Návrh řízený testem, refaktorizace, vlastnictví a správa zdrojového kódu v týmu

Z FITwiki

(motivace, podstata, souvislosti, přínos pro zajištění kvality software, repozitáře a větvení zdrojového kódu)

Obsah

Návrh řízený testem

> Motivace: Testy se skutečně napíší, ujasnění si detailního rozhraní a chování, prokazatelná a opakovatelná verfikace, důvěra v provádění změn.
> Podstata: Nejprve se napíše test a teprve potom se implementuje to, co se testuje. (ne "navrhuje"? vývoj vs návrh řízený testy?)
> Souvislosti: Kód pokrytý testy lze bezpečně refaktorizovat.

Klasický návrh je u agilního vývoje nahrazen kombinací přejímacích testů, refaktorizace a vývojem řízeným testem. Je zřejmé, že významnou roli zde hrají přejímací testy (acceptance test). Již víme, že jsou to validační testy, které ověřují, že zákazník dostává to, co chtěl. Programátor tedy vytváří tento test předtím, než daný příběh implementuje. Tento proces – nejdříve test, potom implementace – se označuje jako vývoj řízený testem (test-driven development) nebo také programování s úmyslem (intentional programming). Přejímacím testem totiž vývojář vyjadřuje svůj úmysl.

Refaktorizace

> Motivace: Zlepšení návrhu, lepší srozumitelnost kódu, snazší hledání chyb.
> Podstata: Vylepšení struktury systému nebo jeho části bez změny jeho funkčnosti.
> Souvislosti: Typicky se používá ve spojení s testy - testy musejí úspěšně procházet na začátku a na konci refaktorizace.

Disciplinovaná metoda přepisu nebo restrukturalizace existujícího kódu beze změny vnějšího chování, aplikace malých transformací kombinovaných s opakovaným spouštěním testů. Vede ke snadnějšímu pochopení kódu a usnadnění dalších úprav.

- Odstranění duplicitního kódu
- Zvýšení srozumitelnosti

- Zkrácení dlouhých metod
- Odstranění „natvrdo“ kódovaných literálových konstant
- Tzv. „pachy v kódu“ (code smells):
  - duplicitní kód
  - rozsáhlé metody
  - třída s mnoha atributy
  - nápadně podobné podtřídy
  - silné provázání (coupling) mnoha objektů

Kdy refaktorujeme

- Refaktorizujeme místo třetího opakování („Poprvé to prostě uděláš, podruhé se rozmýšlíš kvůli duplicitě, ale uděláš to, ale potřetí bys měl refaktorizovat.“)
- Refaktorizujte před přidáním funkcionality (nejčastější případ refaktorizace).
- Refaktorizujte při opravách chyb (lepší pochopení kódu) a při revizi kódu.

Vlastnictví a správa zdrojového kódu v týmu

- **žádné vlastnictví kódu** (každý může měnit cokoli)
- **silné vlastnictví kódu** (každá část kódu má vlastníka, jen ten do ní může zasahovat)
- **slabé vlastnictví kódu** (každý může měnit ale jen se souhlasem/informováním vlastníka)
- **společné/kolektivní vlastnictví** (každý by měl vylepšovat jakýkoli kód, kdykoli je k tomu příležitost)

Kolektivní vlastnictví

Kolektivní vlastnictví znamená, že narozdíl od běžné praxe, kdy vlastníkem kódu a tedy i osobou zodpovědnou za něj je vždy ten, kdo ho vytvořil, zde kód vlastní všichni členové týmu. Každý může v případě potřeby kód vytvořený jiným modifikovat, ale také všichni členové týmu přijímají odpovědnost za celý systém.

Klasická integrace a nasazení jsou u agilního vývoje nahrazeny spojitou integrací a krátkými iteracemi. Každý pár programátorů může v libovolném okamžiku testovat a integrovat jejich kód. Současně ale může více dvojic pracovat se stejným kódem. Mohou tak vznikat konflikty, které je třeba řešit. Tento způsob vývoje také vyžaduje podporu správy verzí, která umožní případné konflikty detekovat.

Větvení zdrojového kódu

- **Sdílený kód s lineární historií**
  - Historie sdíleného kódu je vidět jako přímka, bez větvení. (nikdy veřejně neexistovalo více variant sdíleného kódu)
  - Nevýhody:
    - Protože existuje jen jedna varianta sdíleného kódu, musí být vždy perfektní.
    - Rozpracované kopie kódu se nesdílí - spolupráci si vývojáři musí řešit jinak.

- **Sdílený kód s větvenou historií**
  - Historie sdíleného kódu je vidět jako acyklický graf. (veřejně existuje více větví kódu)
  - Každá větev může mít svůj účel:
    - Stabilní/produkční kód (master)
    - Aktuálně vyvíjený kód (devel)
    - Kód v přípravě na vydání (release - na cestě z devel do master)
    - Větev pro implementaci rysu (feature), opravu chyby (bug), čištění, atp.

- Vývojáři mohou lépe spolupracovat na sdíleném kódu
- Vlastníci kódu mohou kontrolovat přijetí změn

Kategorie: Informační systémy (kategorie)

- Stránka byla naposledy editována 5. 6. 2016 v 18:11.