Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	ZS 2017/2018
Autoři	Thanh Cong Nguzen, ngut19 Manh Hoang Phí, phih00
Téma	Jak prodat refactoring? Případova studie Nordea Bank AB
Datum odevzdání	17.12.2017

1 Úvod

Refactoring je v softwarovém vývoji velmi důležitá a mnohdy zanedbávaná činnost. Cílem této semestrální práce je v první části uvést čtenáře do tématu refactoring. Vysvětlit mu co znamená, kdy se využívá refactoring a jeho přínosy v programováni. Druhá část je věnovaná případové studii společnosti Nordea Bank AB, která v rámci vlastního projektu “akce-30” aplikovala refactoring na svůj bankovnický systém. Refactoring obecně negativně přijímán vedoucími projektu, protože může trvat velmi dlouho, brzdit projekt a nepřinášet na první pohled žádný viditelný výsledek a zároveň i samotnými vývojáři, kteří by radši pracovali na vývoji nové funkcionality. V této případové studii je popsáno, jakým způsobem se řešil refactoring, aby byl dostatečně “prodán” ve společnosti.

2 Co je refactoring?

Refactoring, česky refaktorování je proces změny kódu takovým způsobem, aniž by se změnila jeho funkcionalita za účelem zlepšení struktury kódu, tzn. je to o tom laicky řečeno udělat kód hezčím a přehlednějším bez jeho změny. Při refaktorování se v kódu provádí velmi malé a jednoduché změny, avšak výsledný efekt je velký v podobě čistšího a čitelnějšího kódu, z toho pak vyplývá, že pro programátora je snazší kód udržovat či rozšiřovat. Refaktorování tak obecně pomáhá zvýšit kvalitu kódu a snížit jeho chybovost. (Shvets 2014)

2.1 Proč refaktorovat kód?

Jaký je důvod k refaktorování? Jak již bylo tedy výše zmíněno, programátoři se refaktorováním snaží udělat kód přehlednějším, čistším a kvalitnějším. Jak takový přehledný a čistší kód ale vypadá?

Příklady vlastností přehledného a čistého kódu:

- Kód by měl být čitelný i programátorům, kteří čtou kód po někom jiném.
- Kód by neměl obsahovat duplicity.

- Kód by měl obsahovat jen nezbytný počet tříd a třídy by neměly být moc dlouhé (samé platí pro metody a jiné části kódu).
- Kód by měl projít všemi připravenými testy.
- Přehledný a čistý kód je snazší a méně nákladný na údržbu. (Shvets 2014)

Každý programátor se od začátku snaží, aby jeho kód měl výše zmíněné vlastnosti. Není žádný programátor, který by účelně psal „nečistý“ kód, avšak vždycky existuje nějaká okolnost, která zapříčiní, že programátor napíše „nečistý“ kód. Těmi okolnostmi mohu být např.:

- Nátlak byznysu – někdy kvůli obchodnímu nátlaku je programátor nucen dokončit projekt, aniž by byl řádně dokončen se všemi plánovanými funkcemi. Následně pak musí vydávat různé opravné patche, apod.
- Nedostatek dokumentace
- Nedostatek testů kódu
- Nedostatek komunikace mezi členy týmu
- Neschopnost programátora – toto je příklad, kdy programátor prostě neumí psát přehledně kód, nedodržuje různé konvence programování, apod.

2.2 Kdy refaktorovat?

Existuje pravidlo tří (The Rule of Three) od Dona Roberts, ve kterém se uvádí, že zaprvé pokud musíte něco naprogramovat, tak to naprogramujete. Za druhé pokud musíte naprogramovat něco podobného, naprogramujete to a vznikne kódu duplicita. Za třetí opět pokud musíte naprogramovat něco podobného, refaktorujete.

„The first time you do something, you just do it. The second time you do something similar, you wince at the duplication, but you do the duplicate thing anyway. The third time you do something similar, you refactor.“ Don Roberts

Pravidlo tří tedy říká něco ve smyslu, kdy má být replikovaný kód nahrazen novým postupem. Uvádí, že kód může být kopírován jednou, po druhé a dál by měl programátor refaktorovat kód. (Nadler 2014)

Refaktorovat by se mělo také v případě, kdy chceme přidat novou funkci kódu, tzn. nejdříve refaktorovat kód a tím ho zpřehlednit a vyčistit a až poté přidávat/vyvíjet novou funkcionalitu.

Dále pokud je potřeba opravit bugy v kódu, které jsou běžně obtížné najít v nepřehledném kódu, je při této okolnosti také dobré kód refaktorovat.

V poslední řadě je dobré kód refaktorovat během revize kódu před tím, než je funkcionalita kódu vypuštěna do produkce. (Shvets 2014)

2.3 Jak refaktorovat?

Martin Fowler ve své knize Refactoring: Zlepšení existujícího kódu uvádí 22 takzvaných „pachů“ v kódu (code smells), které jsou zvěstí toho, že bychom měli začít kód refaktorovat. Ve své publikaci také uvádí techniky/postupy jak refaktorovat určitý „pach“ kódu. (Fowler 2003)

Nejčastější příklady „pachů“ kódu:

- Obrovské jednotlivé části kódu (dlouhé metody, velké třídy, dlouhý seznam parametrů, apod.)
    - Příklad dlouhé metody:

      Původní:

      ```
      double basePrice = anOrder.basePrice();
      return (basePrice > 1000)
      ```

      Upravený:

      ```
      return (anOrder.basePrice() > 1000)
      ```
      (Hájek 2013)
- Duplicity
    - Příklad duplicitního volání v kódu:

Původní:

```
if (isSpecialDeal()) {
      total = price * 0.95;
      send();}
else {
      total = price * 0.98;
      send();
}
```

Upravený:

```
if (isSpecialDeal())
      total = price * 0.95;
else
      total = price * 0.98;
send();
```

(Hájek 2013)

- Líné třídy – je to taková třída, která nedělá dost, aby se vyplatila jako samostatná třída. Měla by se odstranit nebo sloučit do jiné třídy
- Mnoho komentářů – tento pach signalizuje, že kód je špatný a nepřehledný. Programátor má potřebu vše komentovat v případě, že by se sám v kódu posléze nevyznal.
- Datové shluky, „mrtvé“ kódy (již nepoužívané metody, třídy nebo parametry), atd. (Fowler 2003)

Každodenní a strategické refaktorování

Pro následující kapitolu, která bude pojednávat o případu společnosti Nordea Bank AB, se refaktorování rozděluje na dva druhy: každodenní refaktorování a strategický refaktorování.

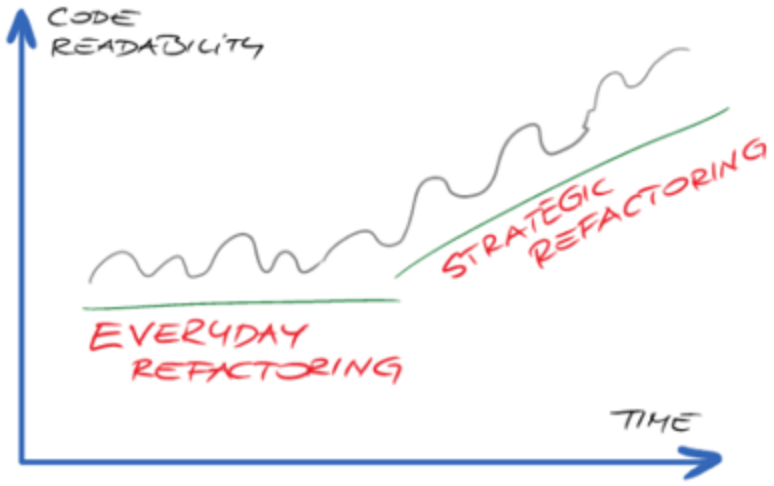
Každodenní refaktorování

Každodenní refaktorování dělá každý programátor a je zvládnutelný během několika minut denně. Programátoři během psaní se neustále snaží vylepšovat svůj kód na každodenní bázi,

jelikož je to v rámci programming practice a není žádná omluva pro programátora tedy, aby svůj kód neustále nevylepšoval.

Strategický refaktorování

Strategický refaktorování je dlouhodobý týmově plánovaný refaktorování, vyžadující pečlivé plánování iterací refaktorování. Jedná se o riskantní dlouhodobou činnost, která vyžaduje neustále intenzivní testování a je náročná jak věcně tak časově. Každý tým by si měl před strategickým refaktorováním nejdříve zkontrolovat, zda strategický refaktorování má dostatečnou přidanou hodnotu pro tým. (Sieraczkiewicz 2014)



Obrázek 1: Everyday and Strategic Refactoring (infoq.com 2014)

3 Případová studie Nordea Bank AB

V následující kapitole je popsáno, jak byl řešen refactoring bankovnického systému v Nordea Bank AB (Bartyzel 2017).

3.1 Nordea Bank AB

Nordea Bank AB je finanční skupina operující zejména v severní Evropě. Tato semestrální práce je zaměřena na nyní již neexistující pobočku Nordea Bank AB - Nordea IT Poland, která mimo jiné vyvíjela bankovnický systém.

3.2 Zasazení do kontextu

Nordea IT Poland přestala vyvíjet bankovnický systém pro polský trh. Systém se používal jen v baltských státech. Následkem tohoto rozhodnutí vzešly tyto problémy:

- Klienti čekají na požadovanou funkcionalitu dlouhou dobu (průměrně měsíce).
- Některé požadavky nelze realizovat kvůli technickým omezením systému.
- Některé požadavky si klienti nechali realizovat od místních poskytovatelů. Tato řešení potom musela Nordea IT Poland integrovat a spravovat ve vlastním systému.

Vedení se rozhodlo tyto problémy řešit a začátkem ledna 2015 byla zahájena iniciativa: *Zkrátit čas vývoje nové funkcionality na 30 dní.* To by se dalo formulovat jako: Vývoj nové funkcionality trvá moc dlouho, pojďme udělat něco, aby vývoj každé nové funkcionality trval maximální 30 dní. Iniciativa má kódové označení *akce-30*. Iniciativy se kromě zaměstnanců Nordea IT Poland zúčastnil i Michael Bartyzel, konzultant, který pomáhá vývojářům a týmům v efektivnější práci.

3.3 Zdroje problémů

- Systém byl původně vyvinut před deseti lety s použitím frameworku pro vytváření webových aplikací, Struts 1.2. Ve světě technologií je toto řešení archaické.
- Systém byl vyvíjen mnoha vývojáři z několika zemích. Zdrojový kód pak nebyl jednotný. Další problém, který se přímo váže k tomuto je to, že některé požadavky, které byly pro jednotlivé země téměr stejné se vyvíjely zvlášť. Jednotlivá řešení se ve finálním kódu poté opodmínkovala. Údržba a vývoj nové funkcionality v takovém kódu pak nebyla vůbec jednoduchá. Kód měl také kvůli tomu způsobu řešení relativně vysokou cyklomatickou složitost.

Zdroje problémů nebyly jen technické, ale i organizační.

- Vývojáři pracovali na mnoha projektech. V extrémních případech mohlo přepnutí práce z jednoho projektu na druhý trvat 30 až 60 minut.
- Větší počet projektů znamená i větší počet schůzek, které byly mnohdy neefektivní.

3.4 Omezení akce-30

Iniciativa akce-30 měla od vedení jasná omezení.

- Akce-30 nesmí brzdit ostatní projekty.
- Na začátku se smí zúčastnit pouze pět dobrovolných vývojářů.
- Každý z vývojářů se smí věnovat akci-30 pouze jeden den týdně.
- Co nejdříve prokázat obchodní přínos.
- Jelikož je akce-30 další přítěží pro vývojáře, práce na projektu by měla být dostatečně zajímavá.

3.5 Řešení

3.5.1 Způsob práce

Se všemi omezeními a náturou projektu, bylo rozhodnuto, že se na akci-30 bude pracovat jako ve scrumovém týmu. Nejedná se ale o čistý scrum, ale vlastní poupravený scrum. V týmu takovému scrumu říkali a'la scrum. Hlavní body a'la scrumu zní následovně:

- Celý tým pracuje na akci-30 pouze ve čtvrtky.
- Jednotlivé iterace, a'la sprinty, trvají jeden měsíc (čtyři čtvrtky).
- A'la sprint je zakončen schůzkou s ohlédnutím za poslední sprint a plánováním na další sprint. Této schůzky by se ideálně mělo zúčastnit co nejvíce stakeholderů.
- Na jednotlivých úkolech se pracuje v párech.
- Jednotlivé úkoly správně rozdělené, aby trvaly maximálně 4 hodiny.
- Vyvarovat se neproduktivním schůzkám.
- Každý a'la sprint má jasný a měřitelný cíl.

3.5.2 A'la sprint 1

Z důvodu složitosti a nestability akce-30 bylo nutné začít s projektem pomalu a zlehka. Neboť sebemenší problém v jiných projektech mohl akci-30 ohrozit a ihned ji ukončit nebo naopak, samotná akce-30 mohla brzdit ostatní projekty. Tým si pro úvodní sprint zvolil úkoly zejména organizačního charakteru, kde se řešilo, jak a co se bude refaktorovat.

Ohlédnutí za sprintem

Tým byl s výsledkem sprintu spokojen. Cíle byly splněny a dokonce se zvládlo splnit více než bylo naplánováno. Vývojáři pracovali totiž na akci-30 i v jiné dny než čtvrtky, avšak ne na úkor jiných projektů. Potvrdilo se to, že když je projekt dostatečně zajímavý, zaměstnanci dokáží pracovat efektivně s časem a na projekt si čas najdou.

Také se ukázala důležitost manažera. Skvělý tým si dokáže poradit s technickými úkoly, ale u organizačních úkolů, kterých v tomto sprintu bylo mnoho, je pomoc manažera velmi vítána. S pomocí manažera dokázal tým redukovat počet potřebných schůzek a lépe komunikovat s ostatními odděleními.

3.5.3 A'la sprint 2

Plánem a'la sprintu 2 bylo začít ukazovat obchodní přínos. Definicí refactoringu změna kódu bez změny funkcionality, proto je velmi těžké jej prodat. Tým se rozhodl, že nebude dělat čistý refactoring a prokáže obchodní přínos následovně:

- Prokázání přínosu pro organizaci - Odstranění přebytečného kódu a tím zkrácení času pro vývoj.
- Prokázání obchodního přínosu - Některé úkoly zabývající se refactoringem jsou spojeny s vylepšením/přidáním nové funkcionality.

3.5.3.1 Prokázání přínosu pro organizaci

Systém se již nepoužíval na polském trhu, ale polská část kódu stále byla v systému. Jak již bylo zmíněno výše, každý oddělený trh měl vlastní kód oddělený podmínkami. V tomto sprintu se týmu podařilo odstranit část tohoto přebytečného kódu a změřit výsledky. Cyklomatická složitost se snížila o 15%. Bylo možné i smazat přebytečné testy, kterých bylo 29.

3.5.3.2 Prokázání obchodního přínosu

Pro prokázání obchodního přínosu byla vybrána funkcionalita, která byla dříve vyžadována, ale zamítnuta kvůli technickým omezením frameworku Struts 1.2. Jako náhrada byl vybrán AngularJS. Struts 1.2 byl nahrazen jen v části systému na frontendu. Tím bylo umožněno pracovat s novým AngularJS v jedné části a zároveň refaktorovat část na backendu.

Manažer dokázal na schůzku ohlédnutí za sprintem přizvat co největší množství ostatních projektových manažerů, stakeholderů, kterým tyto výsledky byly předvedeny.

3.5.4 Následující sprinty

Projektoví manažeři a stakeholdeři byli spokojeni s iniciativou a žádali další podobné nové funkcionality. Následující sprinty byly vždy plánovány tak, aby byly úkoly refactoringu spojené s přidáním/vylepšením funkcionality.

Dalším pozitivem bylo to, že projekt byl dostatečně zajímavý a během následujících sprintů se několik vývojářů z ostatních týmů dobrovolně přidalo.

3.5.5 Konec projektu

I když se akci-30 dařilo, nebyla dokončena. Vedení totiž časem rozhodlo, že s odchodem z polského trhu také kompletně ukončí vývoj systému v Polsku. Nicméně přístup a dobrá práce týmu, který pracoval na akci-30 byl vedením zaznamenán. Tým dostal přiděleno několik důležitých projektů od zahraničních partnerů.

4 Závěr

Případová studie Nordea Bank AB nám názorně popsala, jak probíhal refactoring v rámci akce-30 bankovnického systému. I když je refactoring zejména technická záležitost, studie nezacházela přímo do technických detailů, ale spíše se soustředila na procesní a organizační záležitosti. Znázornila nám způsob a metodu práce, podle které se řídil tým, který refactoring prováděl. Studie vypíchnula důležitost práce v týmu, řádného zpracování úkolů, jasné a měřitelné cíle, důležitost role manažera a nedělat pouze čistý refactoring. Tyto body vedly k hladkému průběhu projektu a úspěšnému “prodání” v organizaci.

5 Seznam zdrojů

BARTYZEL, Michał a Łukasz KORCZYŃSKI. How to Sell Refactoring? The Case of Nordea Bank AB. *InfoQ* [online]. 2017 [cit. 2017-10-23]. Dostupné z: https://www.infoq.com/articles/sell-refactoring-nordea-bank

FOWLER, Martin a Kent. BECK. *Refaktoring: zlepšení existujícího kódu*. Praha: Grada, 2003. Moderní programování. ISBN 80-247-0299-1.

HÁJEK, Lukáš. Refaktorování: Code refactoring. *ČVUT Praha - Fakulta jaderná a fyzikálně inženýrská*[online]. ČVUT Praha, 2013 [cit. 2017-12-17]. Dostupné z: http://kfe.fjfi.cvut.cz/~hajeklu2/files/OOP/Hajek-Refactoring.pdf

NADLER, Bob. The Rule of Three. *Bob Nadler* [online]. 2014 [cit. 2017-12-17]. Dostupné z: http://bobnadler.com/articles/2014/01/26/the-rule-of-three.html

SHVETS, Alexander. What is refactoring. *Refactoring Guru* [online]. ©2014-2017 [cit. 2017-12-17]. Dostupné z: https://refactoring.guru/refactoring/what-is-refactoring

SIERACZKIEWICZ, Mariusz. Natural Course of Refactoring – a Refactoring Workflow. *InfoQ* [online]. 2014 [cit. 2017-12-17]. Dostupné z: https://www.infoq.com/articles/natural-course-refactoring