Kvalita software a techniky pro její zajištění

KIV/ASWI 2017-2018

Obsah a cíl

- Vysvětlení pojmu kvalita software
- Motivace pro zajištění kvality
- Základní techniky včasné detekce chyb

- Pochopit, že

  „where quality is pursued, productivity follows“

Jaký software je „kvalitní“

- Klíčový pojem: fitness for purpose
  - vhodnost pro zamýšlený účel použití
    => primární měřítko jsou uživatelské požadavky

- Aspekty kvality
  - vnější – spolehlivost, výkonnost, použitelnost, bezpečnost, …
  - vnitřní – architektura, odolnost proti změnám, testovatelnost, dokumentovanost, …
    - vnitřní kvalita je základem pro vnější

ISO 25010 Software Quality Model

- Functional Suitability
- Performance Efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

- (Původně ISO 9126)

Úroveň dosahované kvality

- Široké spektrum
  - „semestrální práce“
  - utilita
  - [systémový] produkt
  - systém odolný proti poruchám (*fault-tolerant*)
  - bezpečnostně kritický (*safety-critical*) systém
  - faktor dopadu na majetek a lidské životy

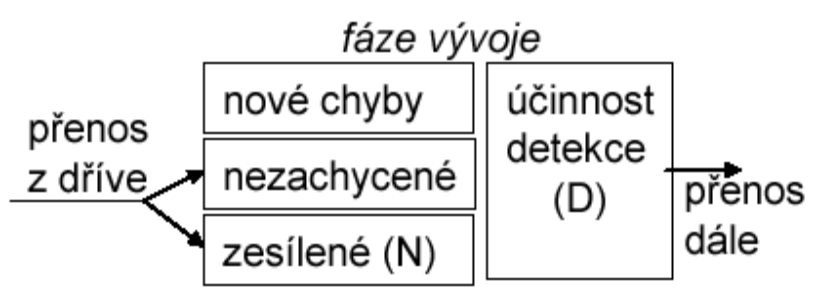
- Určení úrovně kvality důležité pro technické i organizační aspekty projektu

Opak kvality

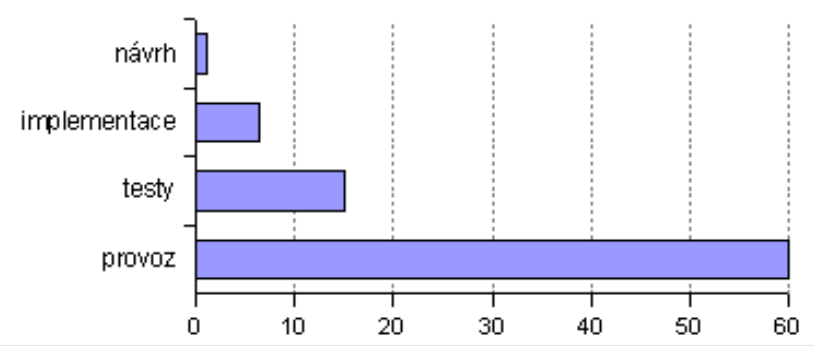
- Omyl (*error*)
  - přehlédnutí, omyl nebo špatné designové rozhodnutí programátora; důsledkem je
- Defekt (*defect*, též *fault* či *bug*)
  - závada, nedostatek v [zdrojovém kódu dodaného] produktu; důsledkem může být
- Chybový stav (*run-time fault*)
  - jiný než očekávaný (správný) run-time stav nebo výstup (sub)systému; důsledkem může být
- Selhání (*failure*)
  - neschopnost systému nebo jeho části vykonávat požadované funkce v požadovaných výkonnostních limitech; pozorovatelné navenek.

Řetězení a cena chyb

- Jednou vzniklá chyba působí řetězovou reakci
  - Model zesilování defektu (IBM 1981)



- Čím později odhalena tím dražší náprava
  - relativní cena odstranění chyby



- … následky chyby se zvětšují během vývoje

Způsoby zajištění kvality

http://www.commitstrip.com/en/2017/02/08/where-are-the-tests/

Způsoby zajištění kvality

- QA = Quality Assurance
- Verifikace (bezchybný produkt)
  a validace (správný produkt)

- Detekční a opravné techniky
  - cíl: najít a opravit již existující chybu
  - testování a ladění
    - typické v koncových fázích („výstupní kontrola“)
- Preventivní techniky
  - cíl: zabránit vzniku event. dalšímu šíření chyby
  - „racionální proces“ a „best practices“
  - kontroly a měření meziproduktů
    - častější v úvodních fázích

Detekční techniky

Viz předmět KIV/OKS

Testování

- Ověření správné funkčnosti implementace
- Úrovně testování
  - jednotkové → integrační → funkční → systémové
  - zátěžové, výkonnostní, bezpečnostní, instalační, použitelnosti
  - interní, akceptační
- Techniky pro testování
  - white-box, black-box
  - výběr dat (hraniční podmínky, fuzz testing, …)

Viz např: http://www.amazon.com/Software-Testing-2nd-Edition-Patton/dp/0672327988

Automatizované testování

- Základní kontrola kvality kódu
  - typicky unit testy
  - „Nemá-li [část kódu] automatizovaný test, který by dokazoval jeho funkčnost, je nutné jej pokládat za chybový.“
- Nástroje: JUnit, NUnit, …

- Viz vztah k SCM/sestavení
- Souputníci: refactoring, automatizovaný build
  - test-driven development

Zkouška těsnosti (Smoke Test)

- Cíl: ověřit, že sestavení vytvořilo funkční produkt
  - samotný překlad/linkování toto nezaručuje
  - kompletní testování trvá dlouho
    - potřebujeme rychlý základní test, jehož provedení „nebolí“
- Postup: vytvořit testy ověřující základní funkčnost, bez nároku na kompletní otestování
  - odchytí nejkřiklavější chyby, odpustí drobnosti
  - automatizace - spuštění, vyhodnocení
  - spuštění při každém buildu (vč. soukromého)
    - podle toho náročnost, rozsah, počet testů
    - může být založena na testech modulů
    - přidání nových funkcí/vlastností => nové testy těsnosti

re·gress Pronunciation Key (rĭ-grĕs′)
v. re·gressed, re·gress·ing, re·gress·es
v. intr.

1. To go back; move backward.
2. To return to a previous, usually worse or less developed state.

– dictionary.reference.com

Regresní testy

- Cíl: zajistit, aby nové funkce a vylepšení nesnižovaly kvalitu již hotového kódu
  - prevence stárnutí kódu: zanášení chyb při implementaci vylepšení
  - vymyslet mechanismus jak vytvářet vhodné testy
- Postup: ověřit build produktu pomocí testů, kterými již dříve prošel (kromě testů nových funkcí)
  - indikátor existence systémových problémů
    - určení zdroje chyby není nutné => testy integrační, modulů
  - testování změn v (nečekaných) integračních aspektech
  - spuštění při integračním sestavení, před velkými změnami
- Dobrý zdroj testů: chyby objevené QA, při validaci, zákazníkem
  - produkt selhal → napsat test, který to dokáže → přidat jej do sady regresních testů (odebírání testů ze sady není dobrý trik)

Statická kontrola kódu

- Ověření formální správnosti
  + dodržování pravidel + metriky

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
~Martin Golding

- Nástroje
  - překladač a jeho hlášení (!)
  - C: lint
  - Java: pmd, findbugs, checkstyle, JSR 305
- Postupy
  - Programming by Contract
  - review, párové programování
  - automatický build – výběr pravidel, průběžné úpravy

- Viz např: J.Kiml: Java code defect analysis – KIV/TSI podzim 2010

Formální verifikace

- Matematické důkazy správnosti
  - návrhu
  - implementace

- Model checking
  - formální model systému – Petriho sítě, algebry (CSP)
    - a-priori nebo získaný analýzou kódu
  - model checker => deadlock-free, liveness, …
  - soulad s implementací
- Nástroje: Java PathFinder, SPIN, PRISM, SMV, …

Preventivní techniky

Preventivní techniky

- Kontroly
  - zpětná vazba z metrik a automatizovaných testů (retrospektivy)
  - prověření meziproduktu nezávislým oponentem
    - dříve než se z něj začne vycházet v další práci
  - technická oponentura a podobné techniky
  - párové programování, refactoring
- Měření
  - kvantitativní ukazatele pomáhají najít slabiny kvality
  - přesnost a dokazatelnost, možnost statistik
  - GQM přístup, FURPS

Doporučení pro psaní bezpečného kódu

- JSF, MISRA C / C++
- EN 50128

Tabulka A.12 – Kódovací standardy

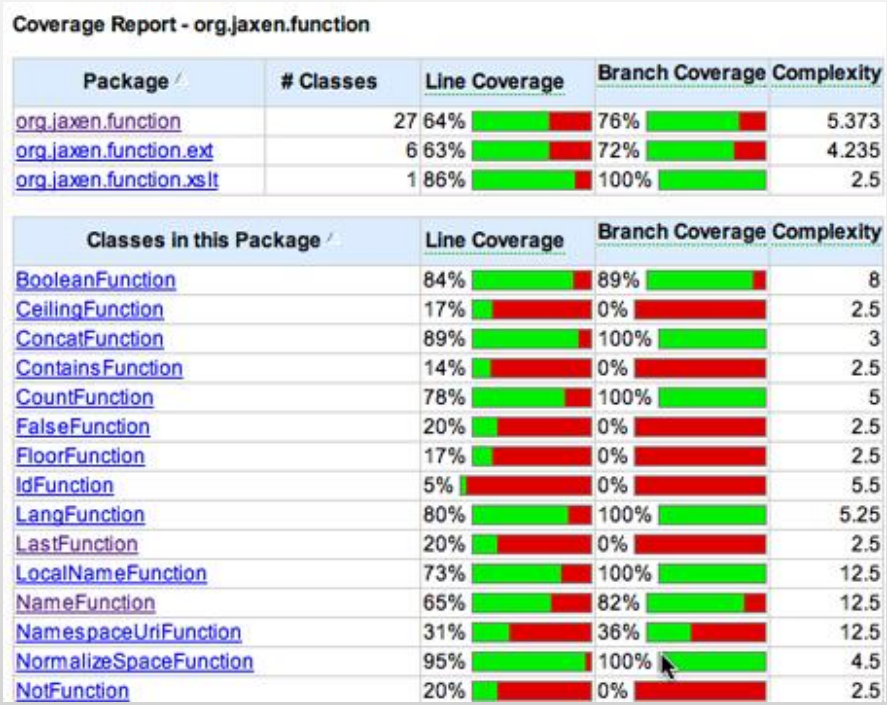
Technika/opatření	Odkaz	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Kódovací standard	D.15	HR	HR	HR	M	M
2. Pravidla pro styl kódování	D.15	HR	HR	HR	HR	HR
3. Žádné dynamické objekty	D.15	-	R	R	HR	HR
4. Žádné dynamické proměnné	D.15	-	R	R	HR	HR
5. Omezené používání ukazatelů	D.15	-	R	R	R	R
6. Omezené používání rekurze	D.15	-	R	R	HR	HR
7. Žádné nepodmíněné skoky	D.15	-	HR	HR	HR	HR
8. Omezená velikost a složitost funkcí, podprogramů a metod	D.38	HR	HR	HR	HR	HR
9. Strategie vstupní/výstupní bod pro funkce, podprogramy a metody	D.38	R	HR	HR	HR	HR
10. Omezený počet parametrů podprogramu	D.38	R	R	R	R	R
11. Omezené použití globálních proměnných	D.38	HR	HR	HR	M	M

Požadavek:

1) Je schváleno, že techniky 3, 4 a 5 mohou být součástí validovaného kompilátoru nebo překladače.

Statistické kontroly

- Základ: metriky
  - indikátor „někde je něco špatně“
- Stanovení očekávaných / správných hodnot
- Průběžné monitorování
  - technika zasévání chyb
- Korekce procesu, komponent při odchylkách



Coverage Report - org.jaxen.function

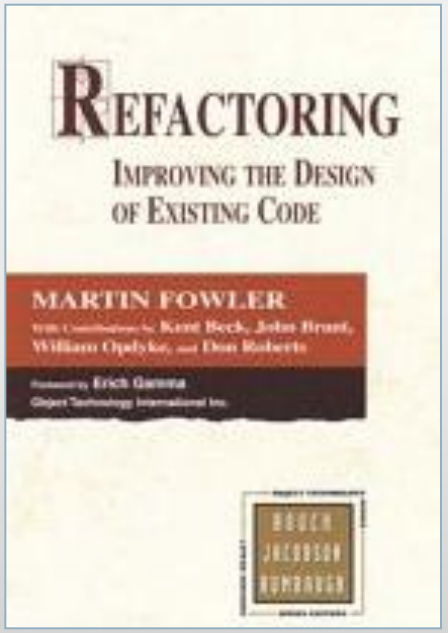
Package	# Classes	Line Coverage	Branch Coverage	Complexity
org.jaxen.function	27	64%	76%	5.373
org.jaxen.function.ext	6	63%	72%	4.235
org.jaxen.function.xslt	1	86%	100%	2.5

Classes in this Package	Line Coverage	Branch Coverage	Complexity
BooleanFunction	84%	89%	8
CeilingFunction	17%	0%	2.5
ConcatFunction	89%	100%	3
ContainsFunction	14%	0%	2.5
CountFunction	78%	100%	5
FalseFunction	20%	0%	2.5
FloorFunction	17%	0%	2.5
IdFunction	5%	0%	5.5
LangFunction	80%	100%	5.25
LastFunction	20%	0%	2.5
LocalNameFunction	73%	100%	12.5
NameFunction	65%	82%	12.5
NamespaceUriFunction	31%	36%	12.5
NormalizeSpaceFunction	95%	100%	4.5
NotFunction	20%	0%	2.5

Refactoring

- Změna interní struktury software, která jej činí srozumitelnějším a snáze upravitelným, aniž by změnila jeho vnější chování
  - též proces k takové změně vedoucí
- Detekce zapáchajícího kódu
- Změna designu, oprava
  - katalog úprav

- Nutný sourozenec: automatické testy



Technická oponentura

- Též Faganovská inspekce (Fagan, IBM 1976)
- Skupinová technika (využití diverzity pohledů)
  - Cíl: odhalit chyby v návrhu/kódu/dokumentu
  - Souběžný efekt: sledování standardů, vzdělávání
  - Ne: dělat potíže autorovi (neúčast vedení), hledat nápravu chyb

- Role ve skupině (cca 4-7 lidí)
  - moderátor – řídí diskusi
  - průvodce – předkládá dílo (není autor)
  - autor – vysvětluje nejasnosti
  - zapisovatel – zaznamenává nalezené problémy
  - oponenti – hledají chyby, obvykle podle seznamů otázek

Technická oponentura – postup

- Příprava
  - distribuce díla (moderátor), projití a hledání problémů (oponenti)
    - několik dní předem, cca 2h práce
- Schůzka
  - sekvenční procházení díla (průvodce či moderátor)
  - vznášení připomínek
  - zapisování nálezů (chyb a otevřených otázek)
    - nejvýše 2 hodiny
    - nepřipouštět dlouhé diskuse, řešení chyb (moderátor)
    - možná následná schůzka pro vyjasnění otázek
- Závěry
  - verdikt: v pořádku / drobné chyby / nutné přepracování / nová oponentura
  - autor odstraní chyby dle nálezů, moderátor zkontroluje
    - dokument „Nálezy oponentury“

Zhodnocení technické oponentury

- Přínosy
  - použitelné ve všech fázích životního cyklu
    - zejména v analýze a návrhu, kdy neexistují strojovězpracovatelné artefakty
  - velmi dobrá detekce chyb (až 75%)
    - některé studie uvádí 10-100x nižší výsledný počet chyb při používání inspekcí
    - většina chyb (až 60%) nalezena před testováním
  - výsledkem jsou nižší náklady na vývoj a vyšší produktivita
    - úspora 40-70% nákladů díky levnější opravě chyb v dřívějších fázích cyklu
    - ze studií IBM (60 projektů) plyne až o 35% vyšší DSLOC při použití inspekcí
- Nároky
  - náročné na čas – nejde automatizovat
    - podle IBM optimální rychlost procházení cca 60-80 SLOC / hod
  - je třeba zkušenost
    - důraz na zaškolení lidí, zejména moderátora
    - účinnost podle pořadí inspekce: č.1: 15%, č.2: 41%, č.3: 61% [Humprey89]

„Lehčí“ techniky

- Strukturované procházení
  - podobné Faganovské inspekci
  - menší důraz na formálnost, větší flexibilita
    - shodné: příprava předem, schůzka s procházením, checklist
    - není: striktně rozdělené role, následná kontrola oprav, statistiky
- Peer review
  - „kontrola nezaujatým čtenářem“
  - autor prochází kód a vysvětluje
  - kolega hledá problémy a komentuje
    - častý jev: autor najde chyby ve svém kódu, když jej má vysvětlit
- XP edition: Párové programování

Závěrečné poznámky

- → Systémy řízení jakosti

Prevence je vždycky lepší

- Účinnost detekce chyb

modelování, prototypování	65% avg	80% max
formální oponentury návrhu	55%	75%
neformální procházení	40%	60%
čtení kódu	40%	60%
integrační testování	45%	60%
testování modulů	25%	50%

- Jednotlivé způsoby doplňkové

Opravdová prevence chyb je …

- … být dobrým softwarovým inženýrem

- Osobní snaha o kvalitu
- Best practices
  - „dvakrát měř, jednou řež“
- Učit se, učit se, učit se