VÝZNAM MĚŘENÍ V TESTOVÁNÍ SOFTWARU

Anna Borovcová
Komix s.r.o., Anna.Borovcová@gmail.com
Alena Buchalcevová
Katedra informačních technologií, VŠE Praha, buchalc@vse.cz

ABSTRAKT:

Úkolem testování je získávání informací skrze zkoumání produktu. Tyto informace jsou pak využívány k manažerským rozhodnutím o řízení procesu vývoje a testování softwaru. Čím lépe a pečlivěji jsou tyto informace zpracovávány, tím zasvěcenější a odbornější rozhodnutí mohou manažeři činit. Příspěvek podrobně vysvětluje význam měření, vhodné kombinace metrik a interpretace výsledků. Vedle toho je představen přehled běžně používaných metrik spolu s vysvětlením a vzorci pro jejich výpočet. Metriky jsou rozděleny do tří skupin: metriky založené na chybách reportovaných během testování, metriky založené na provedených testech nebo sledující účinnost těchto testů a metriky založené na kódu, kam mimo jiné spadá skupina různého pokrytí kódu. Příspěvek je doplněn příklady včetně možností interpretace různých získaných výsledků.

ABSTRACT:

The task of testing is obtaining information through the examination of the product. This information is then used for managerial decisions about development process and software testing. The better and more carefully the information is processed, the more informed and more expert decisions managers can make. This article explains in detail the importance of measurement, why it is necessary to form appropriate combinations of metrics and interpretation of gained results. In addition, it presents an overview of commonly used metrics, along with explanations and formulas for their calculation. The metrics are divided into three groups: metrics based on bugs found during testing, metrics based on tests or measuring test efficiency and metrics based on a code, where falls among other things a group of various code coverage. The paper is accompanied by examples, including different interpretations of gained results.

KLÍČOVÁ SLOVA:

softwarové metriky, zajišťování kvality, testování softwaru, test management, project management

KEYWORDS:

Software Metrics, Quality Assuarance, Software Testing. Test Management, Project Management

1. Úvod

Bez průběžného měření je problematické, ne-li nemožné, řídit jakoukoli lidskou činnost. Efektivní řízení a hodnocení procesu vývoje a testování vyžaduje průběžné měření kvality, které je zásadní pro poskytování informačních podkladů pro manažerská rozhodnutí. Primární oblastí, ve které se sbírají data pro vyhodnocení procesu vývoje softwaru, je testování.

Zatímco testování bylo zpočátku chápáno jako spouštění programu za účelem hledání chyb [6], od začátku osmdesátých let se na něj začalo postupně nahlížet více jako na nástroj získávání jakýchkoli informací o kvalitě produktu skrze zkoumání produktu. Tento posun můžeme vidět například v moderní Bachově definici: "Testování je dotazování se softwaru za účelem jeho ohodnocení" [1].

V současnosti se tedy testování vyvinulo do procesu sběru a třídění informací získaných skrze zkoumání produktu. Tyto informace, které testování poskytuje, slouží primárně manažerům, kteří jsou zodpovědní za vývoj produktu a jeho kvality. Z tohoto důvodu také plánování a řízení procesu testování zahrnuje identifikaci měřítek a kritérií kvality s tím, že vyhodnocené údaje slouží pro podporu řízení celého životního cyklu vývoje.

Proces testování v rámci vývoje softwaru popisují metodiky vývoje softwaru, které definují principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení [3]. Metodik existuje velké množství a liší se zaměřením, rozsahem, oblastí, pro kterou jsou určeny, přístupem k řešení a v neposlední řadě také úrovní podrobnosti, na které jsou popsané jednotlivé postupy, procesy, doporučení, techniky a další složky metodiky. V [4] jsou porovnány přístupy k testování softwaru v několika vybraných metodikách vývoje softwaru. Součástí metodik by však vždy mělo být plánování a řízení kvality nehledě na konkrétní použité metody.

V zahraniční literatuře jsou některé základní metriky popsány a podrobně vysvětleny v Software Testing Fundamentals: Metods and Metrics [5]. U nás se měřením kvality softwaru zabývá především Jiří Vaníček, který vidí důvod nedostatku informací o měření kvality v neochotě firem sdílet praxí ověřené principy [8]:

"*Zatímní pokusy získat reprezentativní výběr atributů a jejich měr vedly vždy k zcela nevyrovnaným, víceméně náhodným souborům. Přitom je jisté, že při hodnocení produktů téměř všechny velké softwarové firmy míry využívají a nehodnotí jakost pouze intuitivně. Zkušenosti s užitím těchto měr však považují za cenné „know how“ a z pochopitelných důvodů se s nimi nesvěřují.*"

Stručný přehled základních metrik, kterému ale chybí popis jejich interpretace, je možné nalézt i v kapitole Metriky v diplomové práci Testování webových aplikací [2], jejímž přepracováním a rozšířením o nové poznatky a o zkušenosti s interpretací výsledků měření tento příspěvek vznikl.

V příspěvku používám termín metrika, tak jak je to zvykem nejen v české, ale zejména zahraniční literatuře týkající se tohoto tématu a v odborných kruzích vedoucích testování a manažerů kvality.

2. Co měříme

Metriky kvality jsou založeny na zpracování údajů, které jsou během testování zaznamenány. Velmi důležité tedy je, jaké informace jsou na daném projektu uchovávány, jaké se používají nástroje, metodiky a dohodnutá pravidla spolupráce dodavatele a zákazníka.

Převážná většina údajů o kvalitě pochází ze systémů pro správu chybových hlášení a systémů pro řízení testování. Další informace mohou pocházet ze systémů pro vykazování zaměstnanců a jiných účetních systémů nebo z oblastí obchodu a marketingu.

Chybové hlášení typicky vytváří řadový tester a hlášení obsahuje informace získané při nalezení chyby (viz tabulka 1).

Položka	**Popis pole**
Popis chyby	Stručné shrnutí a podrobný popis chyby
Závažnost	Indikace, jak podstatné jsou dopady chyby. Dělení chyb dle závažnosti závisí na pravidlech daného projektu, např. závažnost může být na kritická (1 nebo A), vysoká (2 nebo B), normální (3 nebo C), nízká (4 nebo D), velmi nízká (5 nebo E).
Komponenta	Ve které části aplikace byla chyba objevena
Objeveno ve verzi	Ve které verzi aplikace byla chyba objevena
Test	Provázání s testem, který kontroluje přítomnost chyby
Prostředí	Na jakém prostředí byla chyba objevena
Stav chyby	Momentální stav chyby, neustále se mění v průběhy životního cyklu chyby
Vlastník	Kdo chybu nahlásil
Datum nahlášení chyby	

Tabulka 1: Základní položky vyplňované při založení hlášení chyby

Další údaje jsou doplněny v průběhu životního cyklu chyby dalšími osobami, které přijdou do styku s chybou. Tyto údaje jsou uvedeny v tabulce 2.

Položka	**Popis pole**
Priorita	Indikace, jak rychle je třeba chybu opravit, obvykle slovní nebo číselná, hodnoty jsou definované pravidly na projektu
Cílová verze	Do které verze má být zahrnuta oprava
Přiřazená osoba	Kdo se momentálně zabývá vyřešením chyby
Datum uzavření chyby	Datum, kdy byla chyba uzavřena. Uzavřená chyby, které jsou nějakým způsobem už vyřešené, jsou tedy v některém z koncového stavu životního cyklu chyby. Neuzavřené chyby jsou označované jako otevřené.
Doba opravy	Kolik času zabrala programátorovi oprava chyby

Tabulka 2: Základní položky vyplňované během životního cyklu hlášení chyby

V sofistikovaných systémech není potřeba, aby uživatel všechny údaje zaznamenával ručně, systém za něj zaznamená co nejvíce informací automaticky, zejména časy změn hlášení.

Při testování se často počítá s počtem provedených testů nebo počtem hodin strávených testováním. Při testování se využívá různých testovacích technik, u každé z nich může test vypadat odlišně, pokud jde o strukturu i dobu provádění. Proto je obtížné počet testů sledovat. Mezi nejběžnější testovací techniky patří testování dle scénářů (scenario testing), kde je jednotkou testu jeden testovací případ (test case), testování jednotek (unit testing), kde se za jeden test počítá jedna testovací jednotka a průzkumné testování (free, ad hoc, exploratory), kde není možné počet testů zaznamenat.

3. Kombinování metrik

Počet chyb není sám o sobě dostatečně vypovídající metrikou. Teprve v kombinaci s dalšími vlastnostmi, jako je třeba dělení chyb podle závažnosti, stavu, komponenty, ve které se chyba nalézá, vzniká sada užitečných měřítek, jež v kontextu s dalšími vhodně zvolenými měřítky poskytují hodnotné informace.

Například pokud víme, že na projektu je 130 otevřených chyb a zbývá měsíc z celkově ročního projektu do ukončení, má taková informace minimální informační hodnotu. Pro vytvoření představy o alespoň některém aspektu projektu, je třeba zahrnout další informace, jako je srovnání s podobným projektem, předchozím stavem ze stejného projektu nebo rozdělení podle závažnosti a komponenty.

Tabulka 3 je rozšířením prvotní informace o 130 otevřených chybách o počty otevřených chyb dle závažnosti a komponenty.

Komponenta	**# závažných**	**# normálních**	**# drobných**	**Celkový počet chyb**
core	4	1	0	5
crawler	0	4	0	4
datastore	1	6	0	7
downloader	4	5	1	10
FSNG server	5	17	3	25
search engine	3	4	2	9
webový klient	0	20	50	70
Celý produkt	17	57	56	130

Tabulka 3: Dělení chyb podle komponenty a závažnosti

V tabulce 3 vidíme, že nejhůře na tom jsou komponenty webový klient a FSNG server. Chyby u webového klienta nejsou nijak závažné, ale pokud by ho viděl zákazník, pravděpodobně by byl kvalitou velmi znechucen. K tomu, abychom odhadli, nakolik velkým problémem jsou další komponenty, je zapotřebí vědět, jak jsou tyto komponenty velké a konkrétně co za chyby jsou u nich hlášeny.

Pokud by tedy manažer dostal jako podklad tabulku z výše uvedeného příkladu, už by si snadněji udělal představu o tom, jaké další otázky položit a na co se zaměřit při zjišťování stavu projektu.

Tento příklad byl založen pouze na počtu chyb a jejich dělení. Teprve dalším rozšířením o další měřítka, jako je třeba čas a cena, a následným výběrem vhodné kombinace dalších metrik vznikají komplexnější sofistikované pohledy na stav procesu vývoje. Při hledání odpovědi na jakoukoli otázku ohledně procesu vývoje a testování je tedy hlavní zásadou komplexnost. Podklady, které testování poskytuje manažerům pro řízení kvality softwaru, by měly obsahovat pohledy na různé aspekty vývoje a testování, z nichž každý by měl být vytvořen vhodnou kombinací měřítek tak, aby co nejpřesněji poskytoval informace o daném aspektu.

4. Přehled běžně používaných metrik

Kombinováním různých atributů, které jsou během testování zaznamenávány a měřeny vznikají různé metriky kvality vývoje softwaru. Následující přehled běžně používaných metrik vychází až na označené výjimky z vlastních zkušeností z praxe.

4.1 Metriky založené na chybách

Mezi nejjednodušší metriky patří sumarizace nahlášených chyb dle jednotlivých atributů. Podle potřeby se znázorňují ve variantách počet všech či pouze otevřených chyb a celkově za produkt či pouze chyby nalezené v konkrétní verzi. Rozdělení počtu chyb můžeme sledovat u libovolného atributu, který u hlášení chyby vyplňujeme, například:

- Počet chyb dle stavu chyby,
- Počet chyb dle priority,
- Počet chyb dle závažnosti chyby,
- Počet chyb dle komponenty chyby,
- Počet chyb dle testu, ve kterém byla chyba objevena,
- Počet chyb dle verze, ve které byla chyba objevena,
- Počet chyb dle případu užití, ve kterém je chyba objevena.

Přidáním dalšího atributu vytvoříme kontingenční tabulku (anglicky pivot table), která zaznamenává počty chyb dle kombinace dvou vybraných atributů. Příkladem takové kontingenční tabulky je tabulka 3 v předchozí kapitole. Nejvíce používaná bývá kombinace stavu a dalšího atributu, například:

- Počet chyb dle stavu a priority,
- Počet chyb dle stavu a verze,
- Počet chyb dle stavu a komponenty.

Výše uvedené metriky poskytují řadu čísel, která se znázorňují v podobě tabulky nebo grafu. Výsledkem měření dle následujících metrik je jedno číslo, vypočtené podle daného vzorce, které se většinou poměřuje za různé testovací cykly a sledují se jeho změny.

- Průměrné stáří chyby do reakce = suma počtu dní (hodin) od zahlášení chyby do první reakce na chybu za všechny chyby / počet chyb. [1]
- Průměrné stáří chyby do vyřešení = suma počtu dní (hodin) od zahlášení chyby do první reakce na chybu za všechny chyby / počet chyb. [1]

[1] Pro výpočet průměrného stáří chyby za určité období, počítáme pouze chyby reportované v daném období.

- Průměrný počet vracení chyby = celkový počet vracení chyb / počet chyb. [2]
- Rychlost nalezení chyby = počet nalezených chyb / počet hodin strávených jejich hledáním. [3, 4]
- Pravděpodobnost opravení chyby = (počet opravených chyb/ počet nalezených chyb) * 100. [4, 5]
- Průměrná cena nalezení chyby = náklady na testování / počet nalezených chyb.
- Průměrná cena opravy chyby = náklady na opravy / počet opravených chyb.

4.2 Metriky založené na testech

Skupina metrik, která se zabývá primárně testy a jejich efektivností, je poněkud chudší, nicméně je velmi důležitá pro sledování vhodnosti použitého testovacího mixu a indikuje potřebu jeho obměny nebo zpětně hodnotí jeho účinnost.

- Pokrytí testů – (počet provedených testů / celkový počet připravených testů) * 100. [4]
- Akceptační připravenost – počet úspěšných testů / počet všech testů. [6]
- Efektivnost testu – (počet chyb nalezené testem / celkový počet nalezených chyb) * 100. [4, 7]
- Rychlost nalezení chyby testem – počet chyb nalezených testem / počet hodin běhu daného testu.
- Účinnost testovacího úsilí – počet opravených chyb během testování/ celkový počet chyb vůbec kdy nalezených v aplikaci * 100. [4, 8]

4.3 Metriky založené na kódu

První skupinou metrik zabývající se zdrojovým kódem jsou intensity chyb, které sledují počet chyb, které programátor udělá vzhledem k určité části kódu:

---

[2] Chyba je vrácena, pokud programátor označil chybu za opravenou, ale tester vrátil chybu programátorovi, protože s opravou není spokojen.

[3] Podle toho, co vše se počítá do počtu hodin se rozlišují rychlost nalezení chyby a rychlost nalezení a zahlášení chyby. Pokud by se do počtu hodin započetli i jiné činnosti než testování a reportování chyb, jednalo by se už o úplně jinou metriku.

[4] Výpočtový vzorec je převzat z Software Testing Fundamentals: Metods and Metrics [5].

[5] V praxi se počtem opravených chyb chápe počet chyb, které byly opraveny a jejich oprava byla potvrzena testerem.

[6] Akceptační připravenost sleduje, jaká část aplikace je už hotová. Za předpokladu, že množina testů, kterými má aplikace projít, je známa a připravena, je možné sledovat, kolika procenty testů aplikace už dokáže projít. Protože testy stejně jako chyby se dají dělit do mnoha skupin, a ne u všech je použití této metriky vhodné, uvažují se v praxi pouze funkční testy aplikace. Testy by měly být přibližně stejného rozsahu.

[7] Celkový počet chyb je chápán jako všechny chyby nalezené v dané verzi aplikace, tedy všemi možnými testy plus chyby nalezené v dané verzi uživateli, pokud danou verzi uživatelé viděli. Tato metrika slouží k porovnání efektivnosti různých typů testů, porovnání má smysl pouze tehdy pokud vezmeme v úvahu i čas strávený během testu. Pokud jsme tedy např. strávili prováděním jednoho typu testů 90 hodin a efektivnost byla 87,85% a druhým typem testů 10 hodin a efektivnost byla 12,15%, dostaneme že (87,85 /90) = 0,9761 je menší, než (12,15 /10) = 1,215. Tedy v závislosti na době strávené prováděním daného typu testu je efektivnost druhého typu testu větší.

[8] Účinnost testovacího úsilí se počítá několik měsíců po předání do produkce, kdy je pravděpodobnost, že všechny nebo aspoň většina chyb, na které uživatelé mohou najít, už jsou zahlášené.

- Intenzita chyb – (počet chyb / velikost kódu). Velikost kódu je například v tisících řádcích kódu (KLOC) nebo v předaných zdrojových instrukcích (SSI).

Následující metriky vzniklé na základě programovacího kódu tvořícího aplikaci se odvíjejí od některého typu pokrytí kódu testy. Pokrytí se nemusí týkat jen kódu, můžeme sledovat, kolik případů užití máme pokrytých, kolik zákaznízkových požadavků, ale výhodou pokrytí kódu je přesnost a jednoznačná definice. Metriky pokrytí kódu vypovídají o tom, kolik procent příkazů, hran, podmínek, cest, ale i například funkcí bylo otestováno. Nejznámější typy pokrytí jsou:

- Pokrytí příkazů (také pokrytí řádků) – T splňuje kritérium pokrytí příkazů pro daný kód K, jestliže pro každý příkaz p náležící kódu K existuje test t z množiny T, že při provedení t bude spuštěn příkaz p.
- Pokrytí hran (také pokrytí rozhodnutí) – Představme si graf, kdy příkazy tvoří uzly a možné přechody mezi nimi hrany. Pak za sebou provedené příkazy tvoří hranu a podmínka uzel, ze kterého vedou dvě hrany, jedna pro true a jedna pro false hodnotu. Pak množina testů T splňuje kritérium pokrytí hran pro daný kód K, jestliže pro každou hranu h výše popsaného grafu existuje test t z množiny T, že při provedení t projde výpočet hranou h.
- Pokrytí podmínek – Množina testů T splňuje kritérium pokrytí podmínek pro daný kód K, jestliže splňuje kritérium pokrytí hran a pro každou složenou podmínku platí, že pro každou její část p existují testy t, u z množiny T, že při provedení t se p vyhodnotí kladně a při provedení u záporně.
- Pokrytí cest – Množina testů T splňuje kritérium pokrytí cest pro daný kód K, jestliže splňuje kritérium pokrytí podmínek a pro každou cestu C v grafu kódu spojující vstupní a výstupní uzel grafu a obsahující nejvýše n cyklů existuje test t z množiny T, že při provedení t projde výpočet cestou C.

Předchozí definice pokrytí vycházejí z materiálu RNDr. Jana Pavelky, CSc. [7].

5. Interpretace metrik, jejich význam a úskalí

Říká se, že co nelze měřit, nelze řídit. Metriky nám poskytují přehled toho, co můžeme měřit a jak. Na rozdíl od měření nějaké délky předmětu či hmotnosti, kde jsou výsledky jasné a srozumitelné, u měření vlastností spojených se softwarem záleží na správném pochopení metriky a interpretaci naměřených výsledků.

Používání metrik a sledování naměřených údajů může být dobrým podkladem pro řízení procesu vývoje, ale jejich špatné použití přináší problémy namísto užitku. Například, pokud jsou testeři hodnoceni podle počtu objevených chyb. Dále působí spíše škodlivě, pokud je snaha dosáhnout ideální míry nebo křivky bez ohledu na další skutečnosti. Například snaha o dosažení určitého procenta pokrytí příkazů může být problematická, protože výsledek této konkrétní metriky je velmi zavádějící. U řady programů může být dosaženo vysokého procenta pokrytí bez objevení jediné chyby, stačí, aby se někdo záměrně snažil dosáhnout vysokého pokrytí pomocí nevhodných jednoduchých testů. Škodlivá je samozřejmě jakákoli chybná interpretace, protože znamená, že zásah, který manager provede, bude směřován jinde, než kde je skutečný problém. Tedy místo, aby se napravovala chyba v procesu, budou se dělat změny v něčem, co funguje. Právě kvůli těmto problémům je pochopení komplexity metrik a výsledků takto zásadní.

5.1 Interpretace výsledků

Bez znalosti kontextu nelze čísla jednoznačně interpretovat, ale je možné přiblížit proces interpretace tak, že budou nastíněny jednotlivé možnosti interpretace. Je na manažerovi, aby zhodnotil možné interpretace daných výsledků a zjistil, která z možných příčin stojí za daným výsledkem.

Příklad: Rychlost nalezení chyby

Jak bychom interpretovali výsledek 0,34 chyby za hodinu resp. 1 chyba za 3h? Jedna chyba za tři hodiny může "špatná", ale může být i ideální. Různé situace mohou vést ke stejným výsledkům, jaké výsledky tedy mohou odpovídat následujícím situacím?

Situace a) Tester dostane novou napůl vyvinutou verzi produktu a intenzivně ji 8 hodin testuje. Najde a nahlásí 15 chyb. Tedy zhruba 2 chyby za hodinu. To znamená, že tester je pozorný a aplikace má ve vývojovém stádiu řadu nedostatků.

Situace b) Tester dostane novou napůl vyvinutou verzi produktu a intenzivně ji 8 hodin testuje. Najde a nahlásí 30 chyb. Tedy 3,75 chyby za hodinu. - Tady už to začíná být příliš hodně. Nabízí se otázka, zda to není tím, že build byl špatně sestaven a nejedná se o nové chyby produktu, ale o to, že někdo provázal aplikaci se špatnou verzí databáze, nebo něco podobného. Rozhodně je příliš dobré číslo impulsem k tomu, ptát se, co je jeho příčinou. Může to ukazovat na problém vývoje, hromadné zadání chyb, o kterých se ví delší dobu, to, že na projektu se objevil vynikající tester, nebo naopak, tester, který neumí reportovat a přináší do systému pro reportování jen zmatek – špatně popsané, neověřené chyby.

Situace c) Tester dostane novou napůl vyvinutou verzi produktu. Má za úkol přetestovat opravy chyb nalezené v minulé verzi, při spuštění testovacích scénářů. 8 hodin provádí testovací scénáře, přitom zjistí, že z 20 chyb je jich možné 17 uzavřít, 3 zůstávají otevřené a objevily se 2 nové chyby. V takovém případě je rychlost objevení chyby 1 za 4 hodiny.

Je to problém? Není. Úkolem testera nebylo najít nové chyby, ale zjistit, zda byly staré chyby objeveny. Metrika rychlosti nalezení chyby by buď neměla zahrnovat čas strávený na jiných úkolech, nebo pokud se počítá třeba za týden nebo měsíc, mělo by se brát v úvahu to, že ne všechen čas byl stráven hledáním chyb.

Situace d) Produkt je pár týdnů před uvolněním do produktu. Téměř všechny očividné chyby byly opraveny. Testeři se věnují pouze hledání chyb, a přesto reportují chyby rychlostí 1 za 4 hodiny.
Takováto statistika bývá typická ve stavu, kdy už je produkt důkladně otestován. Chyby se hledají složitě a je těžké najít podmínky, kdy se chyby projeví. Snížení rychlosti reportování chyb je typické, když se začíná projekt blížit ke konci.

Situace e) Produkt je testován pouze dle předpřipravených testovacích scénářů, které pokrývají pouze základní toky. Testeři reportují chyby rychlostí 1 za 2 hodiny. Toto je případ, kdy se testy neprovádějí za účelem hledání chyb, ale za účelem dokumentace toho, že aplikace funguje dle dohodnuté specifikace.

Kdyby testeři měli za úkol hledat chyby, ne procházet dané scénáře, nacházeli by chyby několikanásobně rychleji. Chceme-li zvýšit rychlost nalezení chyby, je třeba doplnit o jiný druh testování.

Situace f) Tester dostane novou napůl vyvinutou verzi produktu. Má za úkol se soustředit pouze na hledání co nejzávažnějších chyb. V aplikaci je chyb dost, ale on nalezne jen 1 chybu za 3 hodiny. Pak je třeba hledat příčinu. Chápe tester funkčnost aplikace? Je na projektu úplně nový? Nemá dostatečnou kreativitu a zkušenosti k testování? Atd.

6. Závěr

Výsledky získané z měření kvality prováděného během testování slouží manažerům, kteří řídí vývoj softwaru, jako zpětná vazba. Řídit vývoj softwaru bez jakékoli zpětné vazby je jako řídit auto se zavřenýma očima. Čím je prováděné měření komplexnější a přesnější, tím má manažer lepší přehled o stavu projektu. Vidí, jaké překážky stojí v cestě, a může tak včas a adekvátně zakročit, problémy vyřešit či se jim vyhnout.

Příspěvek uvedl čtenáře do problematiky měření kvality softwaru, ukázal možnosti kombinování a interpretace výsledků a cesty, jak na základě výsledků odhalit problémy. Další postup je už v kompetenci manažera, který se musí rozhodnout, jaké nápravné opatření přijme v případě, že výsledky měření odhalují nějaký problém ve vývoji.

PODĚKOVÁNÍ

Tento příspěvek vznikl za podpory grantu IGA IG406050 Zavedení procesu řízení kvality jako integrální součásti metodiky vývoje informačního systému.

LITERATURA:

[1] Bach, James; Bolton, Michael: Rapid Software Testing V2.1.5, 2007, Satisfice Inc.
Dostupné z WWW: http://www.satisfice.com/rst.pdf
Datum čerpání zdroje: 20.3.2010

[2] Borovcová Anna: *Testování webových aplikací*, diplomová práce, MFF KU, 2008

[3] Buchalcevová, Alena: *Metodiky budování informačních systémů*, 1. vydání, Oeconomica, Praha , 2009

[4] Buchalcevová, Alena, Kučera, Jan:. Hodnocení metodik vývoje informačních systémů z pohledu testování,. Systémová integrace, 2008, roč. 15, č. 2, s. 42–54

[5] Hutcheson, Marnie L.: *Software Testing Fundamentals: Methods and Metrics*, John Wiley & Sons, 2003

[6] Myers, Glenford J.: *The Art of Software Testing*, Second Editio, Hoboken, New Jersey : John Wiley & Sons, Inc., 2004
Dostupné z WWW:
http://www.51testing.com/N_download/lib/TestingTechDL/ArtofSoftwareTesting.pdf Datum čerpání zdroje: 20.4.2010

[7] Pavelka, Jan: *Zajištění jakosti projektů*, materiály k předmětu Softwarové inženýrství (sweng56.ppt), KSI MFF UK, 2005

[8] Vaníček, Jiří: Novinky v normalizaci jakosti softwaru, Sborník konference Tvorba softwaru, TANGER, s.r.o a MARQ, Ostrava, 2005, s. 230 - 237
Dostupné z WWW:
http://formular-ekf.vsb.cz/formulare/F01/tsw/getfile.php?prispevekid=853
Datum čerpání zdroje: 31.3.2010