KIV/ASWI 2007/2008

Techniky zajištění kvality software

- Kvalita software
- Techniky včasné detekce

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
    $\Rightarrow$ 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

Úroveň dosahované kvality

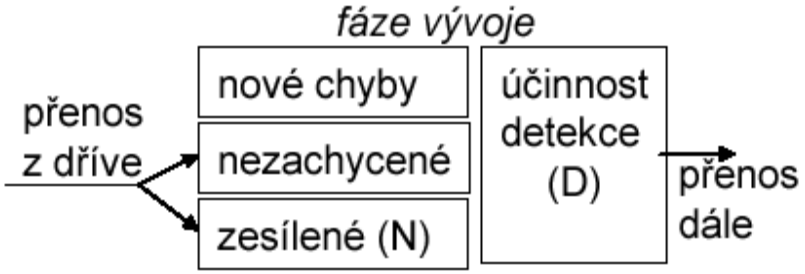
- Široké spektrum
  - » semestrální práce
  - » utilita
  - » systémový produkt
  - » *fault-tolerant* systém
  - » *safety-critical* systém
  - faktor dopadu na majetek a lidské životy
- Určení úrovně kvality důležité pro technické i organizační aspekty projektu

Opak kvality

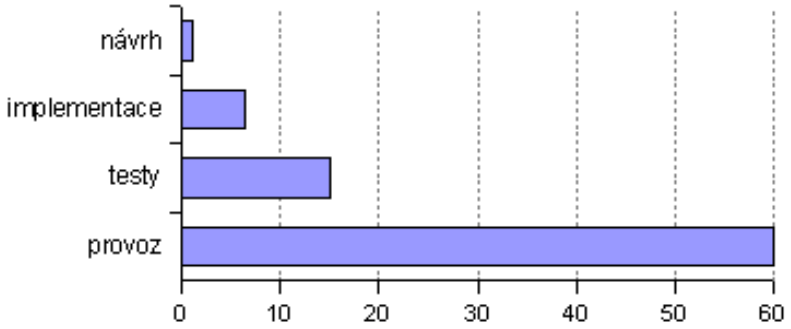
- Omyl (*error*)
  - přehlédnutí, omyl nebo špatné designové rozhodnutí programátora; důsledkem je
- Defekt (*defect*, též *fault* či *bug*)
  - závada, nedostatek v [zdrojovém kódu dodaného] produktu; důsledkem může být
- Chybový stav (run-time *fault*)
  - jiný než očekávaný (správný) run-time stav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

Prevence je vždycky lepší

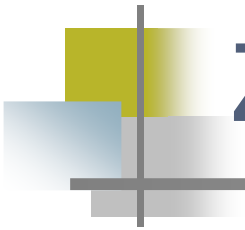
- Jednou vzniklá chyba působí řetězovou reakci
  - Model zesilování defektu (IBM 1981)



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



… následky chyby se zvětšují během vývoje



Způsoby zajištění kvality

Způsoby zajištění kvality

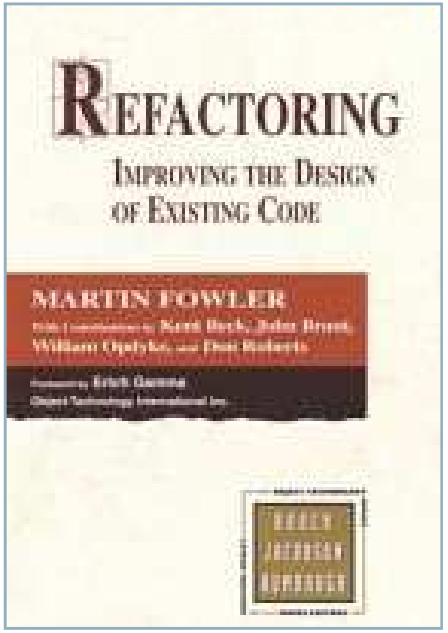
- Preventivní techniky
  - cíl: zabránit vzniku event. dalšímu šíření chyby
  - „racionální proces“ a „best practices“
  - kontroly a měření meziproduktů
    - » častější v úvodních fázích
- Detekční a opravné techniky
  - cíl: najít a opravit již existující chybu
  - testování a ladění
    - » typické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

Preventivní techniky

- Kontroly
  - automatizované testy
  - 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

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



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, ...

- Souputníci: refactoring, automatizovaný build
  - » test-driven development

Technická oponentura

- Též Faganovská inspekce (Fagan, IBM 1976)
- Skupinová technika (využití diverzity pohledů, cca 4-7 lidí)
  - Cíl: odhalit chyby v návrhu/kódu, sledování standardů, vzdělávání
  - Ne: dělat potíže autorovi (neúčast vedení), hledat nápravu chyb

- Role ve skupině
  - *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

Prevence je vždycky lepší (2)

- 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“
  - » 3-tier, test-first, refactoring, modelování, návrhové vzory, …
- Učit se, učit se, učit se

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

Statistické kontroly

- Základ: metriky
  - » indikátor „někde je něco špatně“
- Aplikace – spolehlivost
  - MTBF = MTTF + MTTR
  - dostupnost [%] = (MTTF / MTBF) x 100
- Kód – složitost, přehlednost
  - McCabe cyclomatic complexity, fan-in / fan-out
  - možná někdy případně i také LOC
  - pokrytí testy
- Projektové metriky
  - defect removal
  - project velocity / burndown