tab. 7.3 Použití projektových metrik

7.2. Metriky návrhu

Rozměr metod

Počet vyslání zpráv

Týká se počtu vyslaných zpráv v dané metodě rozčleněných podle typu zpráv (unární - bez argumentu, binární - s jedním argumentem, klíčová slova). Tímto se kvantifikuje rozsah metody relativně nezaujatým způsobem.

Počet příkazů

Co znamená příkaz je dáno jazykem. U této metriky se předpokládá, že hodnota tolerančního prahu je rovna přibližně 80% prahu počtu vyslání zpráv.

Řádky kódu

Tato metrika měří počet fyzických řádků aktivního kódu v metodě.

Toto měření kvantifikuje rozměr metody aniž by bralo v úvahu takové faktory jako např. styl programování. Mnozí proto tuto metriku nedoporučují.

Průměrný rozsah metody

Rozsah metod je dobrým indikátorem kvality OO návrhu. Průměrný rozsah metod v projektu by měl být malý. Větší hodnoty rozsahu indikují funkcionální přístup.

Průměrný rozsah metod pro projekty ve Smalltalku by měl být pod 6 řádků kódu, pro C++ se uvádí prahová hodnota 18.

Interní charakteristika metod

Složitost metody

Většina způsobů měření složitosti metod se týká takových faktorů jako je počet rozhodovacích prvků v kódu, vytvářených pomocí IF-THEN-ELSE a ostatních struktur. Toto ale není užitečné u OO kódů jednotlivých metod, které jsou krátké a nepoužívají např. příkazy CASE.

U OO projektů se doporučuje měření složitosti vycházející z počtu a typů vysílání zpráv v rámci dané metody. Pro jednotlivé typy zpráv jsou uváděny váhy v rozmezí 0.3 - 5.0.

Rozměr třídy

Počet veřejných instančních metod ve třídě

Veřejné metody jsou přístupné jako služby pro ostatní třídy. Těmito službami se může nejlépe vyjádřit množství práce, kterou třída vykonává.

Průměrná hodnota se uvádí více než 20.

Počet instančních metod ve třídě

Celkový počet instančních metod zahrnuje veřejné, chráněné a privátní. Horní hranice pro třídy UI se pohybuje kolem 40.

Průměrný počet instančních metod ve třídě

Počet instančních metod ve třídě koreluje s množstvím odpovědnosti třídy. U dobrých OO projektů je inteligence a zátěž distribuována mezi více kooperujícími objekty.

Průměrný počet instančních metod se pohybuje v rozmezí 12 - 25.

Počet instančních proměnných ve třídě

Třída, která má více instančních proměnných, má více vztahů s ostatními objekty v systému.

Průměrný počet instančních proměnných ve třídě

Průměrné počty instančních proměnných se uvádí 3 - 9.

Menší průměrný počet umožňuje zvýšení opakovaného používání.

Počet třídních metod ve třídě

Třídy jsou objekty provádějící globální služby pro své instance. Počet metod dostupných pro třídu a ne její instance ovlivňuje rozměr třídy. Počet třídních metod by měl být obecně relativně malý ve srovnání s počtem metod instancí.

Počet třídních proměnných ve třídě

Třídní proměnné poskytují informace o společných objektech všem instancím dané třídy. Obyčejně počet třídních proměnných je relativně malý ve srovnání s instančními proměnnými.

Dědičnost tříd

Hladiny zahnízdění třídní hierarchie

Třídy jsou z hlediska dědičnosti organizovány hierarchicky ve stromové struktuře se základnou u nejvyšší třídy, nazývané kořen.

Vzdálenost třídy od kořene je nazývána hladina zahnízdění.

U většiny projektů se tato hladina pohybuje v rozmezí 2 - 6.

Počet abstraktních tříd

Abstraktní třídy slouží k tomu, aby se usnadnilo vícenásobné používání metod u podtříd. Nemají instance, reprezentují však generalizaci konceptů začleněných v kolekci podtříd.

Doporučuje se, aby (10 - 15) % tříd bylo abstraktních.

Použití vícenásobné dědičnosti

Některé jazyky jako C++ umožňují třídě dědění stavu a chování od více nadtříd. (Smalltalk jen od jedné).

Někteří autoři vícenásobnou dědičnost nedoporučují.

Dědičnost metod

Počet překrytých metod v podtřídě

V podtřídě může být definovaná metoda se stejným jménem jako u nadtřídy. To znamená, že příchod zprávy iniciuje v podtřídě tuto metodu namísto metody v nadtřídě.

Počet metod děděných podtřídou

Počet metod děděných podtřídou by měl být vysoký. Malý počet signalizuje nedobrou hierarchii tříd.

Počet metod doplněných v podtřídě

Metody doplněné v podtřídě rozšiřují chování nadtřídy. Existence podtřídy bez doplněných metod je sporná.

Specializační index

Specializace způsobuje rozšiřování chování ve srovnání s úplným využíváním existujícího chování. Specializace pomocí podtříd zahrnuje: přidávání nových metod; rozšiřování chování existujících metod nadtříd; překrývání metod metodami s novým chováním; rušení metod jejich překrytím metodami bez chování.

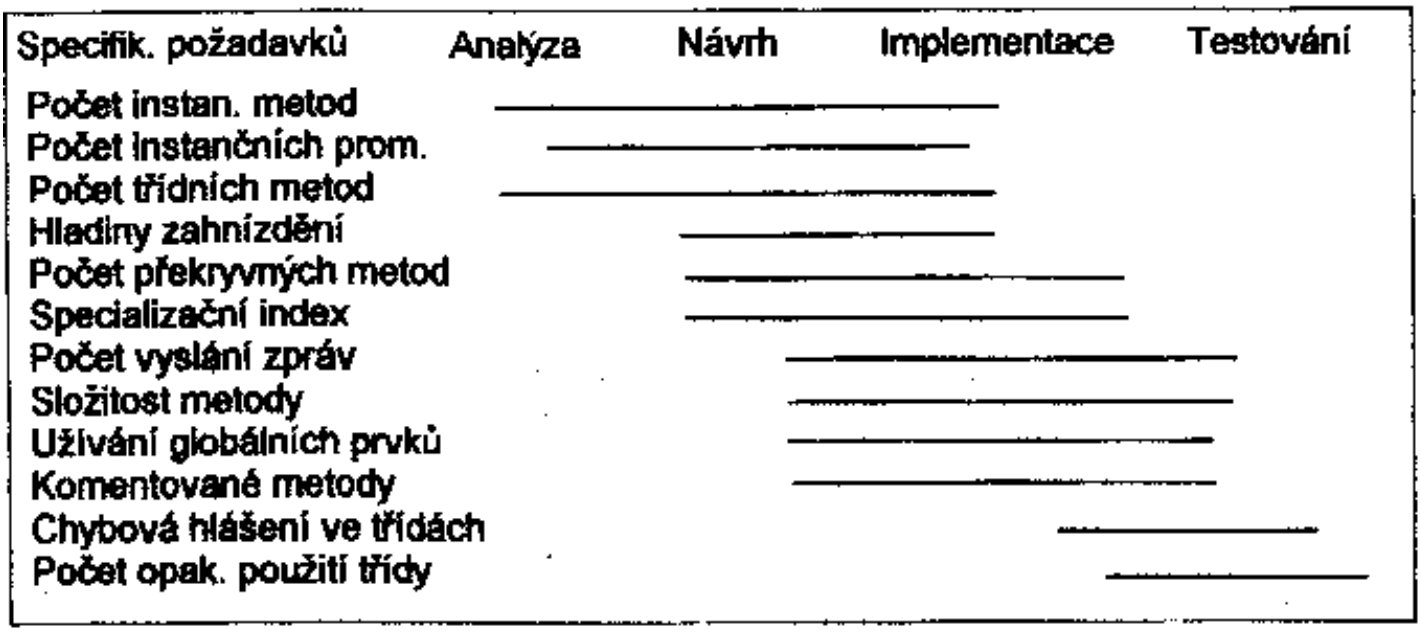
Počty prvků uvedených kategorií jsou hodnotami této metriky.

Interní charakteristiky tříd

Soudržnost tříd
Užívání globálních prvků
Průměrný počet parametrů v metodě
Užívání funkcí, které umožňují porušení zapouzdření
Procento funkčně orientovaných programů v projektu
Průměrný počet řádků komentáře v metodě
Průměrný počet komentovaných metod

Externí charakteristiky tříd

Spojování tříd
Počet opakovaných použití třídy
Počet zrušených tříd/metod



Tab. 7.4. Vztah návrh. metrik k fázím životního cyklu projektu

8. Závěr

V literatuře existuje mnoho metrik, mnohé z nich jsou zcela syntetické, většinou nebyly dostatečně verifikovány a nebyla ověřena jejich užitečnost. Je však řada důležitých atributů produktů a procesů, které mohou být měřeny průkazně. Je důležité najít měření, která podporují specifické cíle organizace a která je možno interpretovat a konstruktivně používat. Nejlepší je srovnání s podobnými projekty realizovanými ve stejných podmínkách. Kriteriem pro dobrou metriku je to, že víme proč ji chceme použít, že může být použita snadno a přesně a také to, že víme, jak použijeme výsledných hodnot. Nemáme-li jistotu v některém z těchto bodů, nejde o vhodnou metriku.

Literatura:

[1] Albrecht, A.J.: Software Function, Source Lines Code and Development Effort Prediction. Softw. Engineering SE-9, 1983
[2] Boehm, B.W.: Software Engineering Economics. SE-10,1984
[3] Král J., Demner J.: Softwarové inženýrství, Academia Praha 1991
[4] Lorenz M.,Kidd J.: Object-Oriented Software Metrics, Prentice Hall, 1994
[5] McCabe, T.J.: A Complexity Measure, SE-8, 1976
[6] Kitchenham B.: Software Metrics, Sofsem 1993
[7] Halstead M.: Elements of Software Science, Holland 1977
[8] Mayerhauser A.: Software Engineering, London 1990

Autor: RNDr. Jitka Kreslíková, CSc.
Ústav informatiky a výpočetní techniky, FEI VUT
Božetěchova 2
612 66, Brno 12
tel: 05-7275260
e-mail:kreslika@dcse.fee.vutbr.cz