Způsoby prevence chyb v software, metriky a oponentury.

Wednesday, May 29, 2013 4:58 PM

[http://www.robertdresler.cz/2012/02/techniky-pro-prevenci-softwarovych-chyb.htm]

[aswi přednáška 04b.pdf, aswi 06-kvalita.pdf]

Způsoby prevence chyb

- Cíl: zabránit vzniku a dalšímu šíření chyby
- Racionální proces a best practices
- Kontroly a měření meziproduktů (častější v úvodních fázích)+
    - **Automatizované testy**
        - Základní kontrola kvality kódu
            - Typicky unit testy
    - **Prověření meziproduktu nezávislým oponentem** (dříve než se z něj začne vycházet v další práci)
    - **Technická oponentura a podobné techniky**
        - Viz oponentury.
    - **Párové programování, refactoring**
        - Párové programování
            - Metafora řidič (udává směr, vysvětluje, naslouchá) + navigátor (dohledává, kontroluje, pomáhá) - svědomí páru (společný cíl, plné nasazení, komunikace)
        - Refactoring
            - Změna interní struktury software, která jej činí srozumitelnějším a snáze upravitelným, aniž by změnila jeho vnější chování
            - Detekce zapáchajícího kódu
            - Změna designu, oprava
    - **Strukturované procházení**
        - Podobné Faganovské inspekci, menší důraz na formálnost
    - **Peer review**
        - Kontrola nezaujatým čtenářem
        - Autor prochází kód a vysvětluje
        - Kolega hledá problémy a komentuje
- Měření
    - Kvantitativní ukazatele pomáhají najít slabiny kvality
    - Přesnost a dokazatelnost, možnost statistik
    - GQM přístup, FURPS

Detekční a opravné techniky

- Cíl: najít a opravit již existující chybu
- Testování a ladění (typické v koncových fázích, tzv. výstupní kontrola)

Psaní čistého kódu

- snižuje riziko "ukrytí" zákeřných programových chyb už v prvotní fázi psaní kódu
- podporuje efektivnější ladění

Refaktorizace kódu = změna struktury kódu, která nemá vliv na jeho celkovou funkčnost (přejmenování proměnné/metody, vyjmutí kódu do samostatné metody,..)

- snížení složitosti kódu
- Zpřehlednění

Revize kódu (Code Review)

Kdy provádět revizi kódu? Ideální by bylo během nebo těsně po vlastní implementaci. Revize kódu je jedním ze základních aspektů párového programování (Pair Programming). Dvojice společně vyvíjí kód a revize probíhá neustále. Pokud neaplikujete párového programování, můžete nastavit povinnou revizi kódu při umisťování změn na server. Revizi můžete také provádět v rámci celého vývojového týmu na pracovních poradách.

Jak je revize kódu efektivní? Záleží na úrovni zkušeností vývojáře a "revizora". U začínajících vývojářů budou revize častější a revizorem by měl být zkušený pracovník. Při revizi kódu vytvořeného zkušeným vývojářem nemusí být objeveno mnoho chyb, ale celý proces může posloužit k tomu, že posluchačům budou předány zkušenosti a praktické rady. Technik revizí kódu je více a liší se svojí formálností, obsazením revizního týmu a způsobem evidence nalezených defektů.

Statická analýza (kontrola) kódu = analýza kódu, která je prováděna bez nutnosti spouštění programu – soubor preventivních technik. Ověření formální správnosti + dodržování pravidel + metriky

- Nástroje: překladač a jeho hlášení
- Postupy: párové programování

Dynamická analýza kódu

Volba technologií

Automatizované testování

Testování je technika dynamické analýzy kódu. Psaní testů by podle metodologie Test-Driven Development (TDD) mělo předcházet vlastní implementaci. Zkuste si tento přístup zažít a uvidíte, že se vám zalíbí. A pokud ne, věžte, že testy stejně musíte napsat. Jinak se pro vás stane prvotní vývoj a především pozdější zásahy do produkčního kódu noční můrou. Testy jsou indikátory chybových stavů.

První typ testů, který napíšete, budou jednotkové testy (unit test). Základní logika jednotkového testu je jednoduchá. Voláte metodu instance testované třídy s určitými vstupy a validujete výstupy. Pokud výstupy neodpovídají předpokladům, test selže a je nutno hledat chybu v implementaci. Lze prohlásit, že vyšší pokrytí testy lépe pojistí váš kód. Ale je také nutno dodat, že testy se nesmí psát jen z formality, ale musí být správně cílené na problémové situace.

Pokud je test napsaný tak, že "vidí" do implementace, mluvíme o white-box testování. Pokud je testovaný kód pro test černou skříňkou, hovoříme o black-box testování. Oba přístupy mají své opodstatnění. Pomocí "bílého" testování snáze pokryjete všechny cesty provádění. "Černé" testování zase zosobňuje naivní přístup klientské strany, který může přinést nečekané způsoby volání.

Vyšší formou testů jsou integrační a systémové testy. Validují interakci více objektů a funkcionalitu větších celků. Pamatujte, že požadavky na kvalitu kódu testů jsou stejně přísná jako na vlastní testovaný (produkční) kód. Kód testů budete udržovat stejně dlouho jako produkční kód. Zjednodušeně shrnuto, testy by měly běžet krátkou dobu, na libovolném "kompatibilním" počítači, měly by po sobě uklidit a neměly by být na sobě vzájemně závislé.

Snažte se co nejvíce testů zautomatizovat, určitě se vám práce vyplatí. Bez automatizace nejsou některé typy testů vůbec proveditelné. Těžko se shání několik stovek uživatelů na zátěžové testy, kteří by v jednom čase začali používat a zatěžovat vaši aplikaci :)

Zautomatizovat se dá i interakce s uživatelským rozhraním vaší aplikace. Volba nástrojů závisí na technologii prezentační vrstvy.

Ruční testování

Automatizace buildů

Prototypování

Prototyp je funkčně zjednodušený základ (předobraz, demoverze) vyvíjeného systému. Měl by vzniknout relativně rychle a slouží k průběžné revizi požadavků. Prototyp můžete předvést

investorovi a získat zpětnou vazbu. Prototyp je vyvíjen v cyklech. V každém cyklu jsou zapracovány získané připomínky.

Revize výstupů v předimplementačních fázích

Motivace lidí

Metriky

[http://wiki.zvesela.cz/index.php/Zp%C5%AFsoby_prevence_chyb_v_software%2C_oponentury.]

Softwarové metriky:

- Cyclomatic Complexity = složitost části programu (např. metody) co do množství větvení a cyklů. Tyto programové konstrukce zvyšují počet možných cest provádění programu. Vysoké číslo indikuje komplikovaně napsané složité metody, které je problematické pokrývat testy. Řešením je metodu rozbít do více menších metod.
- Line Count (SLOC) vyjadřuje množství řádků kódu v metodách. Dlouhé metody jsou nepřehledné a dělají zřejmě více než jednu věc (pro danou úroveň abstrakce). Použitelnost takových metod je problematická, stejně jako pokrytí testy. Řešením je opět refaktorizace do více metod.
- Objektové metriky jako Weighted Method Count (celková složitost metod ve třídě), Depth of Inheritance Tree (počet předků třídy) a Coupling Between Objects (provázanost mezi objekty) mohou indikovat problémy v objektovém návrhu.

Z přednášek aswi 06-kvalita.pdf:

Formální verifikace

- Matematické důkazy správnosti
  - Návrhu
  - Implementace
- Model checking
  - Formální model systému - Petriho sítě, algebry (CSP) (a-priory nebo získaný analýzou kódu)
  - Model checker -> deadlock-free, liveliness, …
  - Soulad s implementací

Statistické kontroly

- Základ: metriky
  - Indikátor někde je něco špatně
- Stanovení očekávaných/správných hodnot
- Průběžné monitorování
- Korekce procesu, komponent při odchylkách

Z přednášek aswi 05-metriky.pdf:

Proč měřit

- Kvantitativní ukazatele
  - Pomáhají najít slabiny -> zlepšení kvality, přesnosti, efektivity…
  - Dávají přehled a kontrolu nad projektem-produktem - plán, kvalita, splnění požadavků…
  - Kalibrují odhady
- Výhody
  - Přesnost a dokazatelnost
  - Možnost statistik a vizuální prezentace

Metrika, měření

- Metrika = měřitelná charakteristika nějaké entity
- Je získána na základě dat (primitivních metrik)
- Měřený objekt - entita: produkt nebo proces
- Metriky samy o sobě "k ničemu" -> měření
- Plán měření - pro projekt
  - Co měřit, proč měřit, jak měřit
  - Jak s daty pracovat
- Organizational focus - záměr zlepšovat kvalitu

    - Vede k potřebě mít informace

Metriky produktu

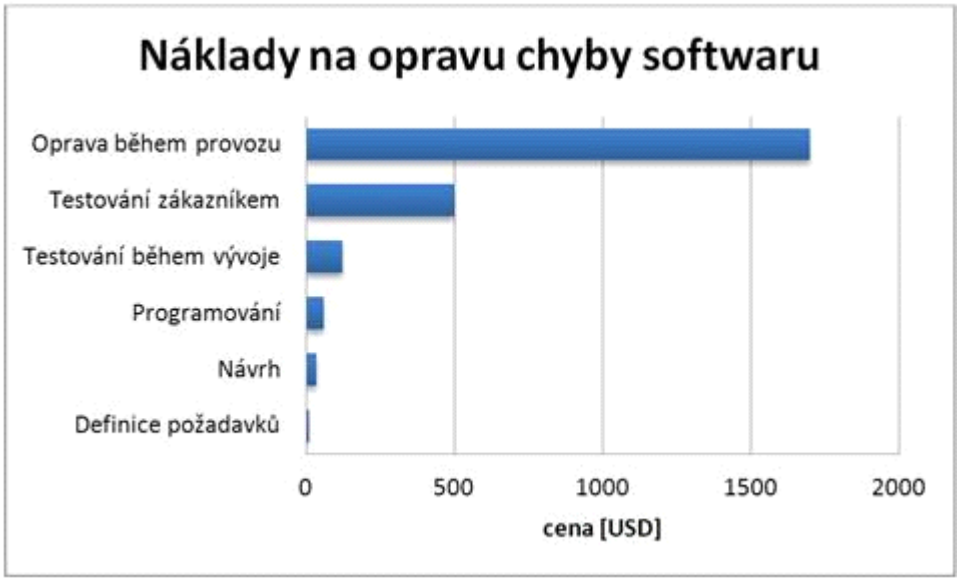
- Složitost, přehlednost
    - McCabe cyclomatic complexity
    - Fan-in / fan-out (afferent / efferent coupling) => stabilita
    - Weighted method per class
    - Lack of cohesion
- Velikost
    - Počet UC, funkčních bodů
    - Možná někdy případně i také LOC
        - SLOC, DSLOC, CBLOC, TLOC
- Metriky produktu (2)
    - Spolehlivost
        - MTBF = MTTF + MTTR
        - Dostupnost je pak (MTTF / MTBF) * 100
    - Kvalita (nepřímé metriky)
        - Pokrytí testy - kódu, požadavků
        - Charakteristiky defektů - hustota, výskyt
        - Kvalita zdrojového kódu
    - Nástroje
        - PMD, FindBugs, …
        - IDE pluginy
- **Projektové a procesní metriky**
    - Postup
        - Project velocity / burndown
        - Jitter - change requesty a jejich zpracování, staff turnover, změny postupu a plánu
    - Kvalita
        - Breakage = průměrná váha změnu LOC / CR (lines of code / change request)
        - Pracnost celkem, přepočtená na Change Requesty
        - Defect discovery rate, defect removal (zpracování, trendy)
- Jak měřit
    - **Top-down**
        - Definovat cíl měření
        - Zvolit metrik
    - **Goal-Question-Metric**
    - **Bottom-Up**
        - Jaké metriky?
- **Goal-Question-Metric**
    - Přístup k definování metrik
    - Rámec pro systém zaměřený na konkrétní problémy
    - **Goal** = problém + cíl měřícího programu
    - **Question** = měřené objekty a způsob měření
    - **Metric** = konkretizují získávaná data
- Nástroje pro měření
    - Spreadsheet, kalendář
    - Bugtracker
    - Statsvn
    - Junit a corbetura
    - Databáze
- Řízení měření
    - Plán měření
        - RUP template
        - GQM přístup
        - Definice metrik, jejich význam a zpracování
        - Způsob získání dat
    - Sledování projektu a produktu

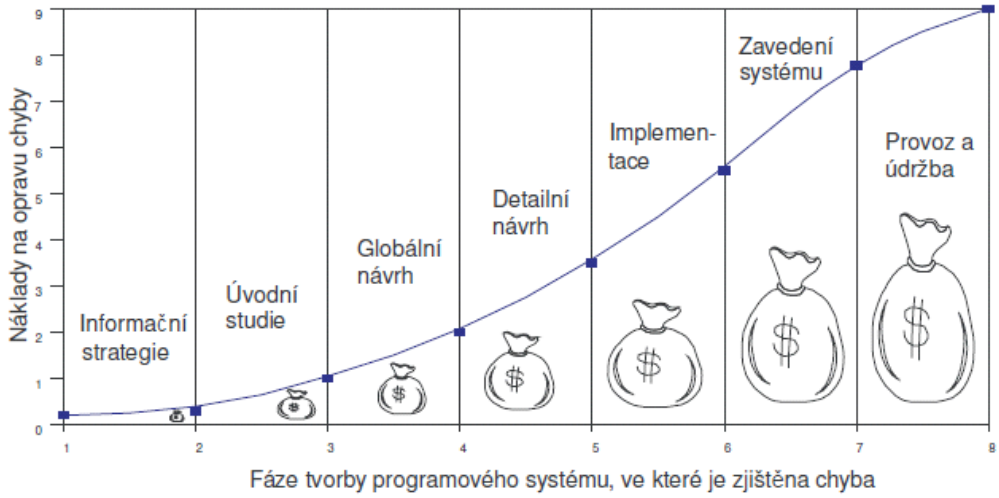
        - Automatické získávání a vyhodnocování
        - Sledování (management)
        - Korektivní akce

Oponentura

Technická oponentura

- Technická oponentura
    - Též Faganovská inspekce
    - Skupinová technika, cílem je odhalit chyby v návrhu/kódu/dokumentu, sledování standardů, vzdělávání
    - Ne: dělat potíže autorovi (neúčast vedení), hledat nápravu chyb
- Role ve skupině
    - Moderátor - řídí diskuzi
    - Průvodce - předkládá dílo
    - Autor - vysvětluje nejasnosti
    - Zapisovatel - zaznamenává nalezené problémy
    - Oponenti - hledají chyby, obvykle podle seznamů otázek
- Postup
    - Příprava
        - Distribuce díla (moderátor), projití a hledání problémů (oponenti) - několik dní předem, cca 2 hodiny práce
    - Schůzka
        - Sekvenční procházení díla (průvodce či moderátor)
        - Vznášení připomínek
        - Zapisování nálezů (chyb a otevřených otázek)
            - Nejvýše 2 hodiny
            - Nepřipouštět dlouhé diskuze, řešení chyb (moderátor)
            - Možná následná schůzka pro vyřešení otázek
    - Závěry
        - Verdikt: v pořádku/drobné chyby/nutné přepracování/nová oponentura
        - Autor odstraní chyby dle nálezů, moderátor zkontroluje
            - **Dokument: Nálezy oponentury**
- Zhodnocení technické oponentury
    - Použitelné ve všech fázích životního cyklu
    - Velmi dobrá detekce chyb (až 75%)
    - Výsledkem jsou nižší náklady na vývoj a vyšší produktivita
    - Nároky
        - Náročné na čas
        - Je třeba zkušenost





Obrázek A 10 Náklady na změny v projektu

From <https://d.docs.live.net/e3534876709763a3/Dokumenty/ZCU/Statnice/Statnice.docx>