Vysoká škola ekonomická v Praze
Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program : Aplikovaná informatika
Obor: Informační systémy a technologie

Lean software development v testovacím týmu

DIPLOMOVÁ PRÁCE

Student : Bc. Ondřej Chodura
Vedoucí : Ing. Luboš Pavlíček
Oponent : Ing. Dušan Chlapek, Ph.D.

2013

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 1. 12. 2012 . . . . . .

Bc. Ondřej Chodura

Abstrakt

Práce se věnuje pozici testování software podle přístupu Lean software development a jeho aplikací v testovacím týmu. První část práce představuje koncept Lean myšlení a jeho základní principy a dále se práce zaměřuje na samotnou pozici testování podle Lean software development.

Tato práce se nejdříve zabývá agilním přístupem k vývoji a pozici testování v Lean software development. Koncept Lean software development definuje několik principů, podle kterých by se měly testovací týmy řídit. Tyto principy a jejich obecná aplikace v testovacích týmech bude následně vysvětlena.

Druhá část práce je věnována aplikaci Lean software development ve firmě. Pracuji na testerské pozici a znám již procesy během vývoje a testování, proto je jedním z cílů této práce aplikovat tento přístup v týmu, ve kterém s kolegy zajišťujeme testování softwaru.

Nejdříve je nastíněna výchozí situace ve firmě. Uvedeny jsou použité techniky a druhy používaných nástrojů a postupů a také nevýhody a problémy, se kterými se v průběhu testování dle dřívějších standardů potkáváme. Poté následuje text zabývající se aplikací principů Lean software development v našem testovacím týmu a zlepšení a výhody, které přináší.

Klíčová slova

Lean, Lean software development, testování, kvalita software

Abstract

This diploma thesis describes the position of software testing according to Lean software development approach and its appliacation in testing team. At first. Lean concept is introduced with its basic principles, then the thesis focuses on the position of testing in the Lean software development itself.

At the beginning this thesis deals with agile approach to development and software testing position in Lean software development. This concept defines several principles, which should be followed by testing teams. These principles and its general application in testing teams are subsequently explained.

The second part is devoted to Lean software development in a real company. I am working as a tester and I already know the development and testing processes – that's why one of my goals is to apply this approach to our testing team.

Firstly the default situation in our company is introduced. Used technics and types of used testing tools are listed together with procedures and also the disadvantages and troubles, which occurred during the original testing process. Then the following parts are devoted to Lean software development principles application in our testing team followed with description of improvements and advantages delivered by the principles.

Keywords

Lean, Lean software development, software testing, software quality

Obsah

1 Úvod

Koncept Lean myšlení vycházející z nápadu Lean production je jedním z mnoha pohledů na přístup k výrobě, vývoji nebo testování produktu. Tento nový způsob práce se ujal v IT prostředí a vznikla myšlenka aplikovat principy Lean pro softwarové společnosti pod názvem Lean software development. V této práci se zaměřím na zavedení Lean software development v testovacím týmu.

V první části práce představím testovací tým, který je součástí softwarového projektu. Uvedu rozdíly oproti tradičnímu přístupu, nastíním situaci v průběhu testování a představím druhy a techniky testování používané v Lean software development.

Poté přejdu k představení samotného konceptu Lean myšlení a jeho aplikace pro softwarové společnosti. Základem Lean software development jsou principy, které by měla každá softwarová společnosti aplikovat. U každého principu se pozastavím a popíšu, jak se konkrétní cíl může obecně aplikovat pro testovací tým.

Další kapitola se týká samotné aplikace těchto principů pro testovací tým ve společnosti, kde pracuji. Nejdříve upřesním výchozí situaci, definuji používané technologie, nástroje, techniky a také postupy při testování softwarových aplikací. Samotné použití principů popisuji tak, že každý z principů definuji, popisuji, jak ho náš testovací tým aplikoval na aktuální situaci, a co se díky tomuto principu zlepšilo nebo zhoršilo.

Na závěr této práce jsem vytvořil dotazník, který jsem rozeslal členům celého týmu v naší společnosti, aby se mohli vyjádřit k testovacímu procesu po přijetí Lean software development. Vyhodnocení dotazníku obsahuje poslední kapitola práce.

Jako cíle této diplomové práce jsem si stanovil představit koncept Lean Software Development a jeho využití pro oblast testování softwaru a popsat jej na základě zkušeností s aplikací v reaálné firmě. Vedlejším cílem práce je ověření hypotézy, že použití Lean Software Development zvyšuje efektivitu testovacích týmů, a to formou srovnání funkčnosti týmu před a po zavedení Lean Software Development a dotazníkovým šetřením mezi pracovníky.

2 Rešerše

Kapitola shrnuje existující nejdůležitější zdroje, pojednávající o konceptu LeanSD a jemu příbuzných tématech formou komentované rešerše.

Výchozím zdrojem pro mou diplomovou práci byla kniha (1) autorů Toma a Mary Poppendieck, kteří se mnoha knižními tituly zasloužili o rozvoj Lean software development a šíření tohoto konceptu mezi širokou veřejnost. Dalším zdrojem byly knižní tituly (4) a (7) rovněž od manželů Poppendieck. Kromě těchto knižních titulů tito autoři vydali několik článků, které tyto knižní tituly doplňovaly nebo obsahovaly nové informace, které nebyly v těchto knižních titulech zakomponovány. Využil jsem informace z článku (25). Manželé Poppendieck jsou profesionály týkající se konceptu LeanSD a jeho aplikace v softwarových společnostech. Díky jejich publikacím jsem pochopil Lean myšlení a historii vzniku, koncept LeanSD, proces vývoje, ale i testování v rámci tohoto konceptu.

Další velice známou zahraniční autorkou, která se zabývá výhradně testováním software a konceptem LeanSD a dalšími agilními metodikami, je Lisa Crispin, jejíž informace z knižního titulu (31) jsem také plně využil. Autorka poskytuje své prezentace na vlastních internetových stránkách, využil jsem prezentace (32) a (35). Prameny této autorky byly pro mě druhým nejdůležitějším zdrojem informací.

Kvalifikační práce, která se nejvíce dotýká tématu mé diplomové práce, je bakalářská práce Lean software development, viz (5). Tato práce se zabývá podstatou přístupu Lean software development s přhlednutím na vývoj podle tohoto přístupu obecně. Práce zahrnuje historické pozadí vývoje, definování Lean myšlení, vznik LeanSD a porovnání s agilními metodikami.

Použil jsem informace také z dalších kvalifikačních prací, jako např. (10), která se zabývá testováním software podle agilní metodiky OpenUP. Přestože se nejedná o koncept LeanSD, pohled na agilní testování software obsažen v této práci, byl pro mě také dostatečným přínosem. Dále jsem použil práci (13), která se zabývá hodnocením agilních metodik, poskytla mi přehled o ostatních agilních metodikách a pomohla mi utřídit si informace a vytvořit tak kapitoly, ve kterých jsem se snažil koncept LeanSD obecně zhodnotit.

Zahraniční kvalifikační práce, která se nejvíce přibližovala mému tématu, je práce (6) popisující jak teoretický základ LeanSD, tak zejména aplikaci LeanSD v softwarové společnosti. Přestože práce nebyla zaměřena na testování podle LeanSD, poskytla mnoho užitečných a hlavně praktických informací.

Žádná z těchto prací není vymezena pouze na testování software podle konceptu Lean software development, ale spíše poskytují teoretický základ určený k pochopení tohoto konceptu a k jeho možné aplikaci při vývoji software. Obsáhlé zahraniční knižní tituly poskytují pohled na testování software, ovšem v příliš obecném měřítku, proto jsem informace z různých pohledů v této práci zkombinoval a použil je pro úspěšnou aplikaci konceptu Lean software development pro testovací tým, ve kterém pracuji.

3 Testovací tým

Donedávna se na testování softwaru nekladl příliš velký důraz. Důkaz poskytují známé chyby, například ve firmě Intel, která měla ve svých prvních procesorech numerickou chybu, jež při náročnějších matematických počtech vracela nesprávné hodnoty. Po několika sporech se Intel rozhodl své procesory zákazníkům vyměnit. Důležité však zůstává to, že tato chyba stála firmu více než 400 milionů dolarů. Tato částka rozhodně stojí za přemýšlení a motivování firem investovat do testování vyvíjeného softwaru či hardwaru. (11)

Testovací proces nebo testovací týmy samozřejmě nikdy nemohou nalézt všechny chyby, které daný systém obsahuje, ale jedná se o důležitý proces ve vývoji softwaru, který ověřuje, jestli daný software splňuje všechny požadavky zákazníka. Testování také kontroluje, zda nejsou v softwaru žádné chyby. (10)

Samostatný testovací tým je složen z několika nebo ve větších společnostech z mnoha testerů, kde je každý tester přiřazen ke konkrétní roli v testovacím týmu. Tyto role mohou být různé a jsou vytvořeny zejména kvůli rozdělení odpovědnosti a přiřazení testerů ke konkrétním úkolům podle jejich zkušeností a znalostí.

V rámci testovacího týmu lze identifikovat tyto role:

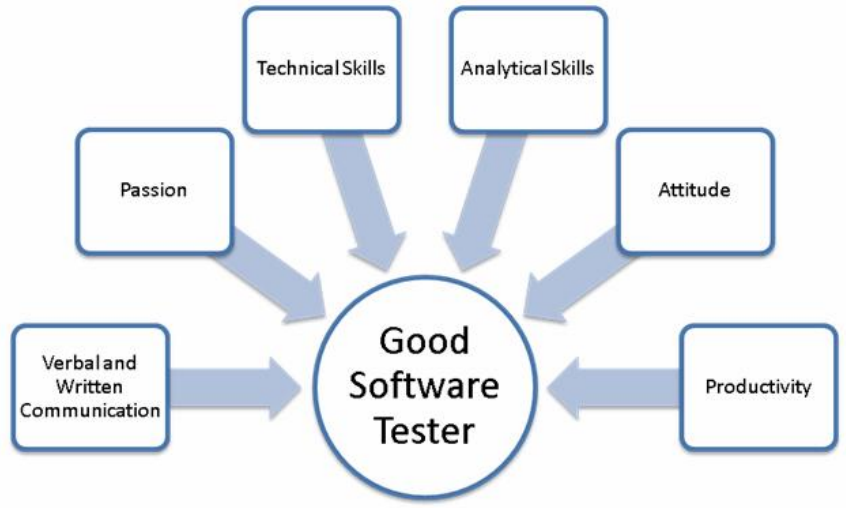
- *"Test designer – zodpovědný za přípravu testů, vytváří testovací případy a rozhoduje o tom, jaká budou testovací data.*
- *Test manager – zodpovědný za plánování, organizovaní, koordinování a reportování stavu testování.*
- *Test executor – zodpovědný za vlastní provedení testů, dokumentuje výsledky testů, kontroluje logy a hlásí defekty."* (66)

Jména rolí jsou rozdílná téměř v každé metodice vývoje software[1], avšak z vlastní zkušenosti vím, že během testování každý člen týmu občas přijme úkoly jiné role, která je za úkol odpovědná. Například test designer běžně připravuje testovací případy[2] (angl. test cases), které poté sám provede. Role v týmu splývají a zejména u menších týmů není příliš důležité členy týmu oficiálně pojmenovávat.

[1] Metodika je komplexní návod pro vývoj software s definovanými principy, procesy, praktikami, rolemi a různými technikami. (autor)

[2] Testovací případy, resp. slovně zaznamenané kroky (akce), které musí tester vykonat, aby otestoval funkčnost software. Takto provedený test patří mezi činnosti manuálního testování. (autor)

Existují charakterové a odborné požadavky, které by měl dobrý softwarový tester mít, i když některé z nich jsou jen velmi těžko prokazatelné.



1 Obrázek - Vlastnosti dobrého SW testera (67)

Podle (67) nedělají člověka dobrým testerem pouze technické dovednosti, ale jsou to také analytické schopnosti porozumět, analyzovat problém a rozdělit ho na samostatné, nejlépe i testovatelné části. Dobrý tester také dokáže své poznatky sdílet a vysvětlit ostatní členům nejen testovacího týmu.

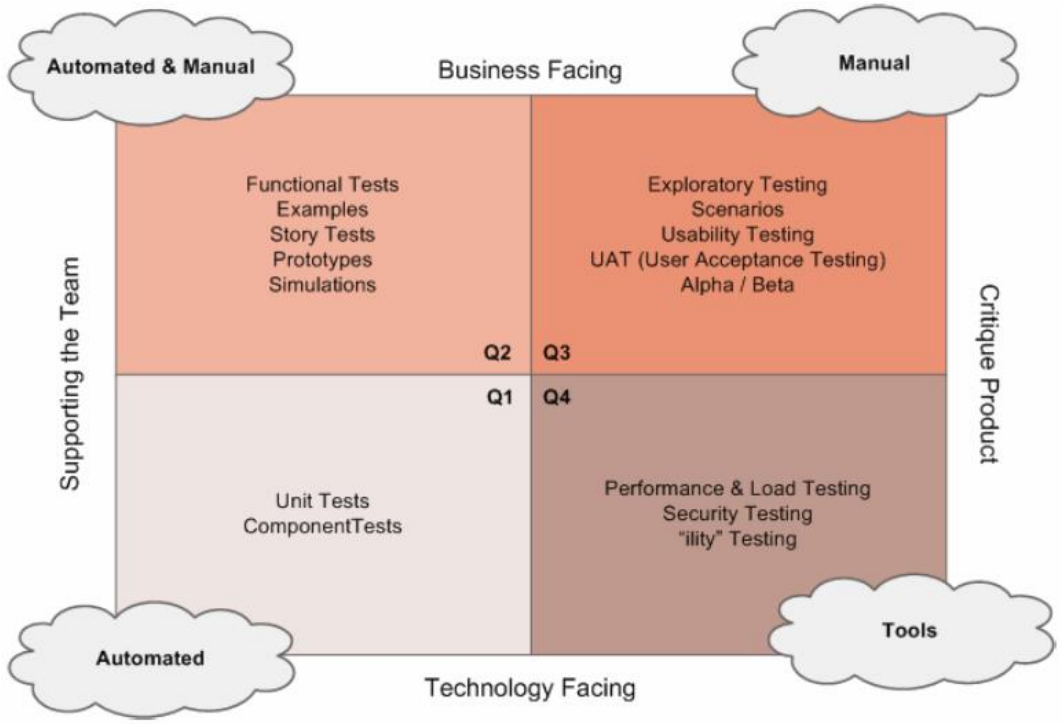
Z vlastní zkušenosti vím, že dobrý tester je také pečlivý člověk, kterému neunikne žádný detail. Myslím si, že dobrý tester se pozná také podle toho, kolikrát je vývojářem tázán ohledně vysvětlení chyb, které vývojářům reportoval. Pokud není tester schopný předat srozumitelně nabyté poznatky z procesu testování, nejedná se dle mého názoru o dobrého testera.

3.1 Úloha testovacího týmu

Úloha testovacího týmu spočívá v zajištění kvality software na takové úrovni, kdy jsou splněny požadavky zákazníka. Zjednodušeně řečeno, testování ma za úkol nalézt co nejvíce chyb, které by mohl zákazník v systému objevit. Navíc výdaje spojené s opravou problémů hotového systému jsou většinou daleko vyšší, než když je software testován průběžně. Softwarové společnosti, které počítají s testováním, mohou mnohem přesněji splnit časový plán projektu, jelikož již od počátku testují a postupně objevují a opravují chyby. Testování také pravidelně informuje management o kvalitě softwaru a podílí se při rozhodování o dalším vývoji produktu.

3.2 Průběh testování

Pro zjednušení přehledu o druzích testů používaných během testování byl podle (32) vytvořen testovací kvadrant, který určuje, které druhy testů se vytváří a k jakému účelu se používají.



2 Obrázek - Testovací kvadrant (32)

Testovací kvadrant je podle (32) rozdělen do čtyř částí – kvadrantů Q1 až Q4. Každý z nich obsahuje příklady druhů testů, které by testovací tým měl vytvářet. Testování začíná levým spodním kvadrantem Q1, který se zaměřuje na nejnižší vrstvu aplikace – zdrojový kód, a končí kvadrantem Q4, který se soustředí na kvalitu celku a obsahuje druhy výkonnostních a dalších testů celého systému.

Použití kvadrantu pomáhá zajistit splnění těchto cílů (35):

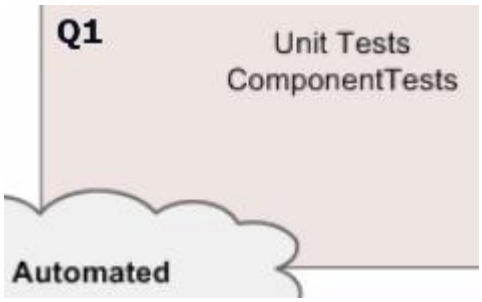
- podpora týmu;
- kritika produktu;
- zajištění business požadavků (požadavky zákazníka);
- zajištění splnění technologických požadavků.

Vertikální kvadranty stojící na levé straně Q1 a Q2 obsahují testy, které podporují tvorbu produktu. Do těchto kvadrantů patří testy, které ilustrují, vyjasňují a ujišťují tým v tom,

že se produkt chová tak, jak by se podle požadavků chovat měl. Testy slouží pro zajištění kvality kódu a produktu a také pro prevenci chyb. Naopak kvadranty na pravé straně Q3 a Q4 obsahují testy, které se zaměřují na výsledný produkt, tedy že se v produktu hledají opomenutí, chyby nebo nesprávná chování. (36)

Horizontální kvadranty ležící ve spodní části Q1 a Q4 obsahují testy, které jsou vytvářeny vývojáři nebo testery na technologické úrovni, což znamená, že zákazník je ve výsledném produktu nevidí a nemůže si z toho pohledu produkt vyzkoušet. Naopak kvadranty ležící v horní části Q2 a Q3 obsahují testy, které si zákazník vyzkoušet může. Jedná se o testy grafického uživatelského rozhraní, funkcionální testy nebo průzkumné testy. (36)

3.3 Kvadrant Q1



3 Obrázek - Kvadrant Q1 (32)

Tento kvadrant podle reprezentuje Test-Driven Development (TDD), což je technika vytváření testů a zároveň jádro agilních testovacích praktik. (37)

TDD je pokročilá technika používající automatizované unit testy, které řídí návrh software a nutí k odbourání vazeb mezi závislostmi. Unit testy (unit tests), často označované jako programátorské testy, testují zdrojový kód tak, že metodám (funkcím) tříd zdrojového kódu předávají vstupní data a kód následně vrací výstupní data. Unit testy testují funkcionalitu zdrojového kódu tím, že tato výstupní data kontrolují proti datům, které programátor očekával. (37)

Výsledkem jsou obsáhlé testovací sady automatizovaných unit testů, které mohou být spouštěny kdykoli a jsou schopné nepřetržitě poskytovat informace o tom, jestli je zdrojový kód a produkt stále funkční. Tato technika je velice zdůrazňována agilními metodikami. Typicky jsou tyto testy vytvářeny programátory, jelikož unit testy testují jejich vlastní zdrojový kód.

Cílem kvadrantu Q1 je vytvoření testů, které vývojářům pomohou navrhnout lépe jejich kód. Zároveň jim dodávají jistotu, že nasazený kód neprovede nějaké nezamýšlené změny v aplikaci. Unit testy i testy komponent jsou vytvářeny ve stejném programovacím jazyce jako je aplikace a zákazník jim pravděpodobně nebude rozumět, ale to není ani jejich účel. Programátoři jsou odpovědni za jejich spouštění ještě předtím, než kód nasadí na testovací prostředí a poskytují tak okamžitou zpětnou vazbu o kvalitě jejich zdrojového kódu. (31)

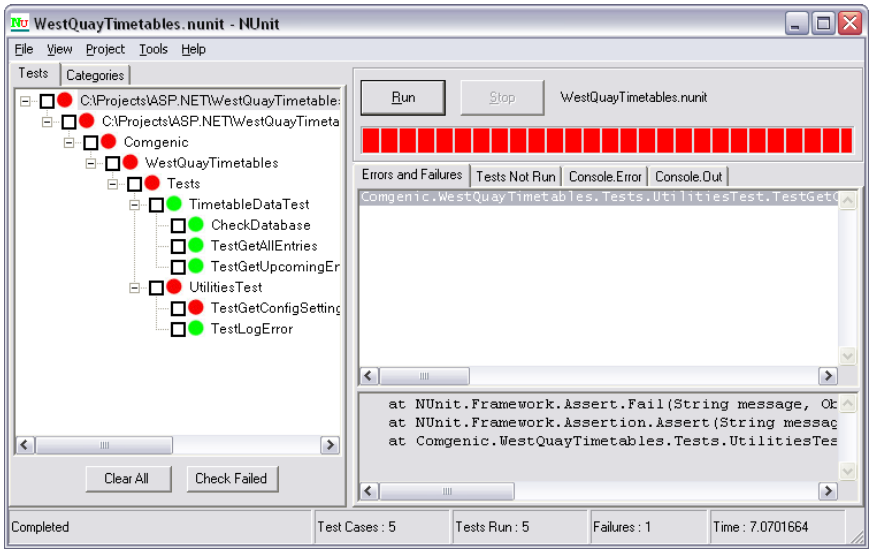
Testovací tým se na vývoji těchto unit testů nepodílí, avšak může tyto testy pravidelně spouštět. Vždycky je totiž lepší, pokud je pro obsáhlé testovací sady unit testů určen jiný hardware, například testovací tým má běžně k dispozici další počítače, určené pouze pro spouštění testů a podobné testovací účely. Výhodou dalšího počítače je také fakt, že programátor nemusí čekat na dokončení svých unit testů na svém vlastním počítači.

3.3.1 Použité nástroje a techniky

Technika TDD se vyznačuje vkládáním unit testů přímo do zdrojového kódu, resp. při vytváření unit testů záleží, který programovací jazyk je při vývoji software použit. Téměř pro každý programovací jazyk existuje framework[3], který pomáhá programátorům používat nástroje k ulehčení vytváření unit testů. Výhodou je, že tyto frameworky jsou zdarma.

Příkladem může byt NUnit framework (http://www.nunit.org/) používající se pro aplikace založené na .NET technologií nebo JUnit framework (http://www.junit.org/) pro programovací jazyk Java. Na trhu existuje mnoho grafických uživatelských prostředí (GUI) pro správu a spouštění unit testů.

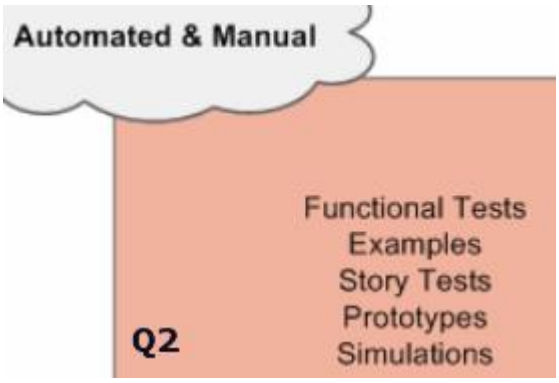
[3] Framework je univerzální, znovupoužitelná softwarová platforma používající se k vývoji aplikací. Software frameworky obsahují knihovny kódů, programové aplikační rozhraní a sadu nástrojů, které spojují rozdílné komponenty k tomu, aby vývoj na projektu mohl začít. (45)



4 Obrázek - NUnit základní obrazovka (43)

Obrázek ukazuje, jak vypadá GUI pro NUnit framework. Na levé straně obrázku je seznam vytvořených NUnit testů. Vývojář nemusí vždy spustit všechny, ale může si checkboxy vybrat ty, které chce spustit (resp. ty, které by měly projít), pokud ve zdrojovém kódu udělal takové změny, aby test prošel. Na pravé straně obrázku je výpis aktuálně provedeného testu označeného v seznamu. Ve spodní části vidíme, že test vrátil výjimku a neskončil úspěšně.

3.4 Kvadrant Q2



5 Obrázek - Kvadrant Q2 (32)

Druhy testů v kvadrantu Q2 podporují práci vývojového týmu stejně jako je tomu v kvadrantu Q1, ale na vyšší úrovni. Tyto testy jsou zaměřené na business (někdy také nazývané testy zákazníka) a definují externí kvalitu a zákaznické požadavky. V agilním vývoji jsou tyto testy vytvářeny podle výstupů zadavatelského týmu, který v požadavcích popisuje uživatelské scénáře. Pod tímto pojmem rozumíme funkcionální testy, které kontrolují, zda produkt splňuje dohodnuté požadavky, tzn. musí být napsány dostatečně srozumitelně, aby byl vývojový tým

schopný danou funkcionalitu vytvořit. Většinu těchto funkcionálních testů potřebuje tým zautomatizovat, protože jedním z účelů kvadrantů Q1 a Q2 je rychlé poskytnutí informací, které okamžitě nastartuje proces opravování chyb. (31)

Zákazník tedy definuje uživatelské scénáře[4] (use cases), resp. doručí dokumenty, kde jsou popsány požadavky. Například: uživatel vyplní pole na první obrazovce a stiskne tlačítko, poté se na druhé obrazovce objeví určitá hodnota. Zákazník v těchto uživatelských scénářích nebývá příliš specifický ohledně rozložení polí a tlačítek na obrazovkách, uživateli systém slouží spíše pro vysvětlení funkcí, proto se také díky těmto scénářům dá napsat mnoho funkcionálních testů, které se dají jednoduše zautomatizovat, což je výsledek kvadrantu Q2.

Ovšem v rámci kvadrantu Q2 se testují nejen funkcionality, ale také grafické uživatelské prostředí (GUI). Aplikace musí odpovídat požadavkům na vzhled a samozřejmě měly by být dodrženy jisté konvence, na které je uživatel zvyklý, např. aplikace vyvíjena pod OS Windows by měla mít možnost zavření křížkem v pravém horním rohu apod. Jedná se tedy o typické intuitivní prvky, které nesčetné aplikace podporují stejným způsobem. Dalším typickým příkladem je umístění tlačítek OK a Storno vedle sebe zleva doprava. Toto jsou příklady, které jsou obecně známé, avšak typicky se testy na tyto prvky nevytváří. GUI testy se týkají hlavně umístění tlačítek, polí, různé popisky, názvy polí konkrétní aplikace. Testovací tým by testy grafického rozhraní měl také zautomatizovat.

Problém však nastane, když žádné takové rozhraní ještě neexistuje, není také vytvořena žádná databáze a nejsou dostupná žádná data. Řešení je použití mock objektů. *„V podstatě se jedná o nahrazení reálného objektu testovací fasádou, která neprovádí žádnou funkcionalitu nahrazovaného objektu – jen se jako tento objekt tváří. Místo původní logiky objektu je vloženo chování, které ve svém testu potřebujete.“* (46) Příkladem je tedy simulace GUI nebo databáze. Například zdrojový kód používající přístup do databáze konkrétním SELECT[5] dotazem by skončil chybou. Ovšem programátor či tester si může vytvořit zmíněné mock objekty a definovat, který konkrétní řetězec se má vrátit. Stejně tak si tester může nadefinovat, co se stane, když uživatel stiskne tlačítko Button1, i když toto tlačítko ještě nemusí existovat.

[4] Uživatelský scénář (angl. use case) je popis, jak by měl uživatel postupovat v systému, aby splnil úlohy. Popisuje sekvenci interakcí uživatele se systémem bez specifikace uživatelského rozhraní. 68)

[5] SELECT dotaz je databázový příkaz, který požaduje vrátit nějaká data z databáze. (autor)

Samozřejmě ne všechny tyto testy je možné zautomatizovat. Nedávno jsem se setkal s chybou ve webové aplikaci, kdy některá textová pole byla příliš široká pro zobrazení, a tak byla větší část pole skryta do neznáma, a navíc toto okno neobsahovalo žádný posuvník.

Testovací tým by se měl při plánování testů rozhodnout, zda je efektivní tyto nebo i jiné testy zautomatizovat. Určitě záleží na druhu aplikace a jejím zaměření. Vytvoření GUI testů na jednu stranu není s pomocí dnešních podpůrných nástrojů tak časově náročné, ovšem problém je, že GUI je částí aplikace, která je upravována velmi často a dost často také po každém novém změnovém požadavku ze strany zákazníka. Automatizované testy je poté nutné aktualizovat, alternativou k testování GUI je bohužel pouze manuální testování.

3.4.1 Použité nástroje a techniky

V rámci testování v kvadrantu Q2 se již klade větší důraz na funkcionalitu aplikace, nejen na kvalitu zdrojového kódu. Více se testuje z pohledu zákazníka, který očekává, že jeho požadavky jsou splněny, jeho testovací scénáře fungují a uživatel si může aplikaci již sám vyzkoušet.

Technika Behaviour-Driven Development (BDD), která se používá v kvadrantu Q2, pomáhá soustředit vývoj k plnění a poskytnutí prověřitelných hodnot aplikace poskytnutím společného slovníku, který překlene propast mezi chápáním zákazníka a technicky zaměřenými členy vývojového týmu. BDD technika se spoléhá na tři základní principy:

- Strany Business (zákazník) i Technologie (vývojový tým) by měly prostudovat systém stejným způsobem – nalézt potřebný slovník a konzistentní termíny, aby nedošlo k nedorozuměním.
- Identifikace prokazatelných hodnot produktu jako celku pro business.
- Nepřikládat velkou váhu obsáhlým plánům – ztrácí na hodnotě s postupem času, jelikož projekt se stále během vývoje mění.

BDD technika spoléhá na použití malého, velmi specifického slovníku k prevenci nedorozumění na všech úrovních týmu, tj. aby byl analytik schopen porozumět vývojáři. Obecně technika zdůrazňuje potřebu, aby všichni členové týmu byli na stejné úrovni chápání produktu, a používali ta samá slova, kterým každý v projektovém týmu rozumí. (47)

Pro podporu této techniky existuje několik frameworků, např. Cucumber, easyB nebo NBehave, osobně jsem se setkal právě s posledně jmenovaným frameworkem NBehave,

což je framework určený pro .NET technologii. Primárním cílem frameworku je jasné definování a realizování cílů aplikace. NBehave kód je psán podobně jako unit testy, avšak unit testy neposkytují dostatek informací ostatním členům týmu (jako to umí NBehave), kdy jsou do kódu vepsány texty popisující každý krok a každý testovací scénář, resp. co se od aplikace očekává při tomto kroku. Po exekuci takového kódu jsou zobrazeny všechny možné scénáře, které proběhly a byly otestovány, s výsledky těchto testů velice podobně jako je tomu u techniky TDD. (48)

Výhoda BDD je v tom, že tester bude mít k dispozici kompletní testovací scénář včetně jednotlivých kroků, které tyto naprogramované BDD testy v programu NBehave provedly. Scénáře jsou mnohem více srozumitelné a ani zákazník nebude mít žádný problém pochopit výsledky testů. Srovnání TDD a BDD ukazuje následující příklad.

Následující kód je výsledkem techniky TDD – ukazuje, jak vypadá typický unit test v jazyce C# s použitím NUnit frameworku. Testem je výběr peněz z bankovního účtu a úkolem testu je vyzkoušet, zda se výběr provede nebo zda byla transakce správně zastavena z důvodu nedostatku prostředků na účtu. Výstupem toho testu je pouze informace, zda test proběhl v pořádku (passed test) nebo skončil chybou (failed test). Více informací test neposkytuje.

```
[Test]
public void VyberTest()
{
  Ucet u = new Ucet(100);
  u.Vyber(30);
  Assert.AreEqual(a.Zustatek, 70);
  Assert.Throws<NedostatekPenezException>(a.Vyber(100));
}
```

(autor, podle (49))

Následující kód je výsledkem použití techniky BDD. Tester vytvořil scénář (test), kdy uživatel vlastní prázdný bankovní účet a provede vklad.

```
story.WithScenario("Vklad penez na ucet")
  .Given("Muj bankovni ucet je prazdny", () => { ucet.Zustatek = 0; })
  .When("Vlozim 100 korun", () => ucet.Vklad(100))
  .Then("Zustatek na uctu by mel byt 100 korun", () =>
ucet.Zustatek.ShouldEqual(100));
```

(autor, podle (49))

Výhodou takového testu je, že po jeho exekuci se vytvoří záznam s detailním výpisem kroků, které tester provedl. Takový výstup s výsledkem testu může vypadat následovně:

```
Scenar1: Vklad penez na ucet
        Given Muj bankovni ucet je prazdny
        When Vlozim 100 korun
        Then Zustatek na uctu by mel byt 100 korun – FAILED


NUnit.Core.Exceptions.NotEqualAssertionException:
  Equal assertion failed: [[0]]!=[[100]]
```

(autor, podle (49))

Test skončil chybou, konečný zůstatek neodpovídá – zobrazen text FAILED, což znamená chybu. Pod scénářem je uvedená specifická chyba – číslo 0 se nerovná číslu 100, tzn. funkce vložení peněz neproběhla v pořádku. Testovací tým okamžitě ví, o jakou chybu jde, a může ji předat programátorům na opravu.

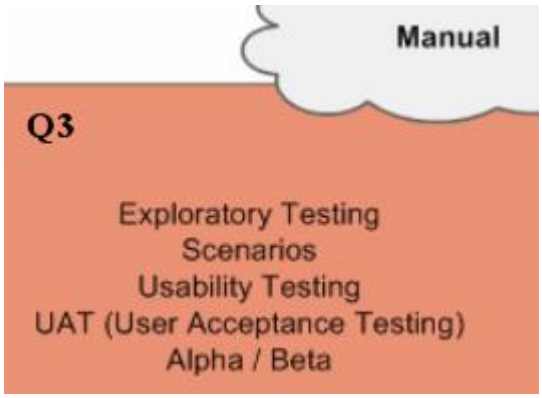
Pokud již existuje grafické rozhraní aplikace, které by již v rámci testování kvadrantu Q2 mělo být v této fázi testování aspoň z části hotové, je vhodné použít specifické nástroje k tomu určené. Existuje jich celá řada a dokonce jsou vypisovány i soutěže, např. ATI Automation Honors (http://www.automatedtestinginstitute.com), které hodnotí tyto nástroje v různých kategoriích a zaměřeních.

Z komerčních produktů uvedu podle mého názoru asi nejznámější QuickTest Professional (QTP), se kterým mám již dobré zkušenosti. Měl jsem možnost vyzkoušet si i celou řadu open

source[6] nástrojů, např. Selenium nebo WatiN. Pokud bych je měl porovnat, každý program k testování přistupuje trošku jinak, v každém testovacím software musí tester naprogramovat více či méně kódu. Společnosti se snaží vytvořit testovací programy, aby byly co nejvíce intuitivní, avšak bez programovacích znalostí se dobré testy neobejdou.

Základem těchto programů je tzv. Recorder, což je obvykle tlačítko sloužící pro začátek záznamu, resp. po stisknutí tlačítka aplikace začne zaznamenávat všechny uživatelské kroky na počítači, ať už jde o zapnutí testovaného softwaru, přihlášení nebo vyplnění dat a jejich vyhodnocení. Každý krok je aplikací, např. QTP, zaznamenán a tato aplikace kroky uloží jako zdrojový kód, který může být v budoucnosti kdykoli spuštěn. Na první pohled se zdá, že k tomuto úkonu nepotřebuje tester žádné zkušenosti s programováním, ale některé kroky je třeba naprogramovat samostatně, například práce s výjimkami, resp. s události, která nastane, jestliže test skončí chybou (nalezena chyba v aplikaci). Takové situace je lepší ošetřit ručně naprogramováním svého vlastního kódu, jelikož standardně celý test skončí chybou, i když se mohlo jednat třeba jen o triviální neshodu, která není chybou, avšak test je přesto ukončen.

3.5 Kvadrant Q3



6 Obrázek - Kvadrant Q3 (32)

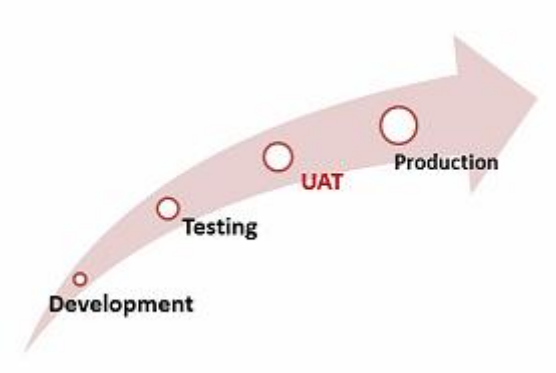
Kvadrant Q3 znázorňuje akceptační testy (UAT)[7], tj. testy prováděny zákazníkem, aby se ujistil, že je vše podle požadavků. Toto testování logicky probíhá ve finální fázi

[6] Open Source software je volně šířitelný počítačový software obsahující zdrojový kód. Přesná definice, viz (50).

[7] User Acceptance Test, resp. akceptační testy patří mezi druh testů, které provádí zákazník, aby se přesvědčil, že všechno bylo vyvinuto podle jeho požadavků. (51)

testování před tím, než se kód přesune do produkčního prostředí, resp. do takového, které již uživatelé denně používají pro svůj pracovní výkon. (51)

Následující obrázek znázorňuje běžná implementační prostředí:



7 Obrázek - Implementační prostředí (51)

Development prostředí je určeno pro programátory, kde vyvíjejí celý systém. Poté svůj hotový zdrojový kód nasadí na prostředí testovací – Testing. Na tomto prostředí si testovací tým definuje svá vlastní data a testuje. Typicky jsou ke každému prostředí nasazeny všechny komponenty, které systém pro fungování potřebuje, od své vlastní databáze až po webové služby apod. Poté, co testovací tým dokončí svou práci, je určen termín, kdy a na jak dlouho bude tento kód z testovacího prostředí, tzn. kompletně otestovaný, nasazen na prostředí UAT, které má plně pod kontrolou zákazník.

Akceptační testy mají za cíl odhalit různé funkcionality, které během návrhu a vývoje zákazník opomenul. Občas se stává, že i když produkt splňuje všechny požadavky a všechny testy byly v pořádku, tak produkt nesplňuje to, co zákazník opravdu chtěl. (31)

Testování v rámci kvadrantu Q3 má za úkol otestovat produkt tak, jako kdyby byl již teď používán v běžném uživatelském provozu. Provádí se zejména manuálním testováním, některé testy se dají zautomatizovat, ale v této fázi je praktické použít spíše své smysly a kriticky se podívat na produkt z pohledu koncového uživatele, resp. vytvořit si své vlastní testy použitelnosti[8]. Sleduje se a zaznamenává (ne)spokojenost uživatele s produktem během testování od procesu přihlášení po proces vytvoření klienta nebo vystavení faktury (pokud se jedná například o účetní software). Během tohoto testování se uživatel soustředí na vyzkoušení všech funkcionalit a s použitím selského rozumu hledá chyby, kde by se

[8] Usability Test, resp. testy použitelnosti se používají pro zhodnocení produktu pohledem koncového uživatele. (52)

pravděpodobně při práci mohly objevit. Často se zapomíná, že pokud software vytváří dokumenty, tak je potřeba otestovat i je, zda jsou použitelné. Tester by také neměl přehlédnout nápovědy v aplikaci a také domluvený vzhled aplikace (GUI), což je první věc, kterou zákazník uvidí. (32)

„Zákazník aplikaci platí, a proto je v pořádku, že ji hodnotí ze svého pohledu. Pro vývojový tým je nutné si uvědomit, že otázky okolo GUI není vhodné odsouvat a že je nutné podobu GUI a jeho řešení se zákazníkem probírat v průběhu celého vývoje. Jen tím se dá částečně odstranit možné překvapení zákazníka, když poprvé dostane aplikaci do rukou. I tak se ale nedá vyhnout tradičním chybám typu požadavku na jinou barvu nadpisů, zvětšení písma v menu atd.“ (56)

Cílem tohoto testování je identifikovat a zaznamenat uživatelskou spokojenost s produktem. Tyto testy dají uživateli nové myšlenky, jakým způsobem produkt v další verzi vylepšit, upravit nebo přidat úplně nové funkce, které ulehčují práci. (31 a 52)

3.5.1 Použité nástroje a techniky

Testovací tým v rámci akceptačního testování testuje spolu se zákazníkem na UAT prostředí, jelikož se na něm mohly objevit chyby při nasazení nového kódu z Testovacího prostředí. Prakticky se v této fázi používají nástroje uvedené během testování v kvadrantu Q2 určené pro funkcionální testování produktu. V rámci použití LeanSD je vhodné, aby se zákazník účastnil testování přímo na pracovišti nebo aspoň na prezentaci produktu. Nejlepší volbou pak je, aby seděl nablízku testerům, kteří mohou zodpovědět všechny otázky, na které určitě již znají odpovědi.

3.6 Kvadrant Q4

8 Obrázek - Kvadrant (32)

V rámci kvadrantu Q4 je již produkt hotov a probíhají výkonnostní a zátěžové testy. Tyto testy by se také daly zařadit pod hlavičku akceptačních testů. Softwarový program musí být schopný něco vydržet, např. připojení mnoha set uživatelů najednou, splnění desítek

uživatelských požadavků v jeden moment apod. Tyto vlastnosti byly jistě také domluveny v průběhu návrhu produktu.

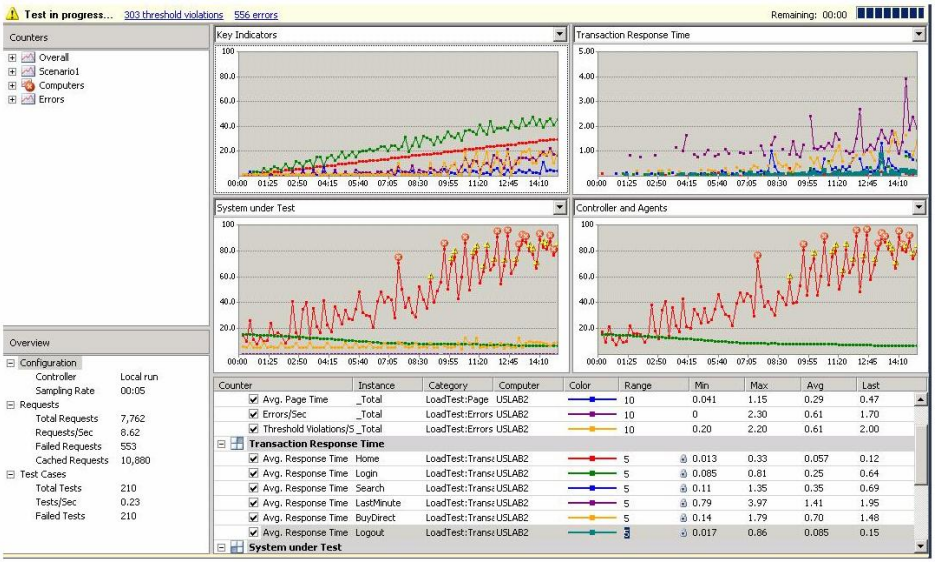
Softwarový produkt musí být také jistým způsobem zajištěn proti neoprávněnému vniknutí, zabezpečen proti jakékoli změně v datech nebo poškození aplikace. Testovací týmy proto vytvářejí bezpečnostní testy (Security test), které zajistí nejen bezpečnost aplikace proti různým softwarovým útokům, ale i proti nesprávnému použití samotnými uživateli, ať už jejich snaha byla provést změnu účelně nebo náhodou.

Jestli vy nebo váš tým plánujete novou verzi produktu nebo celý nový projekt, domluvte se na typech testů, které budete používat v rámci testování kvadrantu Q3 a Q4. Nenechávejte základní aktivity jako je zátěžové testování a testování použitelnosti až na konec, mohlo by pak být příliš pozdě pozměnit architekturu celého systému, která by uspokojila požadavky na robustnost systému. (31)

3.6.1 Použité nástroje a techniky

Kvadrant Q4 se vyznačuje testy výkonnosti produktu. Výkonnost se dá otestovat tak, že pošleme aplikaci mnoho požadavků najednou a testujeme, zda aplikace stále funguje, jestli je schopna zpracovat všechny požadavky a také zejména to, jak rychle je schopna na tyto požadavky vrátit odpověď. K tomuto účelu se používá zátěžové testování (Load testing), jehož cílem je změřit chování systému při vysokém zatížení. Dle mého názoru patří mezi velmi oblíbené aplikace, které se k těmto testům používají, programy loadUI nebo LoadRunner.

Výsledkem jsou pak grafy, které zobrazují rychlost odezvy na požadavky, zatížení aplikačního serveru, databáze apod. Následující obrázek zobrazuje výsledné grafy po ukončeném zátěžovém testování ve Visual Studiu.



9 Obrázek - Výkonnostní testování ve Visual Studiu (53)

Výsledek testů může poradit, které softwarové či hardwarové změny je třeba provést. Pro maximální zatížení, resp. zkoumání při jakém zatížení systém začne být nefunkční, se používají Stress testy, které během času navyšují své nároky na systém a dokážou identifikovat, při jakém zatížení již systém není schopný odpovídat na požadavky uživatele nebo kdy se systém kompletně zastaví.

4 Lean myšlení

Lean production neboli "štíhlá výroba" pochází z firmy Toyota, kde vznikla v 50. letech 20. století jako alternativa k hromadné výrobě v prostředí, které vyžadovalo vysokou úroveň flexibility a postrádalo finance na nákladné investice. Dá se tedy říci, že jde přístup k výrobě, kde je zapotřebí méně lidského úsilí, kapitálu a času. (17)

Základ tehdejší výroby ve firmě Toyota stál na dvou pilířích – JIT (just-in-time) a JIDOKA (autonomation). První pilíř JIT je označení způsobu výroby či dodávky „právě včas“, což znamená, že jakákoli dodávka materiálu na některé z pracovišť oběma směry byla provedena až ve chvíli, kdy byl materiál na konkrétním pracovišti potřeba. Nedocházelo tak ke zbytečným nákladům na skladování a na jiných místech podniku. (19)

JIDOKA je označení pro „automatizaci“ činností na pracovišti. Někdy je označována jako „stop-the-line“. Znamená to, že pokud pracovník pozná vadu na výrobku, proces výroby je automaticky a okamžitě zastaven do té doby, než je chyba opravena. Prakticky se při aplikaci tohoto přístupu používají různá zvuková či světelná zařízení, která okamžitě upozorní pracovníka na vadu v procesu výroby nebo výrobku.

Toyota oba pilíře aplikovala nejen ve svých továrnách při výrobě, ale i při návrhu a vývoji nových automobilů. Tento system pak označovala – Toyota Product Development Systém[9] (TPS). „*TPS je způsob myšlení stavějící na kultuře neustálého zlepšování, podpory zaměstnanců, doručování hodnoty*.“ (19)

Podle (18) lze Lean myšlení prezentovat jako „… způsob práce, zaměřený na přidávání hodnoty pro zákazníka a nepřetržité odstraňování plýtvání z každé aktivity.“

4.1 5 principů Lean myšlení

Podle (20) lze shrnout Lean myšlení do pěti následujících principů:

1. Definice hodnoty
2. Identifikace částí hodnotového toku
3. Tvorba hodnotového toku

[9] Hlavní představitel a zakladatel TPS a dalších systému v Toyotě byl Taiichi Ohno. (1)

4. Princip tahu
5. Dosahování dokonalosti

Firma musí nejdříve definovat hodnotu, která přinese zákazníkovi maximální užitek, tedy přesně určit „misi“ produktu. Odchylka od těchto specifik je v rámci Lean myšlení označována jako plýtvání. Tým by neměl upřednostňovat aktuální potřeby akcionářů, vrcholového vedení či technologie, ale měl by se soustředit jen a pouze na hodnotu produktu, i když by to znamenalo přehodnotit základní výrobní procesy. (20)

Hodnotový tok můžeme vysvětlit jako soubor všech činností ve výrobním nebo jiném procesu. Každý takový proces obsahuje několik činností a podle (20) se dají tyto činnosti rozdělit do následujících třech prvků:

1. Vytvářející hodnotu
2. Bezprostředně nevytvářející hodnotu, ale nevyhnutelné
3. Nevytvářející hodnotu a okamžitě odstranitelné

Tvorba toku spočívá v realizaci takových kroků, které zajistí plynulé navazování činností v rámci procesu výroby tak, aby nedocházelo k prostojům nebo jiným negativním vlivům, resp. podle Lean myšlení by každá činnost měla být vykonána s co nejmenším úsilím a co nejrychleji. (20)

Princip tahu znamená vytvoření takového produktu, který zákazník opravdu chce, tzn. že nejsou vytvářeny produkty, které nebyly nikdy vyžadovány. (20)

Dosahováním dokonalosti se myslí neustále zlepšování podnikových procesů tak, aby se zamezilo plýtvání a stále se vytvářela maximální hodnota pro zákazníka. (20)

4.2 Lean software development

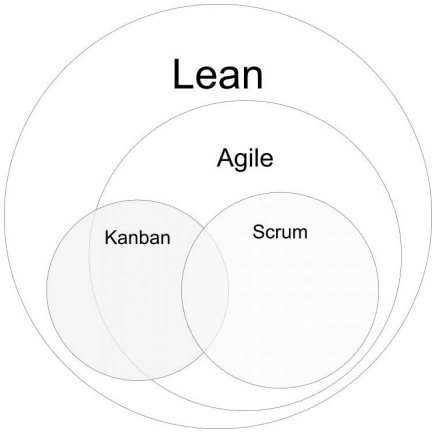
Zjednodušeně řečeno přijetí konceptu Lean myšlení při vývoji software, o kterém pojednávala předchozí kapitola, je označován jako Lean software development (dále jen LeanSD), který je obsahem této kapitoly. V následujícím textu představím samotný koncept vývoje software podle LeanSD a uvedu jeho základní principy. Každý princip podrobně rozeberu a uvedu, jak tyto principy mohou být přijaty testovacími týmy.

4.2.1 Zařazení Lean software development

Metodika je komplexní návod pro vývoj software s definovanými principy, procesy, praktikami, rolemi a různými technikami. Čtenář by tedy mohl namítnout, že LeanSD kvůli absenci několika prvků z definice metodiky jako metodika označen být nemůže, avšak v literatuře se s tímto označením může čtenář setkat.

Můj pohled na označení LeanSD spíše inklinuje k názoru, který vyjádřil (21), a to že LeanSD není ani tak metodikou vývoje, jako spíše metodou pro zlepšování stávajících postupů vývoje.

Další otazník visí nad otázkou, zda se LeanSD dá zařadit mezi agilní metodiky jako je Scrum nebo Extrémní programování (pokud bychom tedy označili LeanSD za metodiku). Dle mého výzkumu a také podle (6) převládá názor, že Lean, resp. koncept Lean myšlení, je nadmnožinou agilního přístupu a poskytuje rámec pro agilní principy, které z Lean myšlení vychází, a dále pak, že metodiky jako Scrum a podobné aplikují principy LeanSD ve svém procesu vývoje software (principům LeanSD se věnuje následující podkapitola). Graficky tento pohled ilustruje následující obrázek:



10 Obrázek - Pozice Lean myšlení (6)

Obrázek znázorňuje, že postavení Lean myšlení a agilního přístupu nelze oddělit a nejedná se tak o dva samostatné přístupy ve vývoji software.

4.2.2 Organizace vývoje podle Lean software development

LeanSD koncept přejímá základní principy agilního vývoje, což znamená, že také stejně jako agilní metodiky požaduje, aby se týmy řídily požadavky vycházející z 12 principů stojícími za Agilním Manifestem podle (27):

Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.

To znamená, že průběh vývoje software bude probíhat v iteracích[10] o délce až jednoho měsíce, kdy na konci každé iterace bude předveden produkt k vyzkoušení zákazníkem, který očekává, že zadané požadavky na začátku každé iterace byly splněny a produkt těmito požadavky disponuje.

Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.

Zákazník na začátku každé iterace definuje, upravuje nebo odstraňuje požadavky z iterace předešlé a očekává, že na konci iterace obdrží produkt s modifikovanými požadavky podle svého přání.

Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.

Všichni členové týmu – od analytiků přes vývojáře až po testery musejí být součástí každodenní schůzky ať už svou přítomností v místnosti či pomocí online konference nebo aspoň na telefonu. Každodenní schůzky trvají typicky necelých 15 minut, ve kterých jsou zodpovězeny otázky typu – kdo a co dělal včera, bude dělat dnes, a co člověku ve vykonání práce brání. Zákazník (lidé z byznysu) by měli být přítomni pro možné zodpovězení otázek. Z těchto předpokladů vyplývá, že LeanSD je určen spíše pro menší týmy s počtem maximálně deseti vývojářů nejlépe situovaných blízko sebe.

Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.

Každý člen týmu by měl být ve své práci znalý, musí být schopný rozhodnout a vzít na sebe zodpovědnost za svá rozhodnutí, bude-li to třeba. Testovací týmy v LeanSD spoléhají na vysokou technickou znalost svých členů, po kterých se vyžadují zkušenosti a jejich přenos napříč celým týmem.

[10] Iterace je několika týdenní časový úsek, ve kterém je naprogramována a otestována část produktu. Tato část je na konci iterace nasazena a připravena ke kontrole zákazníkem. (autor)

Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.

LeanSD počítá s lokací týmu ve stejné místnosti, či ve stejném patře budovy, preferuje se osobní komunikace, která zabraňuje prodlevám při čekání na zodpovězený email apod. (upřednostňuje se osobní komunikace před psanou formou.

Hlavním měřítkem pokroku je fungující software.

Tým musí být schopen na konci každé iterace předat fungující software. Z toho důvodů si musí rozvrhnout úkoly a uvědomit si, které úkoly je schopen nejen testovací tým splnit v rámci aktuální iterace.

Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.

Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.

Jednoduchost--umění maximalizovat množství nevykonané práce--je klíčová.

Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.

Testovací tým se musí naučit pracovat tak, aby si co nejvíce zjednodušil práci, omezil plýtvání ve všech svých činnostech a soustředil pouze na činnosti, které zákazníkovi zvyšují užitek. Samo-organizující tým je každý agilní tým a znamená to, že v rámci možností může být jedna role zastoupena členem pracující na jiné roli (pozici). Například test manager by se mohl dočasně stát test excutorem a vypomoct ostatním členům testovacího týmu nebo test executor může pomoci s přípravou testovacích případů a dočasně přijmout roli test designera.

Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

4.2.3 7 principů Lean software development

V předchozí kapitole jsem vysvětlil, co znamená Lean myšlení jakých je jeho pět základních principů.[11] Tyto principy jsou modifikovány a aplikovány tak, aby bylo možné je přijmout softwarovými společnostmi. Z původních pěti principů vzniklo podle (4) nebo (1) celkem

[11] Poppendieckovi ve své knize (1) představili 22 nástrojů napříč všemi pěti principy, které poskytují rady pro týmy, které se snaží aplikovat tyto Lean principy ve svém agilním vývoji software

sedm principů, které postupně rozeberu v následujícím textu, a zaměřím se na to, jak by tyto principy mohly být přijaty testovacími týmy.

Princip 1: Eliminovat plýtvání

Nejdříve si definujeme pojem plýtvání, který je popisován podle (1) jako všechno, co nepřináší zákazníkovi hodnotu. Např. práce na úkolu, který není v dané chvíli požadován, přeprava, časové prodlevy, nadbytečné procesní kroky a z pohledu testování se jedná samozřejmě i o nalezené defekty.

Eliminovat plýtvání je fundamentální princip Lean myšlení, na který ostatní principy navazují. Proto prvním krokem implementace Lean je naučit se, jak vyhledat místa, kde probíhá zbytečné plýtvání, a eliminovat je. Druhým krokem je najít další taková místa a poté tento krok opakovat. Po nějaké době mohou být dokonce i věci, které se zdály být na první pohled nezbytné, najednou označeny jako plýtvání a postupně mohou být eliminovány.

Podle (1) existuje sedm hlavních druhů plýtvání ve vývoji software:

7 druhů plýtvání
Nedokončená práce
Zbytečné činnosti
Zbytečná funkcionalita
Střídání úkolů
Prodlevy
Defekty
Doba k vyhledání odpovědí

1 Tabulka - 7 druhů plýtvání (autor, podle (1))

Nedokončená práce je při vývoji software taková práce, která musela být odložena kvůli novým požadavkům. Z pohledu programátora se může jednat o nedodělanou část kódu, která by mohla být dokonce otestována, avšak jako celek nemusí splňovat požadovanou funkcionalitu. Z toho pohledu se jedná o plýtvání, jelikož určitá funkcionalita

nebyla dokončena, i když již byly investovány někdy i nemalé prostředky pro vývoj této funkcionality. (1)

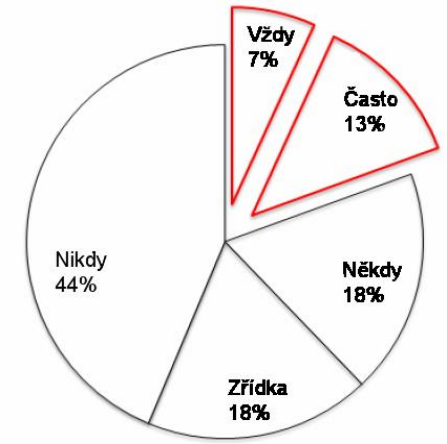
Z pohledu testovacího týmu to mohou být nedokončené automatizované regresní testy, které mají za úkol otestovat již v minulosti hotovou funkcionalitu, jež v předešlých iteracích fungovala, a bylo efektivní k těmto funkcím vytvořit automatizované testy, které v tomto případě nebyly dokončeny. Tato nedokončená práce testovacího týmu může při nedostatečných lidských zdrojích znamenat, že jedna nebo více funkcionalit, které v minulosti (předešlých iteracích vývoje) vždy fungovaly, mohou nyní hlásit chyby, o kterých se testovací tým nemusí dozvědět. To vede podle (1) k vytvoření dalšího druhu plýtvání – k defektům.

Defekt (chyba) je podle (1) množství plýtvání způsobené chybou ve vytvořeném produktu a čas potřebný k opravě chyby. Způsob, jak zmenšit vliv defektu na produkt, je najít chyby co nejdříve, nejlépe ihned, jakmile se objeví. Proto je řešením, jak zabránit negativním vlivům na produkt, testování co nejdříve. Platí totiž, že čím dříve je defekt nalezen, tím menší plýtvání (náklady) jsou potřeba k jeho opravě.

Zbytečné činnosti jsou podle (6) všechny činnosti, které nejsou nikým vyžadovány a samy o sobě nepřinášejí zákazníkovi a tím pádem ani týmu žádnou hodnotu. Jedná se o činnosti, které zbytečně využívají zdroje a skrývají podstatné problémy. Podle (6) se jedná hlavně o byrokratické papírování, které je potřeba během vývoje software předkládat. Jedná se například o schválení změn, vyžadování podpisů zákazníka a evidování manipulace s různými druhy dokumentace. LeanSD počítá s menšími týmy situovaných do jedné místnosti a (6) radí, aby co možná nejvíce činnosti s tímto spojených se vyřešilo rychle na schůzce bez zbytečných detailů a za pomocí prezentace na tabuli.

Zbytečná činnost testovacího týmu může například spočívat v absenci jasně definovaného a zdokumentovaného procesu řízení defektů. Věřím, že zdokumentovat tento proces nepatří mezi plýtvání, avšak co je jasným plýtváním jsou právě tyto činnosti, které vedou ke zdokumentování tohoto procesu, např. posílání emailů, telefonní hovory, tisknutí a předávka procesu na papíře k revizi apod. Tyto činnosti jsou jasným plýtváním, které se mohou nahradit deseti minutovou schůzkou s několika členy v týmu a manažerem projektu pro zajištění (nákupu) potřebného nástroje.

Zbytečná funkcionalita nebo funkcionalita nad rámec požadavků, která by se mohla zákazníkovi hodit, je samozřejmě z lidského hlediska pozitivní chování, avšak každá vlastnost produktu, která nebyla vyžadována, nemusí také nikdy být použita a tento vytvořený kód se může v budoucnosti projevit a vytvářet chyby. Proto zdrojový kód, který není nyní vyžadován, je označen jako plýtvání. (1)



11 Obrázek - Využitelnost funkcí aplikace (autor, podle (24))

Jak je vidět na obrázku, vlastnosti a funkce, kterými typický systém disponuje, jsou využívány pouze z jedné pětiny, bereme-li v úvahu pouze takové, které jsou využívány vždy nebo často. Velmi dobře zde můžeme aplikovat Paretovo pravidlo[12] a říci, že vývojový tým by měl upřednostnit 20 % nejdůležitějších funkcionalit, které přinesou zákazníkovi maximální užitek, a zaměřit se na ně již od počátku projektu.

Stejně tak pro testovací tým to znamená komplexní testy zejména těchto vlastností produktu. Testeři by měli vytvořit testy pokud možno se 100% pokrytím zdrojového kódu a v maximální možné míře použít funkcionální testy na softwarovou aplikaci za pomocí automatizovaných testů z důvodu jejich časté opakovatelnosti.

Tým testerů může také testovat produkt specializovanými testy, které testují vlastnosti produktu, které nebyly vyžadovány, a to například zátěžovými testy, které testují stabilitu systému při extrémně velké zátěži. Zbytečná činnost je vytváření zbytečných testů bezpečnostních, které nemusely být požadovány, proto by se měl tým testerů v průběhu

[12] Paretovo pravidlo 80/20 znamená, že 80% spokojenosti zákazníka získate díky implementací 20% nejdůležitější funkcionality produktu, kterou zákazník skutečně potřebuje (autor)

plánování projektu informovat ohledně vlastností produktu, kterými musí disponovat a které se musí později otestovat.

Ke střídání úkolů dochází, když má testovací tým na práci několik úkolů ve stejný čas a členové týmu tak musejí pracovat paralelně na všech úkolech. V tomto případě dochází k plýtvání tím, že například tester pracuje na několika automatizovaných testech najednou a musí tak pokaždé „přepnout“ svůj myšlenkový proud na jiný test.

Testerova práce není tak efektivní, když má za úkol najednou připravit několik testovacích scénářů k novým funkcím, které je potřeba otestovat. Vytváření automatizovaných testů vyžaduje také plné soustředění, a pokud jste četli předchozí text podrobně, tak víte, že kvůli plýtvání jako je střídání úkolů, by určitě později také docházelo k plýtvání ve formě nedokončené práce. Ze svých zkušeností vím, že při práci na mnoha úkolech najednou se vždy na něco zapomene.

Podle (22) uvedu příklad. Tým má na práci 2 úkoly. Každý úkol zvlášť zabere 2 týdny práce. Pokud by ale tým pracoval na obou úkolech najednou, je velice pravděpodobné, že by tým práci ukončil až ke konci týdne pátého.

Časové prodlevy jsou podle (1) jedním z největších plýtvání při vývoji software. Prodlevy nastávají pokaždé, když jeden člen týmu musí čekat na jiného, tím vznikají zpoždění při startu projektu, čekání na odpovědi, čekání na požadovanou dokumentaci, zadání apod. Tomu protiřečí – jak bude dále v textu uvedeno – jeden z principů LeanSD „rozhodnout co nejpozději“. Ovšem pokud by rozhodnutí mělo zdržovat vývoj či testování, musí být rozhodnuto okamžitě.

Prodlevy vznikají i v průběhu testování kvůli časté potřebě otestovat danou funkcionalitu či vlastnosti manuálně, což znamená zapojit často omezené lidské zdroje, které zajistí kvalitu produktu, a potvrdí, že produkt může být prezentován zákazníkovi (nasazen do produkčního prostředí). Řešením, resp. zrychlením práce testovacího týmu, může být použití automatizovaných testů (vytvořených naprogramovaných testovacích skriptů), avšak z důvodu časové náročnosti přípravy těchto testů je někdy rychlejší otestovat produkt manuálně.

Posledním ze sedmi druhů plýtvání uvedených v (1) je doba potřebná k vyhledání odpovědí. LeanSD nepřikládá tak velký důraz na podrobnou dokumentaci v úvodní fázi projektu, ale návrh a funkcionalita je často vyjasňována až v průběhu vývoje, tzn. mnoho otázek

je pokládáno v průběhu celého vývoje a tyto otázky musejí být zodpovídány např. analytikem či zákazníkem. Je jasné, že plýtvání zde znamená dobu potřebnou pro programátora nebo testera někomu zavolat, přijít, zeptat se a vyřešit otázku.

Podle (1) existuje také druhý pohled na tento druh plýtvání. Představte si projekt, kde je dokument se zadáním odeslán zákazníkem k analytikovi, ten doplní své poznámky a poté po dohodě se zákazníkem odešle dokument programátorovi. Ten funkcionalitu naprogramuje, přidá do dokumentu své poznámky a dokument předá testerovi. V průběhu tohoto předávání vzniká plýtvání, jelikož část tacitní znalosti[13], kterou analytik, programátor či tester disponuje, nelze vždy zachytit a přepsat na papír. Tím vznikají otázky, které musí být následně zodpovězeny pro dostatečné pochopení např. nové funkcionality.

Vyřešení tohoto plýtvání je dodržení předpokladu zavedení LeanSD – lokace týmu v jedné místnosti.

Princip 2: Neustále přidávání kvality

Tímto principem se jednoduše myslí nepřetržité zlepšování kvality výsledného produktu. Tým, který se rozhodne aplikovat LeanSD by měl dodržovat i předpoklady, které LeanSD požaduje pro maximální efektivitu práce. Mezi ně patří neustálá komunikace se zákazníkem, nejlépe však jeho přítomnost na pracovišti. Zákazník zde hraje důležitou roli při zkvalitňování produktu, protože pravidelná kontrola ze strany zákazníka je nutná pro neustálé zlepšování produktu. Revize výsledného produktu není náhodná činnost, která je vykonána, kdy se zákazníkovi zachce, ale je mu pravidelně na konci každé iterace předkládána verze produktu, ke které se může zákazník vyjádřit a předložit další požadavky pro jeho vylepšení. Tento proces nepřetržité zpětné vazby je velice důležitý v každé agilní metodice a LeanSD není výjimkou. (25)

Pozice testovacího týmu hraje i zde významnou roli, protože produkt, který je předkládán zákazníkovi musí být funkční. Je jasné, že první verze nebudou disponovat příliš velkým počtem funkcí, ale na konci každého sprintu musí být předložen zákazníkovi fungující software, tzn. všechny vlastnosti a funkce produktu musí být kompletně otestovány. Testovací tým musí zajistit kapacity tak, aby samotné testování bylo dokončeno v termínu Testeři musejí mít na paměti, že si nelze naplánovat testování tak, aby skončilo poslední den

[13] *"Tacitní znalost je soubor dovedností, zkušeností, intuice, pravidel, principů, mentálních modelů a osobních představ konkrétního člověka"*(23)

iterace, jelikož se i v tento poslední den může vyskytnout chyba, která by již nemusela být opravena v termínu a zákazníkovi by nemusela být předložena funkční aplikace. Pokud by se jednalo o kritickou chybu blokující ostatní fungující části produktu, zákazník by nebyl vůbec spokojený.

Princip 3: Odložení rozhodnutí na později

Tento princip znamená odkládání důležitých rozhodnutí na co možná nejpozdější přijatelnou dobu. LeanSD radí, aby se týmy nerozhodovaly nad důležitými věcmi bez předložení jasných požadavků. Týmy se musí rozhodovat ne na základě nějakých předpovědí, jak by něco mohlo fungovat, ale na základě známých hodnot, ukazatelů nebo požadavků. (25)

Testovací tým, pokud si není zcela jistý, také nebude vytvářet komplikované automatizované testy, které pokrývají funkcionalitu produktu, který „by měl" tímto způsobem fungovat. Jak je vidět, i zde zákazník, resp. jeho zpětná vazba hraje důležitou roli.

Princip 4: Dodávat co nejrychleji

Tento princip vychází ze základu agilního přístupu, a to iterativního vývoje. Vývoj by tak měl být rozdělen do několika iterací (cyklů), kdy je na konci každé iterace předkládán produkt zákazníkovi ke kontrole. Důvod je jasný – pokud jakákoli softwarová společnost slíbí pravidelnou dodávku software pro poskytnutí zpětné vazby, společnost tímto slibem okamžitě získá konkurenční výhodu před ostatními společnostmi. Spokojen bude nakonec nejen vývojový tým, ale i zákazník, jelikož jeho požadavky se také v průběhu vývoje mohou měnit (a často se tomu tak děje) z důvodů stoupající konkurence, měnící se poptávky na trhu nebo z jakéhokoliv jiného důvodu.

Tento princip kompletuje princip třetí – odložení rozhodnutí na co nejpozději. Tím se myslí, že čím dříve softwarový produkt předložíme, tím později může tým učinit důležitá rozhodnutí, ke kterým se zákazník může po předložení produktu vyjádřit. (25)

Princip 5: Posílit postavení členů týmu

Tento LeanSD princip zdůrazňuje, že každý jednotlivý týmový člen je specialista na danou oblast při vývoji a měl by mít možnost prezentovat své nápady, připomínky a mít prostředí pro svou kreativitu a realizaci. Každý pracovník musí vědět, že i s jeho názorem se počítá a jeho názor je stejně tak silný a důležitý jako názor kteréhokoli jiného člena týmu. (25)

Koncept LeanSD počítá s maximální spolupráci každého člena týmu. Pro tento účel agilní přístup zavádí krátké každodenní schůzky týmu, kde každý člen týmu prezentuje, co udělal předchozí den, co bude dělat dnes a jestli mu v dnešní práci nebraní nějaké překážky (v metodice Scrum je tato každodenní schůzka pojmenována jako Scrum meeting). (26)

Princip 6: Integrace systému

Podle (1) existují dvě rozdílné integrity v systému:

1. První integrita je integrita „vnímání“, tzn. jak je produkt vnímán zákazníkem, jaký je jeho dojem z produktu, jestli splňuje nevyslovené požadavky, který by měl každý standardní systém mít, např. jestli nejsou tlačítka rozhozeny po obrazovce náhodně, jestli vyhledávač zobrazí relevantní výsledky v přehledném seznamu a podobně.

2. Druhá integrita je „konceptuální“, tzn. jak systém pracuje jako celek, např. jestli jsou některá pole na každé stránce přeložena jinak, i když znamenají přesně totéž.

Princip poukazuje na důležitost komunikace napříč týmem, každý člen týmu by měl být informován o změně v systému, která může ovlivnit a často také ovlivňuje práci ostatních.

Princip 7: Optimalizace celku

Každý systém je vytvořen z mnoha částí a každý jeden software je propojen mnoha typy technologií. Podle (4) by se vývojový tým neměl soustředit pouze na optimalizaci každé jedné části zvlášť, ale optimalizovat a měřit tak celek, který zákazník nakonec vidí a kontroluje. Například, pokud je část programu pro sestavení různých reportů optimalizována na všechny možné druhy vstupních dat a také kompletně otestována, neznamená to, že druhá část programu odpovědna za odeslání vstupních dat funguje také správně. Zákazník tak nebude moct otestovat všechny části systému.

Tento princip by si měly osvojit i testovací týmy a pamatovat, že software není jen jedna část aplikace nebo jedna nová funkcionalita, která shodou okolností zrovna funguje. Testeři by neměli opomenout na to, že software je složen z mnoha částí, které mezi sebou komunikují. Měly by být vytvořeny integrační testy, které zajistí, že je aplikace schopna komunikovat s ostatními částmi systému, například s webovými službami nebo jinými systémy, které ve společnosti již fungují.

4.2.4 Pozice testování v Lean software development

Pozice testování je už z přístupu agilních metodik k vývoji vcelku jasná. Testování probíhá průběžné s vývojem a programováním zdrojového kódu. Vše se odvíjí od potřeby mít vždy funkční zdrojový kód, který lze spustit a průběžně po každé iteraci informovat zákazníka o stavu vývoje jeho softwaru v podobě demo aplikací.

Iterativní vývojový cyklus

Agilní a také i LeanSD přístup znamená vyvíjet software postupně v malých inkrementech[14], což je označení většinou pro jednu funkcionalitu nebo jeden menší úkol, kterou dokáže naprogramovat obvykle jeden vývojář, tester nebo jiná role v týmu typicky v průběhu jednoho až čtyř dní. V nulté iteraci, což je označení pro úplně první cyklus vývoje se definují prioritní úkoly a každý se vyjádří k úkolu (inkrementu) a určí se jeho časová náročnost. Tyto úkoly jsou přidány do seznamu úloh, tzv. Backlog. Poté si každý člen týmů vybírá pro sebe úkol z tohoto Backlogu – většinou se jedná o výběr mezi vývojáři a rozhodnutí se, kdo na kterém úkolu (funkcionalitě apod.) bude pracovat. Výběr bývá dobrovolný, tím se vyruší nespravedlivé rozdělování úkolů. Tým si v této fázi musí uvědomit, že vybrané úkoly musejí být splněny do konce iterace. Backlog je logicky upravován na začátku iterace. (4)

Iterace je časové rozmezí, kdy jsou přidávány do produktu nové funkcionality nebo vlastnosti. Každá tato iterace trvá obvykle několik týdnu, avšak ne déle než jeden měsíc. Iterace je poté zakončena předvedením verze produktu (demo verze nebo spustitelné verze produktu), která obsahuje některé funkce a může být předvedena zákazníkovi. Znamená to, že všichni členové týmu musejí být se svou práci na úkolu hotovi.

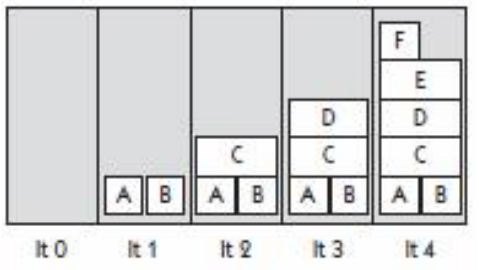
V rámci jedné iterace vývojář musí vytvořit kód, tester musí novou funkcionalitu otestovat, nový kód poté musí být nasazen do první nebo další verze produktu avšechny tyto činnosti musejí skončit před koncem iterace.Harmonogram musí být dobře naplánován a odhady časové náročnosti inkrementů musejí počítat s časovou rezervou v případě výskytu defektů nebo jiných negativních situací. Testovací tým zde hraje nemalou roli ve stanovení časové náročnosti úkolu, resp. jeho časové náročnosti potřebné k otestování.

[14] Inkrementy jsou také někdy označovány jako přírustky. Každý jeden inkrement představuje jednu samostatně realizovatelnou část produktu. (30)

Pokud si vzpomeneme na Vodopádový model, tak zjistíme, že práce na jednom inkrementu (funkcionalitě, vlastnosti produktu) probíhá podle tohoto modelu. Jak uvádí (28), *„Každá iterace = malý vodopád.*

- *první iterace odhalí největší rizika*
- *každá iterace produkuje spustitelnou verzi*
- *každá iterace obsahuje integraci a testování"*

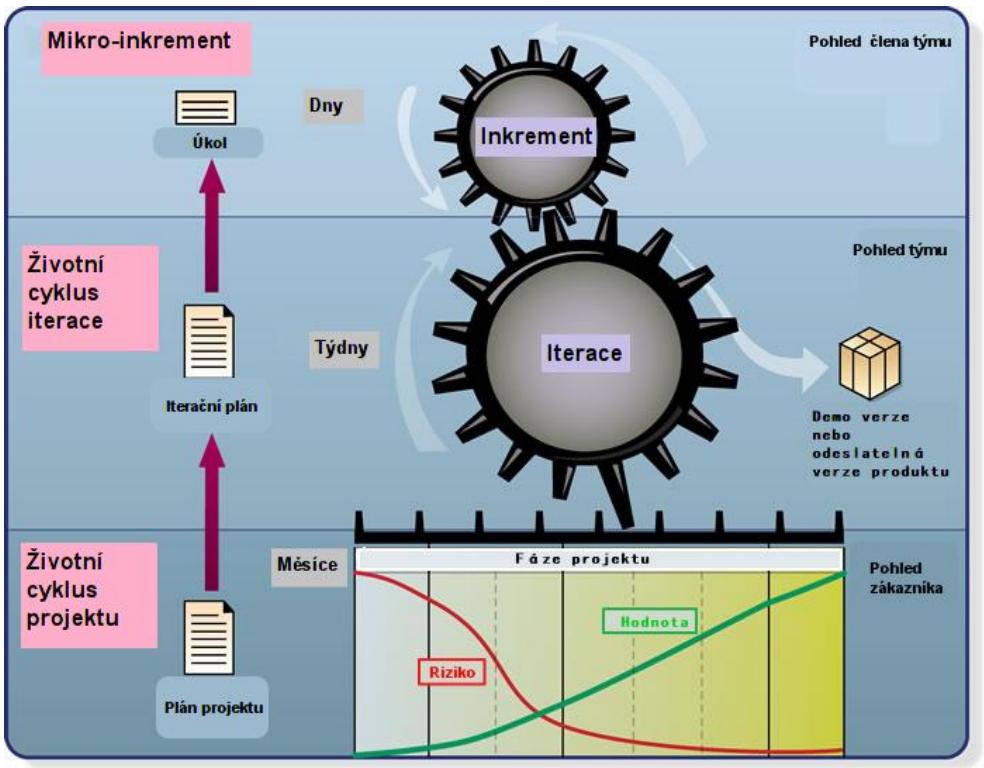
Agilní přístup je iterativní a inkrementální, to znamená, že testovací tým kontroluje každý přírůstek kódu, či nově vytvořenou funkcionalitu okamžitě po jejím vytvoření. Pro lepší představu si lze inkrementální vývoj představit jako následující obrázek.



12 Obrázek - Ukázka inkrementů v iterativním vývoji (30)

Písmena A až F znázorňují inkrementy. Lze vidět, že každou další iterací je přidán nový přírůstek a aplikace je tak vyvíjena po částech v krátkých časových úsecích.

Následující obrázek graficky znázorňuje iterativní vývoj software.



13 Obrázek - Harmonogram iterativního vývoje (autor, podle (29))

Manažerský pohled by navíc do obrázku doplnil pravidelné schůzky týmu, které probíhají obvykle každý den ráno na asi 15min, kdy se sleduje postup práce. Manažer projektu by ještě mohl doplnit pravidelnou schůzku se zákazníkem na konci každé iterace, kde je produkt předveden a předán k vyzkoušení, ačkoli agilní vývoj předpokládá častější komunikaci se zákazníkem, nejlépe jeho osobní přítomnost na pracovišti.

4.3 Testovací tým podle Lean software development

Testovací tým, který se rozhodne provést změnu ve svém testovacím procesu a přijmout koncept Lean software development (dále jen LeanSD) vycházející z agilního přístupu, musí také splňovat specifické požadavky pro agilní přístup. Tyto požadavky jsou již řadu let definovány v Manifestu agilního vývoje[15] a zásadně upravují způsob jak vývoje software, tak i průběhu testování. Typickým příkladem je změna vývoje pomocí krátkých iterací (v cyklech) v každém agilním projektu. V každé takové iteraci musí proběhnout analýza, vývoj, testování a nasazení nové funkcionality.

[15] Více o Manifestu agilního vývoje na www: http://agilemanifesto.org/iso/cs/

Tento agilní přístup může pro mnoho týmů znamenat počáteční šok. Typickou otázkou těchto týmů je: jak můžeme správně definovat požadavky, otestovat a předat zdrojový kód v průběhu jednoho, dvou nebo čtyřech týdnů? Podle LeanSD je odpovědí aplikace LeanSD principů, úzká komunikace se zákazníkem, zaměření se na to, co zákazník opravdu požaduje, a předání produktu s maximální hodnotou.

4.3.1 Úloha testovacího týmu podle Lean software development

Testovací tým se pravidelně účastní každodenních schůzek a pomáhá s vytvářením backlogu[16] (seznamu úkolů) a s rozhodnutími, jak který úkol (inkrement) bude časově náročný na testování. Tým si také musí uvědomit posloupnost úkolů – některé části produktu nemohou být vyvinuty dříve, než bude vytvořena jiná část. Důležité jsou také definice rizik spojených s vývojem, např. schopnost projekt vytvořit v konkrétním čase a stanoveném rozpočtu. V neposlední řadě by si měl tým stanovit, co všechno bude v průběhu projektu dokumentovat. V rámci aplikace LeanSD by měl tým předejít zbytečnému plýtvání a stanovit si, které vytvoření dokumentů je opravdu důležité.

V nulté iteraci (prvním cyklu) vývoje software jsou se zákazníkem definovány požadavky a následně jsou testovacím týmem vytvářeny testovací případy, které slovně popisují průběh testu, resp. vytváří se kroky nebo akce, které vykoná uživatel, aby dosáhl naplnění konkrétní funkce aplikace. Vytvoření takových testovacích případů pomůže testerům uvědomit si, jak by tato funkcionalita měla fungovat konkrétně v aplikaci, a pomůžou tak s definováním specifických otázek. Otázky následně zodpovězené zákazníkem vyjasní nepřesná zadání a tím se zvýší hodnota produktu a zákazník bude potěšen.

Podle (31) je důležité, aby někdo z testovacího týmu napsal testovací případy ještě dříve, než je vůbec napsán jediný řádek zdrojového kódu. Dobrý příklad uvedla jedna z autorek knihy (31), která radí, aby si testovací tým uvědomil kritickou cestu[17] nezbytné funkcionality, která se bude v aktuální iteraci vyvíjet. Tato cesta by se měla v rámci jedné funkcionality naprogramovat jako první a měla by být otestována jako první, jelikož je mnohem lepší

[16] Backlog je dynamický dokument obsahující seznam požadovaných vlastností, omezení, potřebných nástrojů. Obvykle je určen jeden z členů týmu, který se stará o aktualizaci dokumentu a upřednostňování práce na prioritních úkolech z backlogu podle zákazníkových požadavků. (4) Označení Backlog se používá nejčastěji v metodice SCRUM.

[17] Kritická cesta funkcionality v softwarovém produktu je vykonání základních a nezbytně fungujících kroků, které uživatel musí mít možnost provést, aby software na uživatelskou akci vrátil požadovanou reakci v rámci jedné funkcionality. Kriticka cesta je pak posloupnost těchto kroků. (autor, podle (34))

zákazníkovi předvést jádro funkcionality, která funguje (bez kosmetických úprav[18]), než kdyby fungovala jen z poloviny nebo dokonce vůbec.

Testovací tým nesmí zapomenout na vytvoření testů, kterými zajistí požadovanou výkonnost produktu například tak, že software bude stabilní při připojení stovek uživatelů v jeden moment a podobně

Každý projekt, tým a někdy i každá iterace je odlišná a to, jak tým řeší problémy, závisí na konkrétním problému, lidech a používaných nástrojích. Proto by měl být každý člen agilního týmu schopný adaptovat se na potřeby týmu.

Pravděpodobně nejviditelnějším rozdílem v agilním projektu oproti tradičnímu je v rychlé zpětné vazbě testovacího týmu, která táhne projekt dopředu, a neexistence žádných blokujících faktorů, které brání v postupu projektu, pokud předem dané milníky nebyly splněny. To je také jeden z faktorů, proč se týmy bojí přijmout agilní přístup ve svých projektech. Týmy se bojí, že nastane chaos z důvodu slabé disciplíny nebo nedostatku dokumentace a slovo „agilní“ je pro ně pouze buzzword[19], které zjednodušeně znamená, že každý si může dělat, co chce. To ovšem není pravda, skutečné agilní týmy jsou opravdu komunikující týmy v neustálém kontaktu. Tým není rozdělen na oddělení, ale každý člen týmu je ve své podstatě odpovědný za doručenou kvalitu software. Každý člen je také odpovědný za kvalitu – podle (30) *„každý člen agilního týmu je tester“*, což znamená, že každý člen týmu může ověřit kvalitu produktu, z jiného úhlu pohledu. (30)

4.3.2 Kdo je tester podle Lean software development

Tester, který dokáže přijmout změny, spolupracuje dobře s technicky zaměřenými lidmi i se zákazníkem (business people), rozumí konceptu vytváření testů z dokumentů s požadavky a řídí vývoj, je podle (30) schopen obstát v testovacím týmu postupujícím podle LeanSD. Tito testeři by měli také disponovat dobrými technickými znalostmi, vědět, jak spolupracovat

[18] Kosmetická úprava je např. vzhled aplikace nebo grafická úprava, resp. doplňky (barvy tlačítek nebo vabraný font písma), které s funkcionalitou přímo nesouvisí, ale zvyšují např. přehldenost nebo orientaci v produktu. (autor)

[19] Buzzword je popisován mnoha definicemi, avšak podle mého názoru se nejlépe hodí tato podle (64): důležitě znějící obvykle technické slovo nebo fráze s nevelkým významem používající se hlavně k zapůsobení na laiky.

s ostatními členy testovacího týmu při automatizaci testů, a také mít zkušenosti jako průzkumný tester[20] se snahou vždy porozumět a pochopit požadavkům zákazníka.

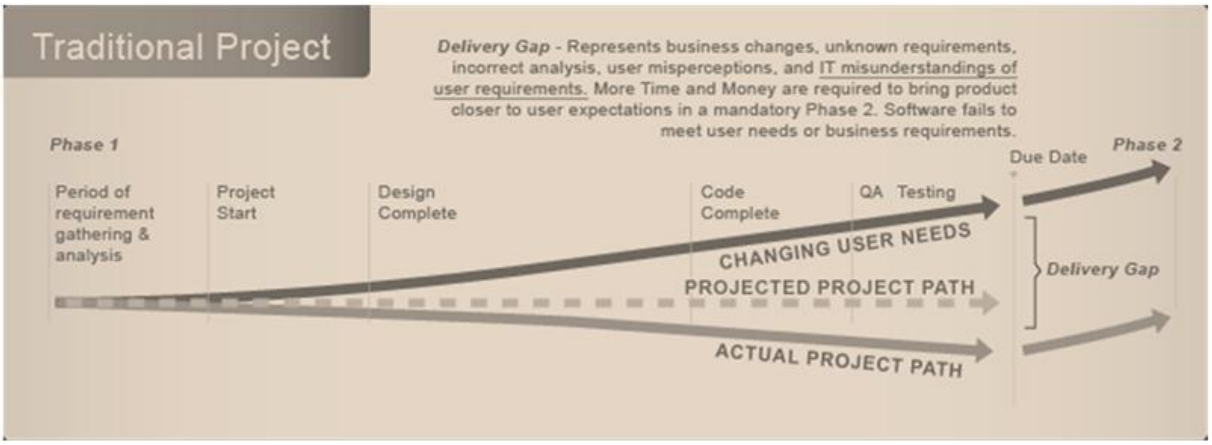
Existují principy, které by měl agilní tester splňovat. Zde jsou principy definované podle (30):

- průběžně poskytovat zpětnou vazbu;
- přinášet hodnotu zákazníkovi;
- být schopný osobní komunikace;
- mít odvahu;
- pracovat jednoduše;
- praktikovat průběžné vylepšování;
- být schopný reagovat na změnu;
- být samo-organizující;
- být zaměřený na lidi;
- mít zalíbení ve své práci (práce ho baví).

4.4 Rozdíl oproti tradičnímu přístupu

V tradičním přístupu vývoje software je hlavní pozornost skupině funkcionalit, které jsou stanoveny na začátku projektu, a poté je ve většině případů prezentován jeden produkt na konci vývoje. Harmonogram takového tradičního vývoje může pomoci vysvětlit následující obrázek. (8)

[20] Průzkumné testování (z aj. Exploratory testing) je návrh testů a exekuce zároveň. Je to opak skriptovacího testování (kdy jsou definované kroky před exekucí testů). Tyto testy nejsou nijak připraveny předem a nejsou součástí přesně definovaného plánu. (33)

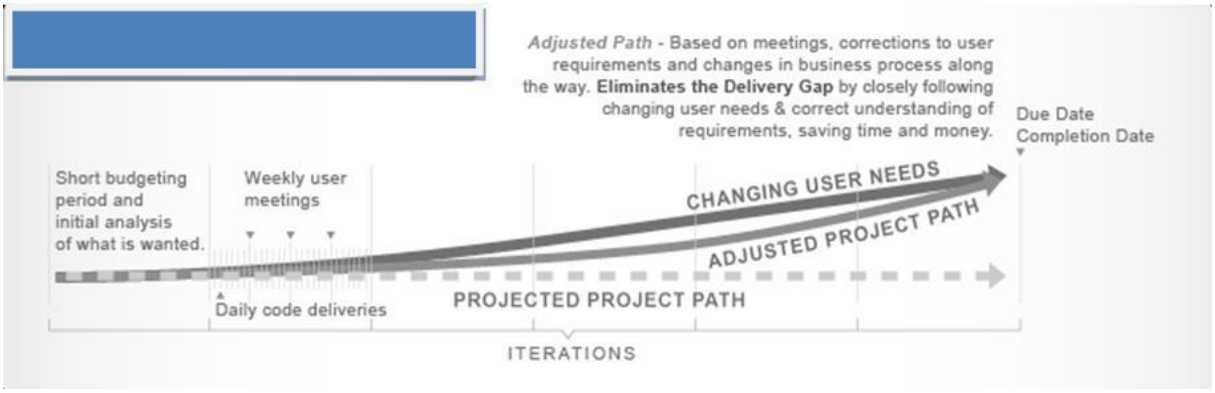


14 Obrázek - Vývoj podle tradičního přístupu (54)

Z obrázku je vidět, že v průběhu vývoje systému se postupně měnily požadavky zákazníka, jeho aktuální potřeby na produkt a funkcionality, o kterých během návrhu třeba nevěděl, ale aktuálně je požaduje do produktu implementovat. Tento rozdíl oproti plánu projektu zobrazuje horní křivka, která jde mimo navrhované požadavky.

Navíc jak se běžně stává, během plánování a návrhu systému může dojít k nečekaným nedorozuměním a systém je vyvíjen s nepřesnými vlastnostmi – prostě tak, jak si vývojový tým myslel, že má aplikace fungovat. Tuto nepřesnost a nedorozumění během vývoje zobrazuje spodní křivka, která jde také mimo linku naplánovaného projektu. Rozdíl v aktuálním projektu oproti potřebám zákazníka znázorňuje rozdíl mezi vrchní a spodní linii. Rozdíl to není zanedbatelný. (8)

Oproti tomuto přístupu a vylepšení procesu vývoje bojuje agilní přístup, který lze popsat pomocí následujícího obrázku.



15 Obrázek - Vývoj podle agilního přístupu (55)

V agilním vývoji se vybere a naplánuje vývoj, otestuje a nasadí jedna funkcionalita (v její nejjednodušší podobě) předtím, než se vybere funkcionalita další. V rámci LeanSD konceptu

díky tomuto průběhu plníme jeho hlavní princip a tím je eliminace plýtvání: tým aktuálně nepracuje na funkcionalitě, která není aktuálně nezbytná a nepřináší hodnotu zákazníkovi. Díky tomu jde aktuální křivka projektu ruku v ruce s křivkou měnících se požadavků zákazníka. (8)

4.5 Výhody

Každá technika nebo koncept vývoje znamená změnu v týmu, v procesu vývoje, testování, ale i řízení týmu, který se pro takovou změnu rozhodne. Jako každá nová věc s sebou přináší mnoho výhod a několik nevýhod. V následujícím textu představím několik takových vlastností konceptu Lean software development. Jak výhody, tak nevýhody logicky vycházejí z principů LeanSD.

Využití principu eliminace plýtvání vede ke zvýšení efektivnosti celého vývojového procesu. Samozřejmě také ovlivňuje testovací tým, který se zaměřuje na vytváření skutečně potřebných testů a neplýtvá časem nebo jinými zdroji. Zároveň se tak zkracuje doba potřebná k dodání výsledného produktu, na kterou navazuje další výhoda v LeanSD: dodávka výsledného produktu je v LeanSD jako u ostatních agilních metodik velmi častá a tím, že se testovací tým soustředí na skutečně důležité testy, je tým schopný doručit výsledek testů co nejdříve a zkrátit tak vývojový cyklus.

Další princip radí, aby se testovací tým soustředil na aktuální problémy a k nejasným požadavkům zaujal postoj v tom smyslu, odložit tak rozhodnutí na co nejpozději. To pro tým testerů znamená, aby nevytvářeli zbytečné testovací scénáře, když ani sám zákazník si často není jistý, co vlastně chce. Tím se zamezí plýtvání zdroji testerů a naopak jejich přenesení na skutečně důležité věci.

LeanSD očekává od testovacího týmu, že maximum testů bude zautomatizovaných, čímž je možno dosáhnout časové úspory proti vykonání testů manuálně. Princip neustálého zvyšování kvality nutí testovací tým stále přidávat další automatizované testy a přemýšlet o krocích vedoucích k urychlení procesu testování produktu.

Každý člen v testovacím týmu má stejnou sílu hlasu, v LeanSD neexistují rozdíly mezi členy týmu a jejich postavením – každý se může vyjádřit. Tým a komunikace uvnitř týmu je v LeanSD na velmi důležité pozici. Jsou odbourány mezilidské bariéry, každý je na schůzkách vyslyšen. Motivace každého člena týmu se tak zvyšuje, a proto i lépe pracuje.

Pravidelné schůzky probíhají v LeanSD mnohem rychleji a jednoduše zabraňují nedorozuměním, zbytečné práci nad nevyžádanými změnami a nad zbytečnými testy. Na schůzkách je často přítomen zákazník a pomáhá s vývojem, tak i definuje požadavky na systém, které by měl produkt splňovat, které musí poté být otestovány. Lepší komunikace je tak jednou z mnoha výhod, které LeanSD s sebou přináší.

4.6 Nevýhody

LeanSD je aplikovatelný spíše pro menší týmy, kde závisí vývoj projektu na jednotlivých členech týmu, po kterých se očekává splnění svých závazků kvalitně a včas. Je proto nutné nalézt dostatečně zkušené testery, kteří jsou schopni sami naprogramovat jednoduché, ale i složitější automatizované testy, mají zkušenosti s několika nástroji a z lidského hlediska by to měli být pro-aktivní osobnosti, schopni hovořit s programátory i zákazníkem o všech problémech, které nastanou nebo kterých se obávají.

Na kvalitě testerů si zakládá například i firma Microsoft, kde podle (2) testeři nejsou jen lidé, kteří umějí pouze testovat a znají jen testovací nástroje, ale jsou schopni umět porozumět zdrojovému kódu nebo i programovat. Z toho důvodu se testerům v Microsoftu v překladu říká: softwaroví vývojáři se zaměřením na testování (software design engineer in test). Celkově tedy výsledek a schopnosti jednotlivých členů testerů jsou závislé na kvalitě výsledného produktu.

Když jsem v kapitole 3.2.3 psal o principech LeanSD, tak jeden z nich byl – odložit rozhodnutí na později. Tento princip lze u některých projektových manažerů, například podle (58) chápat jako nevýhoda, jelikož zákazník není nucen rozhodnout o důležitých rozhodnutích hned na začátku vývoje, jak je tomu u tradičních metodik, ale odkládá své rozhodnutí na později. Nevýhoda pro testovací tým je samozřejmě následný nepřesný návrh testovacího plánu a rozdělení testerských kapacit.

LeanSD nedává příliš velký důraz na rozdělení rolí v týmu, z toho může vzniknout problém, jelikož pokud tým nemá vyhrazeného některého z členů tým v roli business analytika, který nemá dostatečné zkušenosti, problém s vysvětlením zákazníkových požadavků může být pro celý nejen testerský tým obtíž. (58)

4.7 Shrnutí

Každá metodika, koncept nebo technika má své výhody a nevýhody. LeanSD převážně neradí, který nástroj a jak použít, jsou to spíše rady, jak se co nejvíce vyvarovat chyb a jak co nejrychleji zefektivnit a zrychlit vývoj software. Bohužel, každý přístup je zbytečný, pokud v týmu nejsou žádní experti nebo zkušenější členové, kteří jsou schopni správně a rychle rozhodnout. Pravděpodobně proto koncept LeanSD nedefinuje žádné role v týmu, ale přenechává toto rozhodnutí na samotných členech. Myslím si, že tento koncept dobře radí testerům, jak optimalizovat efektivitu testovacího týmu.

5 Aplikace principů Lean software development pro testovací tým

Kapitola 3.2 představila koncept Lean software development jako celek a jeho spojení s agilním přístupem, proto se v následujícím texu věnuji tomu, jak a s jakými výsledky byla provedena aplikace tohoto konceptu v testovacím týmu, ve kterém aktuálně pracuji.

5.1 Výchozí situace

Fima, ve které nyní pracuji, má celý svůj vývojový tým rozmístěn mezi pobočkami napříč celým světem. Náš testovací tým testuje pouze část vývoje, do které patří hlavně programátorský tým v Praze, Belgii a Francii, proto odpadají problémy spojené s prací ve více časových pásmech (např. v případě programátorů Spojených státech).

5.1.1 Použité technologie

Předmětem testování je interní aplikace pro podporu hlavní činnosti firmy, kterou je prodej životního pojištění a plánů zaměstnaneckých benefitů společnostem pro jejich plánování, koordinaci a evidenci. Jedná se o webovou aplikaci postavenou na technologii ASP.NET. Aplikace komunikuje s databází (Microsoft SQL Server) a dalšími datovými zdroji používanými pro zajištění snadného dohledání klientů napříč firmou. Každá webová stránka obsahuje zdrojový kód několika programovacích jazyků pro zobrazení dynamického obsahu. Jedná se o základní kód ASP, pak Visual Basic, Javascript a další jazyky podporované technologií .NET. Logika aplikace je naprogramována v C#, další části pak pomocí aplikace InRule nebo pomocí příkazů na databázové úrovni. Některé funkce aplikace zajišťují samostatné webové služby. Část aplikace je propojena s BPM aplikaci pro zajištění specifické funkcionality (pro zjednodušení práce a také pro sledování statistik práce koncovými uživateli).

Celý tým používá vývojové prostředí Visual Studio 2010 s vybranými add-iny.

5.1.2 Průběh vývoje software ve firmě

Poslední verze aplikace v1 byla vyvinuta pomocí tradičního vodopádového modelu. Na začátku byl naplánován harmonogram, který byl vymyšlen tak, aby nová verze byla dostupná koncovým business uživatelům, tedy našim klientům, v termínu 10 měsíců.

První fáze byla přípravná a klienti kladli své požadavky na vylepšení aplikace svým nadřízeným, ti následně své požadavky rozebírali s projektovými manažery a hlavními vývojáři. V naší firmě bohužel neexistovala analytická role. Během jednoho měsíce vzniklo nespočet dokumentů a návodů s mnoha revizemi a detailními požadavky. V tento okamžik se hromadily informace na několika místech, komunikace probíhala zejména přímo mezi klienty a programátory, testovací tým nebyl v této plánovací části vývoje vůbec zahrnut. Poté, co již byly požadavky v přijatelném stavu sepsány, mohla začít práce programátorů a testerů.

Vývoj aplikace postupoval tradičním způsobem, aplikace po vytvoření byla předána testerům na testování. Počet testerů zdaleka nedosahuje potřebného počtu, a tak o vytvoření komplexních automatizovaných testů nemohla být ani řeč. V průběhu vývoje a poté i během testování se stalo očekávané – klient přidal nové požadavky, takže funkcionalita, která byla v pořádku připravena a dokonce i ta, která nefungovala a byly na ni vytvořeny defekty, musela být přeprogramována. Vývoj se automaticky opožďoval každým novým požadavkem, navíc defektů se našlo mnoho a s každým novým požadavkem ještě více.

Náš testovací tým testoval každou funkcionalitu bohužel jen jednou, větší regresní testy nebyl čas provádět manuálně, natož je zautomatizovat. Po našem testu byla aplikace předána klientům k dalšímu testování pro kontrolu business pohledu, resp. začal akceptační test. Tento test probíhal také mnohem déle, než bylo původně myšleno.

5.1.3 Pozice testování

Náš testovací tým nebyl příliš velký, obsahoval pouze dva členy. Podporovali jsme deset vývojářů z několika zemí. Všechna komunikace probíhala v angličtině, včetně komunikace s klienty v různých zemích a časových pásmech.

Harmonogram testování

Každý harmonogram začíná plánem a i náš testovací plán byl navržen, avšak kvůli absenci kvalitního nástroje pro sdílení informací ve firmě, naší nepřítomnosti při zadávání požadavků a také slabé komunikaci s programátory jsme měli problém vytváření plánu dokončit a nechat si plán schválit. Dle tradičního modelu měli programátoři vlastní plán skončit svou část práce na nové funkcionalitě a předat ji k testování. Neexistoval prostor pro prioritizaci jednotlivých testovacích scénářů, časový rozpis testů nebo naplánování vytváření a spouštění automatizovaných testů. Náš testovací proces byl zpožděn o necelý měsíc a odhaduji, že jsme

z časových důvodů svými testy nepokryli asi 30% nové funkcionality s vědomím, že stávající funkcionalita nebyla otestována vůbec.

V průběhu vývoje nových funkcí byly testerům k dispozici změnové požadavky a požadavky na nové funkce aplikace. Některé byly vyplněny nedostatečně, a tak musela probíhat zdlouhavá komunikace a vznikaly velké časové prodlevy. Přesto v tomto období náš testovací tým vytvářel testovací scénáře (test cases), které sloužily jako příprava k exekuci testů. Účelem vytváření testovacích scénářů je seznámit se s novými funkcemi a pochopit jejich smysl.

Poté, co programátoři odevzdali svou aplikaci k testování, probíhalo samotné testování – funkcionální, převážně manuální, podle našich vytvořených scénářů. Pokrytí aplikace pomocí pouze tohoto testování je podle mého názoru nedostatečné a velice časově náročné.

Když jsme s testováním skončili, mnoho defektů bylo nevyřešených, ovšem již v té době jsme s testováním byli necelý o měsíc pozadu a programátoři nestíhali opravovat své defekty. Přesto byl zdrojový kód nasazen na prostředí určené pro akceptační testy.

Klienti našli nemalé množství chyb, jejich počet se pohyboval v desítkách a dlouhou dobu se nesnižoval, jelikož oprava jednoho defektu trvala déle než objevení jednoho nového defektu. Tímto způsobem se testování na straně klienta zpozdilo mnohem více. Celkově se projekt prodloužil o 50% na konečných 15 měsíců s tím, že některé části aplikace nebyly nasazeny vůbec.

Používané druhy testů

Testovací tým v průběhu vývoje používal převážně funkcionální manuální testování, ale snažil se některé testy zautomatizovat. Byly to testy, které po částech procházely každý odkaz zobrazující se na stránce, a testovaly, zda požadovaná stránka byla skutečně nalezena. Zabránili jsme tak nepřijemné situaci, kdy ne že by aplikace nedělala přesně to, co měla, ale tomu, že vůbec byla použitelná a mohla tak být předána klientům ke kontrole. Tento test byl spouštěn denně a několikrát objevil chyby. Takovýchto testů jsme stihli udělat několik, mezi nimi také několik regresních testů, které testovaly standardní operace, denně používané, například vytvoření klienta, vytvoření produktu nebo úpravy kontaktů a podobně. Pokud bych měl zařadit naši pozici testování, tak se jednalo nejvíce o spouštění testů z kvadrantu Q2 (viz kapitola 3.4).

Naše testování probíhalo v rámci kvadrantu Q2 a Q3 a akceptačního testování klienty, se kterými jsme byli poté denně v kontaktu a řešili problémy; testovací tým se v této fázi stal spojovacím článkem mezi klienty a programátory.

Náš testovací tým očividně již neměl kapacity pro výkonnostní testování aplikace. Tento druh testování nakonec byl zadán testovacímu týmu v USA.

Nástroje

Dokumenty s novými a změnovými požadavky od klientů byly ukládány v JIRA, což je mj. dle (58) nástroj pro sledování projektů, týmové plánování, vytváření a start velkých projektů. Tento nástroj umožňoval vytvářet úlohy – každý takový úkol byl poté přidělen programátorovi a připojen byl také dokument s požadavky. Přístup do JIRA měl i zákazník a mohl vkládat nové verze dokumentů, podle kterých probíhal následný vývoj a testování. Následná komunikace byla většinou přes emailové zprávy a dotazů bylo velmi vysoký počet.

Bok po boku k systému JIRA existoval Team Foundation Software (TFS), který byl určen pouze pro vývojový tým pro rozdělení práce a kde byly také diskutovány technické otázky a vkládány všechny defekty nalezené testovacím týmem. Svou funkcionalitou se dá označit jako konkurenční program k JIRA.

Jako vývojové prostředí se používalo Visual Studio (VS), ve kterém byla vytvářena logika webové aplikace a kde byly programátory vytvářeny unit testy. Testeři měli k těmto testům přístup, avšak na jejich přípravě se nepodíleli.

V době testování nebyl určen ani koupen žádný testovací nástroj, a tak po domluvě v interním týmu jsme se rozhodli použít pro automatizované testy open source nástroje. Prvním z nich byl framework WatiN pro testování webových aplikací na technologii .NET. Zdrojový kód testů byl v C#, avšak vytváření testů bylo velmi pomalé. Framework nedisponoval žádným grafickým test recorderem, takže zdrojový kód musel být psán ručně. Výhodou tohoto postupu byl alespoň fakt, že tester přesně věděl, co od testu může čekat, a prováděl přesně to, co tester chtěl. Práce s tímto frameworkem byla časově náročná, proto těchto automatizovaých testů nebylo mnoho a nepokrývaly dostatek nových i stávajících funkcí aplikace.

5.1.4 Shrnutí

Předchozí text čtenáře uvedl do výchozí situace vývojového týmu, použití softwarových nástrojů, ale zejména popsal pozici testovacího týmu během vývoje aplikace. V následujícím

textu v bodech zrekapituluji nedostatky a problémy vyskytující se během vývoje pro testovací tým.

1) Nedostatečná komunikace s ostatními členy vývojového týmu

Testovací tým nebyl přítomen během schůzek při sestavování projektového plánu, v průběhu plánování neměl žádné slovo a nebyl ani pozván ke schůzkám s klienty ani projektovým managementem. Testeři se nemohli vyjádřit k časové náročnosti testování a ovlivnit tak celý harmonogram vývoje. Testovací tým neznal náročnost nových funkcionalit a nebyl dostatečně připraven.

2) Neexistující plán testování

Tento problém vychází z předchozí nedostatečné komunikace a zapříčinil tak nemožnost testovacícho týmu rozvrhnout lidské zdroje ke konkrétním testům. Nebyl navržen žádný testovací plán. Nikdo z testerů až do poslední chvíle nevěděl, na čem bude pracovat. Příprava testerů tak nebyla dostatečná. Harmonogram práce testovacího týmu nebyl vytvořen, avšak ze strany managementu ani nebyl požadován.

3) Nedostatečná podpora testování

Testovací tým oficiálně neměl stanoven žádné testovací nástroje dodány zaměstnavatelem, podpora managementu v poskytnutí těchto nástrojů byla nulová.

Následující tabulka zobrazuje efektivitu používaných nástrojů pro konkrétní testy, které náš tým v průběhu vývoje vytvářel a spouštěl.

Typ testu	Nástroj	Rychlost psaní testů	Jednoduchost používání	Vhodnost použití	Efektivita celkem
Integrační testy	NUnit	90%	70%	80%	80%
Funkcionální testy	WatiN	30%	20%	30%	40%
GUI testy	WatiN	40%	30%	70%	70%

2 Tabulka - Výchozí nástroje v testovacím týmu (autor)

Tabulka napovídá, že aktuální situace s dostupnými nástroji není dobrá. Efektivita nástrojů by se měla blížit, ne-li rovnat 100%, ovšem tak tomu nebylo. Již tak omezená kapacita lidských zdrojů v testovacím týmu se musela potýkat s nevhodnými nástroji. Testovací tým neměl kapacity ani nástroje pro vytváření ostatních druhů testů.

Document Management System[21] (DMS) byl nedostatečný, vznikalo mnoho různých dokumentů s různými verzemi lokálně[22]. Testovacímu týmu nebyly vždy doručeny aktuální dokumenty.

Existovaly dva různé nástroje pro sledování projektu, byl to systém JIRA a TFS. Vznikaly duplicity ve vytvořených úkolech a zmatky v jejich přidělování konkrétním lidem. Oba nástroje měly různě definované workflow[23], tudíž postup předávání úkolů a defektů byl v obou nástrojích odlišné.

4) Nedostatečná kapacita testovacího týmu

Testovací tým je složen pouze ze dvou členů, kteří navíc nebyli u zrození prvních verzí aplikace, kdy testovací tým neexistoval vůbec. Aplikace se tak postupně stala příliš složitou pro vytváření automatizovaných regresních testů ve dvou lidech. Prioritní bylo zaměření na test nových funkcí aplikace, tím pádem nedostatek lidských zdrojů v testovacím týmu omezil automatizaci testů.

V následující kapitole navrhnu řešení problému, se kterými se testovací tým pravidelně potkával a které vedly k celkovému prodloužení doby testování produktu.

5.2 Zavedení principů Lean software development v našem testovacím týmu

Předcházející kapitola napověděla, že testovací tým nebyl ve své práci efektivní jak z pohledu použitých softwarových nástrojů, tak z pohledu řízení celého týmu, ale také ve spolupráci s ostatními členy vývojového týmu. Definoval jsem 4 nejčastější problémy, které se v průběhu testování stále objevovaly.

---

[21] Document Management System (DMS), přeloženo jako systém pro řízení dokumentů v organizaci. DMS je navrhnut od základů pro podporu celé organizace, která požaduje řídit tvorbu, vyhledávání a archivování. DMS není adresářová struktura, jak tomu bývá na klasických počítačích, ale obsahuje tzv. centrální úložiště, kolem kterého se všechny dokumenty točí a kde jsou uchovávány libovolné typy informací. DMS také chrání data před ztrátou. (59)

[22] Lokálně v tomto kontextu znamená, že každý uživatel pracuje s dokumentem uloženým na svém vlastním počítači a ostatním uživatelům tak není k dispozici aktuální verze. Takový uživatel musí po provedení změň dokument uložit do firemní počítačové sítě. Nevýhoda této lokální práce je, že dva a více uživatelů může pracovat na stejném dokumentu najednou, avšak o sobě nemusí vědět a vznikají tak duplicitní dokumenty a verze dokumentů ve firemní počítačové síti. Oprava takových dokumentů je časově náročná. (autor)

[23] Workflow je termín používáný k popisu kroků vykonávající organizace, konkrétní lidé, nástroje. Každý takový krok vyžaduje vstupní a výstupní informace. Posloupnost těchto kroků vykonává nějaký business proces, kdy je vstup transformován na výstup. (60)

Tato práce se věnuje aplikaci konceptu Lean Software Development pro testovací tým, proto se v následujícím textu věnuji přijetí principů LeanSD a definování požadavků, které jsou po týmu vyžadovány a představím výstupy a očekávána zlepšení v procesu testování, kterých lze aplikací LeanSD dosáhnout. Podle mého názoru správné přijetí těchto principů je klíčové pro efektivní řízení testovacího týmu a pro zajištění maximálního pokrytí produktu testy.

V následujícím textu opět uvedu 5 základních principů LeanSD, ale aplikuji jejich význam pro náš testovací tým tak, abych eliminoval nedostatky, které jsem uvedl v kapitole 4.1.4.

5.2.1 Princip 1: Eliminovat plýtvání

Podle svých zkušeností v našem testovacím týmu představím dále v textu několik typických příkladů, jak je možné tento princip dále aplikovat. Věřím, že mnoho testovacích týmů se dokáže s následujícím textem ztotožnit.

Testovací tým by se měl vyhnout veškerým aktivitám, které zákazníkovi nepřinášejí hodnotu. První z nich je vytváření zbytečné dokumentace. Každý zákazníkův požadavek nemusí být zdokumentován do testovacího scénáře. Někdy bývají tyto požadavky triviální a jak programátor, tak tester je schopný změnu aplikace bez přípravy pochopit a otestovat. Testovací scénáře slouží spíše pro přípravu testování komplexních funkcí s mnoha různými cestami, kdy při mnoha různých vstupech může nastat příliš různých výstupů – a tím nemám na mysli matematické výpočty. Pokud nejsou testovací scénáře vyžadovány projektovým manažerem nebo jiným subjektem, nejsou nutné a jedná se o plýtvání. Aktuálně se našemu týmu podařilo prosadit tento názor a eliminovali jsme časové plýtvání.

Dalším příkladem plýtvání je vytváření automatizovaných testů tam, kde jsou zcela nepotřebné. Dobrý tester disponuje jistou intuicí a dokáže posoudit testovanou funkci tak, aby věděl, jestli se jedná o rizikovou funkci aplikace, která by měla být testována pravidelně při každé změně nebo nemusela. Automatizované testy jsou většinou časově náročné na vytvoření, a pokud by byly programovány na každou triviální změnu, docházelo by k příliš velkému plýtvání časem. Několikrát se našemu týmu přihodila situace, kdy během vytváření automatizovaných testů byla funkcionalita aplikace dvakrát změněna, komunikace s vývojáři toto plýtvání eliminovala. Následně se testovací tým dohodl, že automatizované testy budou programovány na základě vyjádření všech testerů.

Pokud bych se zaměřil na samotnou organizaci týmu, tento princip také týmům radí, aby jejich členové měli k sobě navzájem okamžitý přístup, nejlépe seděli v jedné místnosti. Preferuje se osobní komunikace a možnost okamžitého vyřešení problému. Každý zbytečně odeslaný email kolegovi, který pracuje příliš daleko od jiného, se bere jako plýtvání, a to kvůli časové prodlevě než přijde odpověď. Stejně tak jako mezi testery by komunikace měla probíhat nejlépe na osobní bázi v rámci celého vývojového týmu. Bohužel je náš tým rozmístěn mezi několik států, proto si celý tým nainstaloval instantní komunikátor. Další druh plýtvání byl úspěšně eliminován.

Každý tester by se měl soustředit na zadaný požadavek zákazníka. Znám situace, kdy i já bych jisté funkce systému udělal jinak, podle mého názoru lépe, ale jestliže jsou jasně dané požadavky, není třeba vykonávat extra činnosti, tedy testy nad rámec požadavků, které se podle LeanSD principu berou také jako plýtvání.

Vyhledávání jak dokumentů, tak všech ostatních druhů informací je jedním z častých problémů a počítá se mezi další druh plýtvání, který musí testovací tým eliminovat. Testovací tým by se měl dohodnout, který DMS bude při své práci používat, pokud počítáme s tím, že DMS v týmu existuje. Náš testovací tým zamezil duplicitním a nesprávným verzím dokumentů pořízením sotwarového nástroje sloužící k tomuto účelu.

Duplicita a přidávání stejných dat do různých, avšak účelově stejných systémů je plýtvání také, protože se jedná o zbytečnou činnost jakéhokoli člena týmu. V kapitole 4.1.4. jsem zmínil přítomnost dvou systémů, kde do prvního mohl zasahovat zákazník a poskytovat své dokumenty a do druhého měl přístup pouze vývojový tým. Sloučení těchto dvou systému a evidence duplicit se nám také podařilo eliminovat pomocí zrušení jednoho z nich.

Následující text poskytne příklady nástrojů, které nám v našem týmu pomohly eliminovat plýtvání podle principu 1 konceptu LeanSD.

Použité nástroje

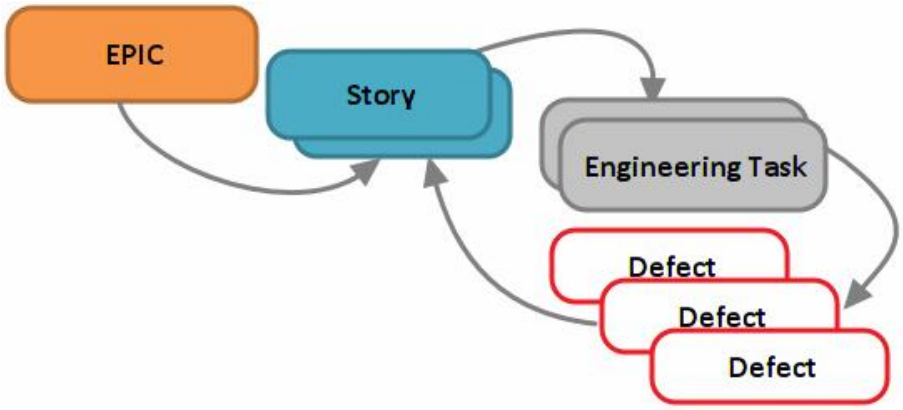
JIRA

Celý vývojový tým se dohodl na používání jednoho softwarového nástroje pro sledování postupu vývoje projektu. Byl zvolen nástoj JIRA, do kterého byly přidávány úkoly vývojářům a defekty. Přístup do tohoto systému byl povolen i zákazníkovi, který mohl sledovat práci na úkolech a případně přidat dokumenty s požadavky. Druhý systém pro sledování práce na projektu TFS byl následně zrušen.

Výhodou systému JIRA je jednoduché vytváření reportů projektovému vedení, možnost jednoduše vkládat úkoly, vytvářet podúkoly a přidělovat je konkrétním lidem a tak efektivně rozdělit práci. Navíc celý systém disponuje intuitivním a přehledným grafickým uživatelským rozhraním.

Zrušením systému TFS jsme dosáhli eliminaci plýtvání ve vytváření duplicit, provádění zbytečných činností a doby vyhledání odpovědí. TFS byla v tomto případě zbytečná funkcionalita, a tak byla zrušena.

Tým se dohodl na jednom používaném workflow, který JIRA systém bez problému umožňoval. Standardní rozdělení pracovních položek v tomto systému vypadalo následovně.



16 Obrázek - Rozdělení pracovních položek v JIRA systému (autor)

Základním kamenem při rozdělování úkolů v systému JIRA je pracovní položka, které se říká EPIC, což je v našem týmu používáno pouze pro definování, která část produktu bude změněna nebo nově vytvořena. Příkladem EPIC záznamu může být například Produkt. Každá taková EPIC položka slouží spíše jen pro přehled a informaci, která část systému bude ovlivněna.

EPIC položka se dělí na mnoho Story položek. Jedna Story položka je prakticky jeden zákazníkův požadavek na změnu, resp. jedna nová požadovaná funkcionalita. Zákazník nebo člen týmů vkládá dokumenty se změnovými požadavky do každé Story položky. Problém s takovým požadavkem může být příliš velká složitost pro jednoho vývojáře, a proto bývá většina Story položek dále rozdělena na jednodušší úkoly – Engineering Tasky, které jsou rozděleny konkrétním vývojářům. Každý samostatný Engineering Task je prakticky část funkcionality, která se většinou může samostatně otestovat. Pokud testovací tým najde chyby, jsou přidány do Story položek jako Defecty. Ve Story položkách si lze poté zobrazit seznam Engineering Tasků a Defectů přidělených k dané funkcionalitě. Každá nová

funkcionalita (Story položka) je uzavřena a považována za kompletně vytvořenou a otestovanou, pokud jsou všechny Engineering Tasky uzavřeny a všechny nalezené Defecty opraveny.

Sharepoint

Celý vývojový tým, včetně testovacího týmu, eliminoval výskyt duplicitních dokumentů, obrázků, návodů, dokumentace a jiných typů dat na centrálním úložišti se všemi standardními funkcemi, které používá každý dobrý systém pro řízení dokumentů. Byl vybrán nástroj Microsoft Sharepoint. Testovací tým do tohoto systému ukládá testovací scénáře, testovací plány, harmonogramy, rozdělení úkolů, ale také zdokumentovanou komunikaci mezi klienty nebo informace od kohokoli dalšího pro definování bližších informací týkající se projektu. Sharepoint eliminuje plýtvání ve zbytečných činnostech a době potřebné k vyhledání dokumentů.

Skype

Celý tým hledal cestu k nejrychlejší a efektivní komunikaci napříč celým týmem. Bylo nalezeno několik možností instalace firemních komunikátorů, avšak těchto programů byla nedosažitelnost člena týmu, pokud byl zrovna odpojen od firemní sítě. Potřebovali jsme také možnost konferenčních hovorů, resp. přenos hlasové komunikace mezi několika členy týmu, kteří se navíc nacházejí v různých státech. Nejlepším řešením se stala aplikace Skype, kterou nyní má celý tým nainstalován.

5.2.2 Princip 2: Neustále přidávání kvality

Předchozí kapitola ukázala, jak by se dala zefektivnit práce pomocí aplikace principu eliminace plýtvání podle LeanSD. Ovšem samotná eliminace míst, kde dochází k plýtvání, sama o sobě nevylepší proces testování, resp. nezajistí vyšší kvalitu dodávaného produktu.

Testovací tým se musí neustále zlepšovat ve svých činnostech, stále hledat možná vylepšení a být schopný přijmout nové nástroje a techniky, které proces testování zefektivní.

V kapitole 4.1.4 jsem zmínil nedostatečnou podporu testovacími nástroji, což je klíčová komponenta efektivního testovacího týmu. Bez pořádného softwarového nástroje ani sebelepší tester nedokáže zajistit co nejvyšší kvalitu produktu. Náš testovací tým po komunikaci s projektovým manažerem dostal pozitivní odpověď a byly zakoupeny licence programu HP Quick Test Professional (QTP), HP Service Test (ST) a loadUI.

Celý tým také přijal některé klíčové faktory LeanSD a i vývoj produktu je nyní řízen agilním přístupem. Agilní přístup znamená, že programování i testování je řízeno v iterativním vývojovém cyklu, o kterém jsem psal v kapitole 3.2.4. Toto rozhodnutí znamená nezbytnost pravidelných schůzek, během kterých se účastní každý člen a kde jsou rozděleny úkoly a předávají se zde informace ohledně postupu vývoje, či testování.

Při těchto schůzkách probíhá rozdělování úkolů a každý programátor ví, na které části systému bude pracovat, a také zhruba ví, jak moc času bude potřebovat. Kromě toho, že testovací tým se vyjadřuje k časové náročnosti testování, bude také vědět, které části produktu, resp. funkce budou v nejbližší době vytvořeny a připravuje se na jejich testování podle zákazníkových požadavků. Během tohoto rozdělování úkolů probíhá diskuze, do které se zapojuje i testovací tým. Celý tým, nejlépe včetně přítomností zákazníka, rozděluje priority k úkolům. Úkoly s nejvyšší prioritou mají přednost a i testovací tým bude vědět, na které funkce produktu se bude soustředit jako první a začne se již připravovat a vytvářet testovací scénáře. Samotné rozdělování úkolů v rámcí celého ale i jen testovacího týmu probíhá za pomocí techniky Kanban.

Použité nástroje

HP Quick Test Professional, HP Service Test a loadUI

Quick Test Professional (QTP) je testovací nástroj pro automatizaci testů, který je určen pro funkcionální testování jak webových, tak klientských aplikací. Pomocí QTP je možno v rámci jednoho testu testovat kroky pro různé uživatelé, různá vstupní data a taky různé webové prohlížeče, je-li to potřeba. Programovacím jazykem těchto automatizovaných testů v QTP je Visual Basic Script, který sice omezeně, ale podporuje objektově orientované programování. (61)

Tento nástroj našemu testovacímu týmu mnohem ulehčil práci. Vytváření automatizovaných testů se stalo jednodušší a rychlejší a samotná exekuce testů je natolik jednoduchá, že spouštět tyto testy dokáže každý člen týmu. Díky tomuto nástroji jsme nyní schopní pokrýt aplikaci s mnohem vyšším počtem automatizovaných funkcionálních testů.

Druhým pořízeným testovacím nástrojem, který byl zakoupen je Service Test (ST). Tento nástroj umožňuje jednoduché vytváření integračních testů, resp. testů webových služeb komunikujících jak mezi sebou tak s naší aplikací. Tento nástroj disponuje grafickým prostředím pro intuitivní programování testů webových služeb.

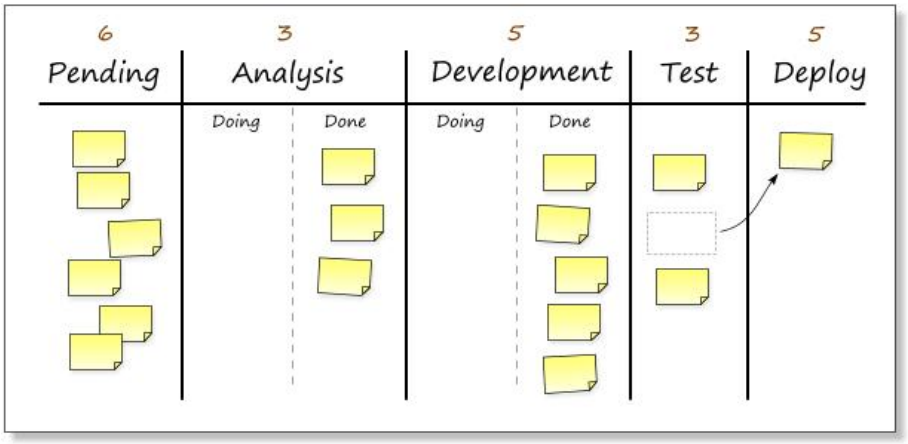
Nevýhodou obou těchto nástrojů je jejich vysoká pořizovací cena.

Náš testovací tým je nyní připraven take vytvářet výkonnostní testy, protože byl pořízen nástroj loadUI, který toto testování umožňuje.

Kanban

Kanban je podle (62) technika řízení softwarového procesu vysoce efektivním způsobem. Základním nástroj této techniky je kanbanová tabule, na které se umísťují papírky s úkolů tak, jak probíhá práce na jejich dokončení. LeanSD radí, aby si tým vytvořil kanbanovou tabuli, kde pravidelně bude aktualizovat stav práce na konkrétních funkcích produktu. Tabule byla zakoupena a je pravidelně aktualizována během denních schůzek s celým týmem. Díky tomu se nám zlepšil celkový pohled na vývoj, máme přehled o práci ostatních členů týmu a také v rámci testovacího týmu.

Pro představu, jak se dá technika Kanban využít, asi nejlépe znázorní následující obrázek Kanbanové tabule.

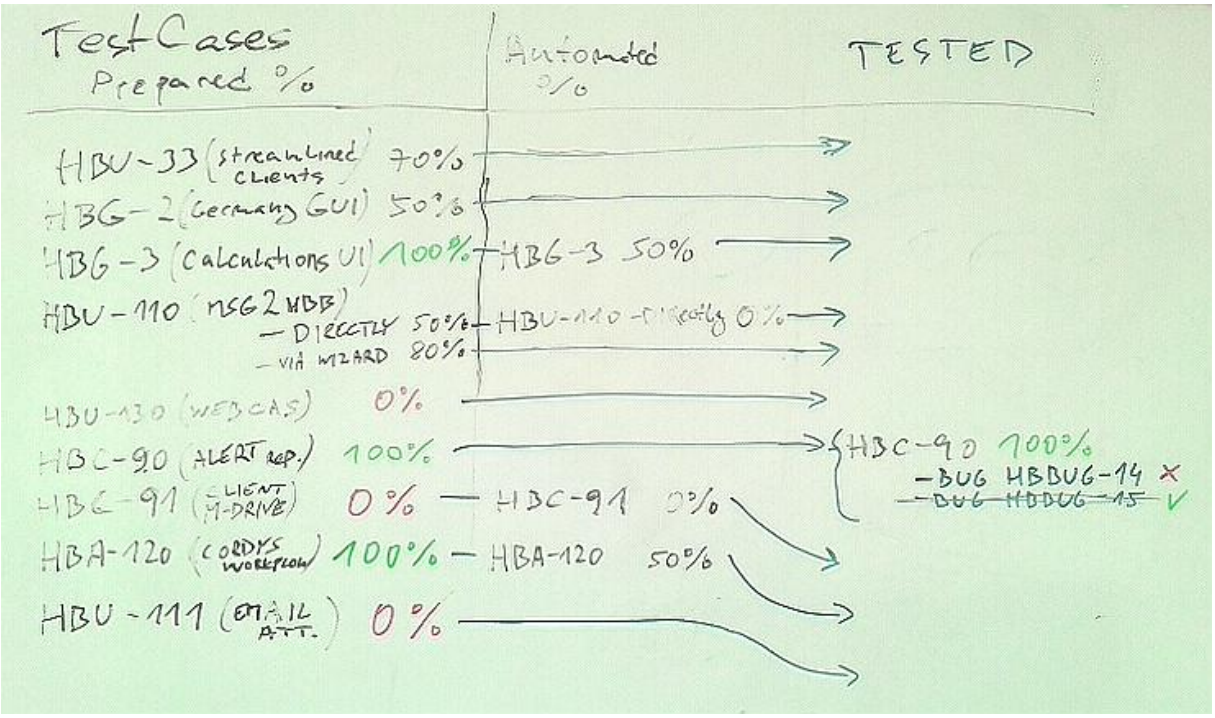


17 Obrázek - Kanbanová tabule pro celý vývojový tým (62)

V našem týmu se jedná o nástěnnou tabuli, kde každý žlutý papírek obsahuje jeden úkol, který musí být splněn v určitém čase. Obrázek jako tento může znázorňovat aktuální sprint, resp. iteraci, během které je nutné každý takový úkol postupně přesunout z prvního sloupce s čekajícími úkoly na provedení až do posledního sloupce, kdy je nová funkce předána, například do produkčního prostředí zákazníkovi.

Podobnou kanbanovou tabuli si vytvořil i náš testovací tým a podobně jako celý tým má přehled o stavu všech úkolů, tak i náš tým má přehled o stavu vytváření našich testů. Testovací kanbanová tabule vypadá stejně jako následující obrázek. Priorita úkolů

je například naznačena jinou barvou papírku, v tomto obrázku červenou, a z logiky věci vyplývá, že tyto úkoly musí testovací tým otestovat co nejdříve, pokud je daná funkce již naprogramována.



18 Obrázek - Kanbanová tabule pro testovací tým (autor, podle (62))

Na obrázku je vidět seznam úloh, které je potřeba otestovat v rámci jedné iterace. V našem případě je důležité rozhodnout, které funkce produktu musí obsahovat testy manuální a které funkce se dají zautomatizovat. Pro toto rozhodnutí vedeme denně osobní komunikaci a porovnáváme, zda k vytvoření automatizovaného testu je nutné větší usíli, než provedení testu manuálně, avšak opakovaně.

Nemusí se vždy jedna o nástěnnou tabuli, kde jsou tyto úkoly nebo testy manuálně vyvěšeny, ale na trhu již existuje několik softwarových nástrojů, a tak se rozdělení práce může vykonávat přehledněji v elektronické podobě. Většina těchto nástrojů disponuje mobilní aplikací, která je automaticky synchronizována s vybraným nástrojem. Jedním z oblíbených nástrojů je například LeanKit Kanban (http://leankitkanban.com/).

5.2.3 Princip 3: Odložení rozhodnutí na později

Princip odkládání rozhodutí na pozdější dobu je aplikovatelné spíše pro vývojový tým, který není během úvodní fáze vývoje pevně rozhodnut například s návrhem některé části systému a brzké rozhodnutí by mohlo v pozdějším stádiu narušit. Avšak testovací tým může tento princip přijmout podobně, a to tak, že nebude nepromyšleně vytvářet automatizované

testy nebo jiné testy bez jasně předložených požadavků. Pokud neexistuje požadavek, který nutí kohokoli z týmu rozhodnout o daném problému okamžitě, měl by rozhodnutí odložit. Rozhodnutí jsou vykonány „právě včas“.

Tento princip jsme také aplikovali v našem testovacím týmu a s pomocí kanbanové tabule a prioritizace úkolů byla vedlejší rozhodnutí odložena a tým se tak mohl soustředit na aktuální testování a vytváření prioritnějších testovacích scénářů nebo samotných testů.

Použité nástroje

Just In Time (JIT)

Jedná se o firemní strategii k zajištění vyšší efektivity a zamezení plýtvání zdrojů. Do češtiny přeloženo jako „právě včas“. Podstata této strategie by se dala vysvětlit pomocí následujícího příkladu – továrna pro výrobu aut je závislá na materiálech mnoha dodavatelů, ovšem není potřeba tento materiál skladovat, jestliže se ví, že konkrétní materiál ve velmi blízké budoucnosti potřeba, tzn. takový materiál není objednán, sníží se tak náklady na skladování, nejlépe však vyruší úplně a místo toho, je materiál objednán „právě včas“. Toto je základ strategie JIT, která v tomto příkladě znamená výrobu s velmi nízkými zásobami a rozhodnutí, kdy nakoupit nový materiál, je provedeno právě včas.

Stejně tak testovací tým provádí rozhodnutí o následující práci „právě včas“: eliminujeme plýtvání tak, že zbytečně nestřídáme zaměření myšlenek na více úkolů najednou, a také snižujeme výskyt nedokončené práce.

5.2.4 Princip 4: Dodávat co nejrychleji

LeanSD klade důraz na dodávku produktu co nejrychleji, proto se náš celý tým rozhodl vývoj rozdělit na iterace (cykly). Pro testovací tým to znamená, že musí být schopný otestovat danou funkci systému včas v rámci jedné iterace, po které je produkt představen zákazníkovi. Odpovědnost, že test proběhne včas a testovací tým je tak schopen zajistit kvalitu dodaného produktu, leží na zkušenosti testerů a jejich schopnosti odhadnout náročnost testování vybrané části systému, které jsou naplánovány pro aktuální iteraci. Tato komunikace a upřesnění úkolů a jejich rozdělení probíhá na schůzke začátkem každé iteraci a každý člen v našem týmu je povinen se této schůzky účastnit, proto jsme nyní po aplikace tohoto principu zajistit, že produkt bude otestován a dodán na konci každé iterace.

Použité nástroje

Iterativní vývoj (testování)

Vysvětlení iterativního vývojového cyklu a pozici testování jsem již vysvětlil v kapitole 3.2.4.

5.2.5 Princip 5: Posílit postavení členů týmu

LeanSD jako přístup vývoje software pro menší týmy také počítá s každým členem jako specialistou na svou část práce s dostatečnými zkušenostmi. Tyto zkušenosti se jistě hodí celému testovacímu týmu, nejen testerům, ale názor každého člena se bere v potaz na pravidelných schůzkách týmu.

Myslím, že všichni členové našeho testovacího týmu jsou dostatečně zkušení, abychom se byli v případě potřeby schopni zastoupit a rychle a dobře vyřešit jakýkoli problém. Všichni členové našeho testovacího týmu se podílejí pravidelných schůzek a náš názor je na těchto schůzkách zohledněn.

5.2.6 Princip 6: Integrace systému

Každá dnešní aplikace obsahuje mnoho a mnoho různých obrazovek, na které se uživatelé dívají a se kterými pracují a vykonávají svou pracovní činnost. Každá taková obrazovka obsahuje různá pole, popisky, tlačítka a jiné ovládací panely, pomocí kterých s aplikaci komunikují.

Aplikace tohoto principu pro celý vývojový tým znamená, že dvě textová pole, která mají za úkol zobrazit na dvou různých obrazovkách tu stejnou hodnotu, mají stejný význam a vlastně se jedná o ta dvě stejná pole, na obou obrazovkách budou samozřejmě také se stejným popiskem, aby uživatel věděl, k čemu zobrazena hodnota pole patří. K tomuto účelu se nejlépe hodí testy grafického uživatelského rozhraní (GUI), který náš testovací tým také vytváří. Ovšem jedná se většinou o testy manuální a chyby, které se v průběhu testu objeví, jsou občas nalezeny až klientem, který nejlépe ví, co a kde chtěl v aplikaci zobrazit, a třeba jen náhodou nesprávně vyplnil text do dokumentu s požadavkem na změnu.

Použité nástroje

O používaných nástrojích k účelům testování GUI našeho produktu jsem psal v kapitole 4.2.2.

5.2.7 Princip 7: Optimalizace celku

Aplikace LeanSD principu pro testovací tým znamená, že nesmíme zapomenout nahlížet na produkt, ale jako na systém, který je běžně v dnešní době vytvořen z několika oddělených částí, které lze buď otestovat samostatně, nebo v kontextu „hlavní“ aplikace. Pro tento účel existují integrační testy nebo tzv. komponentové testy a v našem testovacím týmu se s nimi počítá (jsou zařazeny na pozdější fáze vývoje, kdy bude již funkcionalita produktu v pokročilé fázi a bude vůbec tyto testy možné vytvořit).

Použité nástroje

O používaných nástrojích k účelům testování komponent našeho produktu jsem psal v kapitole 4.2.2.

5.2.8 Shrnutí

V této kapitole jsem ukázal, jak vypadá aktuální situace našeho testovacího týmu, který přijal koncept LeanSD a aplikoval jeho principy v procesu testování. Uvedl jsem pozitivní vlastnosti tohoto přijetí a zdůraznil výhody, které LeanSD našemu týmu přinesl.

V kapitole 4.1.4 jsem popsal nevýhody situace v našem testovacím týmu a problémy, se kterými jsme se denně potkávali ještě před přijetím konceptu LeanSD. Myslím si, že všechny tyto problémy jsou vyřešeny, přestože jedním z problému byla nedostatečná kapacita testovacího týmu, která se díky změně průběhu testování, ale i vývoje nezdá v tuto chvíli jako aktuální problém.

Následující tabulka zobrazuje aktuální přehled používaných softwarových nástrojů v porovnání s předchozí situací:

Typ testu	Nástroj	Rychlost psaní testů	Jednoduchost používání	Vhodnost použití	Celková spokojenost
Integrační testy	Nunit	100%	80%	90%	90%
Testy webových služeb	Service Test	100%	90%	100%	97%
Funkcionální testy	QTP	90%	95%	75%	87%
GUI testy	QTP	90%	95%	90%	92%
Výkonnostní testy	loadUI	100%	90%	100%	97%

3 Tabulka – Nástroje používající se po přijetí LeanSD (autor)

Pokud bych srovnal tuto tabulku s taboulku v kapitole 4.1.4, která zobrazovala situaci před přijetím konceptu LeanSD, tak se značně zvýšila efektivita naší práce, protože našemu týmu byly zakoupeny license nástrojům, které jsme nezbytně potřebovali pro vytváření našich automatizovaných testů. Díky tomu se samozřejmě zvýšila i celková spokojenost a jsme nyní

také schopni vytvářet nové druhy testů, které jsme efektivně vytvářet nedokázali. LeanSD samozřejmě nestojí za dodání specializovaného testovacího software, avšak přijetí jeho principů a díky následným změnám v procesu vývoje náš tým potřeboval dostatečnou podporu pro maximální zajištění kvality.

5.3 Hodnocení Lean software development v testovacím týmu

Aktuální situaci vývojového týmu a hodnocení pozice testování jsem zhodnotil pomocí dotazníkového šetření. Otázky byly rozeslány 20 pracovníkům vývojového týmu včetně 6 členů týmu testerů. Vytvořil jsem jednoduchý otazník a zkoumal, jak je vnímán proces testování, ale i vývoj produktu celkově. Kompletní dotazník lze najít v příloze 1.

Do šetření se zapojilo 10 respondentů (6 testerů a 4 vývojáři), kteří tak tvořili základ pro ověření mé hypotézy.

5.3.1 Dotazník

Otázka 1: Je podle vás testovací proces rychlejší díky LeanSD?

Ano	20%
Ano, trochu	60%
Ne, moc ne	20%
Vůbec ne	0%

4 Tabulka - Otázka 1 (autor)

Spokojenost s novým přístupem LeanSD v procesu testování převládá. Aplikací principů eliminace plýtvání a dalších jsme zredukovali nadbytečné oblasti plýtvání a zrušili jsme některé nástroje pro řízení testovacího procesu. Testovací proces a řízení defektů se tak stalo přehlednějším a zároveň rychlejším.

Otázka 2: Myslíš, že pravidelné schůzky mají smysl?

Ano	60%
Ne	40%

5 Tabulka - Otázka 2 (autor)

Odpovědi na tuto otázku jsme rozdělili na dva podobně velké tábory. Osobně jsem se ptal členů týmu a zjišťoval, proč si myslí, že schůzky nemají smysl. Pochopil jsem, že důvodem je zbytečná prodleva, která by nastala, kdyby člen týmu čekal se svými otázkami, než se bude konat pravidelná schůzka. Jako argument také připojili okamžitě odeslaný email,

který všechno vyřeší, případně telefonní konference, je-li nutná přítomnost více lidí. Díky tomuto přístupu, jsou pak schůzky podle nich zbytečné.

Ovšem větší polovina však také uvádí, že schůzky smysl mají a to protože ostatním členům týmu dávají přehled o celkovém stavu produktu. Jak jsem již zmínil v předchozích kapitolách, výhodou těchto schůzek z manažerského pohledu je sledování práce na konkrétních úkolech, které jsou na schůzkách rozdělovány a lze sledovat práci konkrétního člena na zadaném úkolů a poté vyhodnotit jeho práci, což samozřejmě platí i pro testovací tým.

Otázka 3: Myslíš, že je díky LeanSD testování aplikace rychlejší?

Ano	20%
Ano, trochu	60%
Ne, moc ne	20%
Vůbec ne	0%

6 Tabulka - Otázka 3 (autor)

Také i při této otázce se tým z 80% shodl, že samotné testování aplikace je rychlejší. Vývoj v iteracích umožnil nebo spíše způsobil potřebu zakoupení efektivních nástrojů, větší podporu mezi druhy testů, které je náš testovací tým schopný provádět a také se zvýšila rychlost vytváření těchto automatizovaných testů s kvalitními nástroji. Jsme nyní schopní vytvořit obsáhlejší regresní automatizované testy ve stejném čase.

Otázka 4: Myslíš, že díky LeanSD je aplikace kvalitněji (více testů) otestována?

Ano	60%
Ne	40%

7 Tabulka - Otázka 4 (autor)

Většina testerů si myslí, že aplikace je kvalitněji testována a dochází méně často k chybám. Je pravdou, že díky LeanSD máme větší přehled o tom, kdo na jakém úkolu pracuje a testovací tým se může připravit na konkrétní testy s přípravou testovacích scénářů a rozvrhnout si tak testovací kapacitu.

Argument členů, kteří odpověděli, že kvalitnější testy ani po přijetí LeanSD nejsou, je ten, že počet testerů určených k testování je pořád nedostatek a i sebelepší tester není schopný vytvořit tak kvalitní a obsáhlé testy v tak krátkém čase.

Otázka 5: Jsi spokojen s testovacím procesem podle LeanSD?

Ano	0%
Ano, trochu	80%
Ne, moc ne	20%
Vůbec ne	0%

8 Tabulka - Otázka 5 (autor)

Eliminace plýtvání, agilní vývoj, testování v iteracích, komunikace se zákazníkem, pravidelné schůzky a díky aplikací dalších principů LeanSD se přijetí tohoto konceptu stalo oblíbeným a spokojenost členů nejen testovacího týmu se zvýšila.

Rozvrhnutí času, přehled o práci ostatních členů a použití ostatních technik a nástrojů jako je například kanbanová tabule, to všechno pomohlo spokojenému přijetí LeanSD.

Otázka 6: Napiš, jaké výhody tobě LeanSD přinesl.

Kratší cykly při odevzdání produktu znamenají rychlejší zpětnou vazbu pro náš tým velmi důležitou.
Větší přehled. Víme kdy, co a jak má každý člen týmu za úkol.
Nestojí za řeč.
Nevím
Nejsem si jistý.

9 Tabulka - Otázka 6 (autor)

Při této vypisovací otázce se nálada přijetí LeanSD různí, avšak i z těchto odpovědí se dá usoudit, že přijetí LeanSD pomohlo v řízení testovacího procesu. Každý člen testovacího týmu ví, kdo a na čem pracuje, proto případné otázky lze směrovat na konkrétního člověka. Pravidelné schůzky našeho týmu umožňují diskuzi nad náročností a lze tak denně a efektivně nasměrovat testovací kapacitu na konkrétní úkol.

Mezi odpovědmi lze najít i některé nejasné, které spíše naznačují, že výhody nejsou zcela očividné, avšak tyto odpovědi přikladám na vinu toho, že LeanSD je v naší společnosti aplikován pouze po dobu necelých dvou měsíců.

Otázka 7: Napiš, jaké nevýhody tobě LeanSD přinesl.

Bez konkrétní funkcionální specifikace vznikají nejednoznačnosti o tom, co a jak by mělo být vytvořeno.
Nevím.
Zatím žádné.
Nevím.

10 Tabulka - Otázka 7 (autor)

Nevýhody se daly také očekávat a zeptal jsem se členů testovacího týmu, jaké nevýhody jim přinesli. Většina týmu si nebyla jista, avšak jeden názor se objevil. Nevýhoda LeanSD se zdá být v agilním přístupu a aplikací principu odložení rozhodnutí na co možná nejpozději.

Zákazník není často pevně rozhodnut s požadovanou funkcionalitou a až v průběhu vývoje vznikají problémy související s aplikací nové funkcionality, tudíž nutnost provést změnu v mnoha regresních automatizovaných testů. Dále dokumentace je po přijetí LeanSD slabší a nedostatek detailních dokumentů s požadavky neumožňují správně pochopit a otestovat novou nebo upravenou funkcionalitu aplikace.

Otázka 8: Bylo podle tebe snadné v testovací týmu přijmout principy LeanSD?

Ano	20%
Ano, trochu	20%
Ne, moc ne	40%
Vůbec ne	20%

11 Tabulka - Otázka 8 (autor)

Při této otázce se názory velmi různí. Rozmanitost odpovědí přikládám ke zkušenostem každého člena týmu, který již tento agilní přístup již zažil v minulosti nebo dostatečně adaptibilní pro přijetí změny. Ne každý je rád nebo schopný okamžitě přijmout změnu,

proto vznikají negativní ohlasy, ale věřím, že po několika měsících již budou všichni členové týmu zvyklí na tento přístup a budou spokojeni.

Jak jsem již zmínil, tento přístup je v naší společnosti aplikován pouze v době necelých dvou měsíců, což může znamenat to, že 60% všech dotázaných odpovědělo, že přijetí LeanSD principů pro tyto členy snadné nebylo.

Otázka 9: Myslíš, že se díky LeanSD celkově zrychlil vývoj aplikace?

Ano	40%
Ne	60%

12 Tabulka - Otázka 9 (autor)

Na tuto otázku bylo těžké odpovědět, jelikož LeanSD je teprve nedávno v naší společnosti zaveden a vývojové ani testerské kapacity nebyly navýšeny. Tudíž jsou ohlasy spíše negativní, jelikož práce vývojového i testovacího týmu se nedá zrychlit během týdne nebo měsíce, než budou principy LeanSD přijaty kompletně a navíc je poměrně těžké ještě při své práci hledat místa plýtvání a aplikovat ostatní principy. Bude to jistě stát ještě nějaký čas, než bude většina členů spokojena s tímto přístupem a práce bude maximálně efektivní.

Otázka 10: Napiš, co bys změnil v testovacím týmu po přijetí LeanSD.

Všechny funkce musí pracovat v užším okolí. Tým vývojářů nemůže být geograficky vzdálen od testovacího. Rychlejší vývoj a brzké odevzdání aplikace nezaručí vyšší kvalitu.
Přijal bych další 2 testery.
Netuším.
Nevím.

13 Tabulka - Otázka 10 (autor)

Poslední otázka se týkala hlavně osobního názorů testerů, co by se mělo po přijetí nebo v rámci LeanSD ještě změnit v testovacím týmu. Kromě navýšení testerských kapacit se objevila další myšlenka. Jelikož je naše společnost roztroušena po celém světě tak i několik vývojářů a testerů pracuje geograficky na jiné lokaci. Toto je podle některého z členů týmu myšlenka, která negativně ovlivňuje LeanSD v našem týmu. Agilní přístup LeanSD je určen

pro menší týmy, ale doporučuje se, aby týmy byly v jedné místnosti, kanceláři a podobně. Přes všechny dnešní vymoženosti ve vzdálené komunikaci, tak ta osobní je stále nejlepší.

5.4 Shrnutí

Vytvořil jsem dotazník, který definoval základní otázky na proces testování z pohledu zbytku týmu po přijetí LeanSD, tak i na samotný pohled testování. Některé výsledky mi byly také podrobněji vysvětleny a tak jsem ke každé otázce přidal upřesňující odpověď, aby nedošlo k nějakému omylu. Co se týká samotných výsledků, můžeme říci, že většina členů testovacího týmu je spokojena s konceptem LeanSD, s vylepšeným procesem testování i s testováním samotným, ale přestože jsou principy LeanSD jasně definovány a lze je jednoduše pochopit, tak aplikovat je pro vlastní testovací tým tak lehké není a přijetí LeanSD v testovacím týmu přinese výhody až v delším časovém období.

6 Závěr

Ve své diplomové práci jsem seznámil čtenáře s konceptem Lean myšlení a jeho modifikací na poli výpočetních technologií pod pojmem Lean software development. Uvedl jsem základní principy a aplikoval tento přístup na testovací tým, ve kterém pracuji.

V první kapitole jsem představil pozici agilního testovacího týmu a jak vypadá průběh testování, dále jsem popsal testovací kvadrant, který se dává za příklad testovacím týmům. Každý kvadrant jsem popsal a seznámil testovací tým s druhy testů, technikami a potřebnými nástroji.

Kapitola 2 navazuje a popisuje Lean myšlení obecně. Čtenáře jsem seznámil s historií konceptu Lean software development. Tento koncept definuje principy, které jsem rozebral a vytvořil obecný návod, jak by tyto principy mohly být aplikovány testovacími týmy.

Dále jsem zhodnotil výhody a nevýhody, které v Lean software development vidím z pozice testera, a přešel k další kapitole, která popisuje aplikaci tohoto konceptu v testovacím týmu, kterého jsem součástí.

Následující kapitola nejdříve popisuje náš výchozí stav testovacího týmu ve firmě. Popisuji použité technologie, nástroje, techniky, dostupné kapacity, geografické rozložení týmu a různé podstatné informace s důrazem na průběh testování.

Od popisu výchozí situace jsem se dostal na samotné zavedení principů Lean software development a konkrétně jsem každý z nich aplikoval a popsal, jak byl přijat naším testovacím týmem.

Pro hodnocení přijetí Lean software development jsem vytvořil dotazník a myslím, že tento koncept u nás v testovacím týmu obstál a pomohl nám s průběhem testování. I když na konkrétní výsledky, zda přijetí konceptu přineslo očekávané hodnoty, musíme ještě nějakou dobu počkat.

Myslím si, že cíle definované pro tuto diplomovou práci, které jsem uvedl v úvodu, jsem splnil. Představil jsem Lean software development a provedl čtenáře aplikací tohoto principu v našem testovacím týmu v reálné firmě.

Hypotézu, že testování podle LeanSD je efektivnější než stávající způsob práce testovacího týmu, jsem ověřil dotazníkovým šetřením, že kterého vyplývá spokojenost 80 % pracovníků

s tímto konceptem a jejich subjektivní pocit celkového zlepšení a zkvalitnění testovacího procesu. Doufám, že tato práce bude dalším zdrojem inspirace pro přijetí konceptu Lean software development v dalších testovacích týmech.

7 Seznam literatury

(1) Poppendieck, Tom & Mary. *Lean Software Development: An Agile Toolkit.* Addison-Wesley Professional, 2003. ISBN: 978-0321150783

(2) Alan Page, Bj Rollison, Ken Johnston. *Jak testuje software Microsoft*. ČR: COMPUTER PRESS, 2009. ISBN: 978-80-251-2869-5

(3) Mountain Goat Software. *What are the main Scrum Activities?* Mountain Goat Software, 2012. Cit. 24.10.2012. Dostupné z www: http://www.mountaingoatsoftware.com/topics/scrum

(4) Poppendieck, Tom & Mary. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional, 2006. 1. edice. ISBN: 978-0321437389

(5) Hefnerová, Lucie. *Lean software development*. Praha, 2012. Bakalářská práce. Vysoká škola ekonomická.

(6) Norrmalm, Thomas. *Achieving Lean Software Development.* Uppsala, 2011. Semestrální práce. Uppsala Universitet. ISSN: 1650-8319. Dostupné z www: http://uu.diva-portal.org/smash/get/diva2:400627/FULLTEXT01

(7) Poppendieck, Mary. *Principles of Lean Thinking.Minnesota*, Poppendieck.LLC, 2002. Dostupné online: http://meidling.jvpwien.at/uploads/media/LeanThinking.pdf

(8) Kirtesh Jailia, Sujata, Manisha Jailia, Manisha Agarwal. *LEAN SOFTWARE DEVELOPMENT ("As a Survival Tool in Recession").* International Journal of Software Engineering and Its Applications Vol. 5 No. 3, July, 2011. Dostupné online: http://www.sersc.org/journals/IJSEIA/vol5_no3_2011/6.pdf

(9) Tom Coplien, Gertrud Bjørnvig. *Lean Architecture: for Agile Software Development*. Wiley; 1 edition, 2010. ISBN: 978-0470684207

(10) Chodura, Ondřej. *Testování softwaru v metodice OpenUP.* Praha, 2010. Bakalářská práce. Vysoká škola ekonomická.

(11) Patton, Ron. *Testování softwaru.* 1. vyd. Praha, Česká Republika: Computer Press 2002. ISBN 80-7226-636-5.

(12) Profinit. *Testování softwaru* [online]. 2009, cit. 30.11.2009. Dostupné z WWW: http://www.profinit.eu/uploads/ souvisejici-obsah/dokumenty/cz/Profinit_Testovani%20software.pdf

(13) Kučera, Jan. *Hodnocení metodik vývoje informačních systémů z pohledu testování*. Bakalářská práce, VŠE, Praha, 2008.

(14) *Manifesto for Agile Software Development* [online]. 2001, cit. 18.11.2009. Dostupné z WWW: http://www.agilemanifesto.org/
(15) *Vodopádový model.* Dostupný z www: http://upload.wikimedia.org/wikipedia/commons/2/24/Vodopadovy_model.png
(16) LBMS. *Agile* [online]. 2012, cit. 26.10.2012. Dostupné z www: http://www.lbms.cz/Tema/R%20Agile.htm
(17) Lean Company. *Historie* [online]. 2006, cit. 26.10.2012. Dostupné z www: http://www.leancompany.cz/historie.html
(18) ATTN Consulting. *Lean Management* [online]. 2009, cit. 26.10.2012. Dostupné z www: http://www.attn.cz
(19) Differ! Dělejte věci jinak… *Agile samotný nestačí, je čas pro Lean* [online]. 2010, cit. 26.10.2012. Dostupné z www: http://www.differ.cz/?p=189
(20) James P. Womack, Daniel T. Jones. *Lean Thinking: Banish Waste and Create Wealth In Your Corporation.* New York : Free Press, 2003. ISBN 0-7432-4927-5.
(21) Allan Kelly. *Agile Demystified (v.2). DevelopMentor* [Online]. 2009. Cit. 16.3.2012. Dostupné z: www.develop.com/agiledemystified
(22) Eliyahu M. Goldratt. *Critical Chain*. North River Press, 1997. ISBN: 978-0884271536
(23) Doc. Ing. Ludmila Mládková, Ph.D. *Dvě dimenze znalosti, explicitní a tacitní* [Online]. 2008. Cit. 27.10.2012. Dostupné z www: http://bpm-tema.blogspot.cz/2008/06/dve-dimenze-znalosti-explicitni-tacitni.html
(24) Luu Duong. *Applying the "80-20 Rule" with The Standish Group's Statistics on Software Usage* [online]. 2009. Standish Group Study. Luuduong.ca. Dostupné z www: http://luuduong.com/blog/archive/2009/03/04/applying-the-quot8020-rulequot-with-the-standish-groups-software-usage.aspx
(25) Poppendieck, Mary. *Lean Software Development. C++ Magazine: Methodology Issue* (Podzim 2003). Poppendieck.LLC, 2002.
(26) Mountain Goat Software. *The Daily Scrum Meeting.* Mountain Goat Software, 2012. Cit. 27.10.2012. Dostupné z www: http://www.mountaingoatsoftware.com/scrum/daily-scrum
(27) Manifest Agilního vývoje software. *Principy stojící za Agilním Manifestem.* 2001. Dostupné z www: http://agilemanifesto.org/iso/cs/principles.html
(28) Wikidot.com. *Životní cyklus IS* [online]. 2010. Dostupné z www: http://ari.wikidot.com/zivotni-cyklus-is
(29) OpenUP. *Introduction to OpenUP* [online]. 2012. Dostupné z www: http://epf.eclipse.org/wikis/openup/

(30) Michal Krčál. *Analýza nasazení podnikového portálu.* Brno, 2010. Diplomová práce. Masarykova univerzita.

(31) Lisa Crispin, Janet Gregory. *Agile testing : a practical guide for testers and agile teams.* Addison-Wesley, 2008. ISBN: 978-0-321-53446-0

(32) Lisa Crispin. *An Overview of Agile Testing. Lisa Crispin*, 2009. Prezentace. Dostupné z www: http://lisacrispin.com/downloads/AgileTestingOverview.pdf

(33) James Bach. *What is Exploratory Testing?* Satisfice Inc. 2002. Dostupné z www: http://www.satisfice.com/articles/what_is_et.shtml

(34) Samuel L. Baker. *Critical Path Method (CPM).* 2004. Dostupné z ww: http://hadm.sph.sc.edu/courses/J716/CPM/CPM.html

(35) Lisa Crispin. *Agile Test Planning with the Agile Testing Quadrants*. Lisa Crispin, 2009. Prezentace. Dostupné z www: http://lisacrispin.com/downloads/AdpTestPlanning.pdf

(36) rubytester. *Agile Testing Quadrants Explained*. Prezentace. 2012. Dostupné z www: http://www.slideshare.net/testrus/agile-testing-quadrants-explained

(37) Jeffrey Palermo. *Guidelines for Test-Driven Development*. Microsoft Corporation, 2006. Dostupné z www: http://msdn.microsoft.com/en-us/library/aa730844%28v=vs.80%29.aspx

(38) Vrstevnice. *Otázka 17 - Y36SI3*. Vrstevnice, 2012. Dostupné z www: http://www.vrstevnice.com/akce/grandaction/vskola/7statnice/otazky/otazkaSW17_36OMO.pdf

(39) UncleBob. *The Three Rules of TDD*. Blog, 2005. Dostupné z www: http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

(40) Gerard Meszaros. *Component.* xUnitPatterns.com, 2008. Dostupné z www: http://xunitpatterns.com/component.html

(41) Margaret Rouse. *Service oriented architecture (SOA).* TechTarget, 2008. Dostupné z www: http://searchsoa.techtarget.com/definition/service-oriented-architecture

(42) PIE Software Inc. *What is Debugging?* PIE Software, 2002. Dostupné z www: http://www.piesoftwareinc.co.uk/textonly/debugging.html

(43) *Original NUnit 2.2 GUI*. New, slicker GUI for NUnit, 2004. Dostupné z www: http://homepage.ntlworld.com/james.greenwood4/software/images/blog/nunit1.gif

(44) Webopedia. *Add-in.* Dostupné z www: http://www.webopedia.com/TERM/A/add_in.html

(45) Maxxess. *What is a Software Framework?* Maxxess, 2010. Dostupné z www: http://www.maxxess-systems.com/whitepapers/what_is_a_software_framework.pdf

(46) Otec Fura. *Mock testing – Potěmkinovy vesnice*. Myšlenky dne otce Fura, 2007. Dostupné z www: http://blog.novoj.net/2007/01/19/mock-testing-potemkinovy-vesnice/
(47) Dan North. *Behaviour-Driven Development.* BddWiki: BehaviourDrivenDevelopment, 2009. Dostupné z www: http://behaviour-driven.org/
(48) NBehave. *NBehave homepage*. CodePlex, 2012. Dostupné z www: http://nbehave.codeplex.com/
(49) Dmitri Nesteruk. *Behavior-Driven Development with NBehave.* CodeProject, 2009. Dostupné z www: http://www.codeproject.com/Articles/32512/Behavior-Driven-Development-with-NBehave
(50) *The Open Source Definition*. Open Source Initiative. Dostupné z www: http://opensource.org/osd
(51) *All About - User Acceptance Testing – UAT*. Guru99. Dostupné z www: http://www.guru99.com/user-acceptance-testing.html
(52) *Usability testing.* Usability.gov. Dostupné z www: http://www.usability.gov/methods/test_refine/learnusa/index.html
(53) *Visual Studio Load Testing Graphs.* VS2010 Load Testing for Distributed and Heterogeneous Applications powered by dynaTrace. About:Performance, 2010. Dostupné z www: http://blog.dynatrace.com/2010/03/10/vs2010-load-testing-for-distributed-and-heterogeneous-applications-powered-by-dynatrace/
(54) *Traditional Project.* LEAN SOFTWARE DEVELOPMENT ("As a Survival Tool in Recession"). International Journal of Software Engineering and Its Applications Vol. 5 No. 3, July, 2011. Dostupné online: http://www.sersc.org/journals/IJSEIA/vol5_no3_2011/6.pdf
(55) *Lean Approach.* LEAN SOFTWARE DEVELOPMENT ("As a Survival Tool in Recession"). International Journal of Software Engineering and Its Applications Vol. 5 No. 3, July, 2011. Dostupné online: http://www.sersc.org/journals/IJSEIA/vol5_no3_2011/6.pdf
(56) David Herbst. *Testování GUI - malé nakousnutí velkého tématu.* SW Testování, 2012. Dostupné z www: http://www.swtestovani.cz/index.php?option=com_content&view=article&id=68:testovani-gui-male-nakousnuti-velkeho-tematu&catid=3:zaklady&Itemid=11
(58) Susan de Sousa. *The Advantages and Disadvantages of Lean Software Development.* My-Project-Management-Expert.com, 2009. Dostupné z www: http://www.my-project-management-expert.com/the-advantages-and-disadvantages-of-lean-software-development.html

(58) *JIRA.* Atlassian, 2012. Dostupné z www:
http://www.atlassian.com/software/jira/overview
(59) *What is Document Management.* Enterprise Content Management, 2011. Dostupné z www: http://www.contentmanager.eu.com/dms.htm
(60) Margaret Rouse. *Workflow.* TechTarget, 2005.Dostupné z www:
http://searchcio.techtarget.com/definition/workflow
(61) Ankur Jain. *What is HP QuickTest Professional (QTP)?* LearnQTP, 2012. Dostupné z www: http://www.learnqtp.com/what-is-qtp/
(62) David Peterson. *What is Kanban?* Kanban blog, 2009. Dostupné z www:
http://www.kanbanblog.com/explained/index.html
(63) *Just In Time – JIT*. Investopedia US, 2012. Dostupné z www:
http://www.investopedia.com/terms/j/jit.asp#axzz2CUed25fF
(64) *Buzzword.* Merriam-Webster, Incorporated, 2012. Dostupné z www:
http://www.merriam-webster.com/dictionary/buzzword
(65) Václav Kadlec. *Agilní programování: Metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.
(66) Miroslav Čermák. *Testování SW*. Clever and Smart, 2012. Dorstupné z www:
http://www.cleverandsmart.cz/testovani-sw/
(67) *Software Testing As a Career - Complete Guide*. Guru99. Dostupné z www:
http://www.guru99.com/software-testing-career-complete-guide.html
(68) Usability.gov. *Use Cases.* U.S. Department of Health & Human Services. Dostupné z www:http://www.usability.gov/methods/usecases.html

8 Seznam obrázků

9 Seznam tabulek

10 Terminologický slovník

Termín	Zkratka	Význam
Add-In		Je softwarový program, který rozšiřuje schopnosti původního, většího programu. (44)
Agile Software Development	AgileSD	Vývoj softwaru, který je „Rychlý, hbitý, čilý, aktivní a svižný, který nepostává zdlouhavě u jednotlivých fází a jehož touhou je co nejrychleji postupovat k vytyčenému cíli – k dodání fungující aplikace.“ (65)
Backlog		Dynamický dokument obsahující seznam požadovaných vlastností, omezení, potřebných nástrojů. Obvykle je určen jeden z členů týmu, který se stará o aktualizaci dokumentu a upřednostňování práce na prioritních úkolech z backlogu podle zákazníkových požadavků (4)
Buzzword		Důležitě znějící obvykle technické slovo nebo fráze s nevelkým významem používající se hlavně k zapůsobení na laiky (64)
Debugging		Proces hledání a opravování chyb (defektů) v počítačovém programu nebo hardware (42)
Behaviour-Driven Development	BDD	Pomáhá soustředit vývoj k plnění a poskytnutí prověřitelných hodnot aplikace poskytnutím společného slovníku (47)
Iterace		Iterace je několika týdenní časový úsek, ve kterém je naprogramována a otestována část produktu. Tato část je na konci iterace nasazena a připravena ke kontrole zákazníkem. (autor)
Inkrement		Někdy označovány jako přírustky. Každý jeden inkrement představuje jednu samostatně realizovatelnou část produktu. (30)
Kritická cesta		Znamená vykonání základních a nezbytně fungujících kroků, které uživatel musí mít možnost provést, aby software na uživatelskou akci vrátil požadovanou reakci v rámci jedné funkcionality. Kriticka cesta je pak posloupnost těchto kroků. (autor,

		podle (34))
Lean Software Development	LeanSD	Koncept vývoje software (autor)
Metodika vývoje softwaru		,,Oblast popisující, jakým způsobem by měl probíhat vývoj softwaru s cílem získat kvalitní, dobře fungující a přitom rychle a pokud možno levně vytvořený programový produkt.“ (65)
Testovací scénář (test case)		Znamenají slovně zaznamenané kroky (akce), které musí tester vykonat, aby otestoval funkčnost software. Takto provedený test patří mezi činnosti manuálního testování (autor)
Open Source		Volně šířitelný počítačový software obsahující zdrojový kód. Přesná definice, viz (50).
Paretovo pravidlo		Paretovo pravidlo 80/20 znamená, že 80% spokojenosti zákazníka získate díky implementací 20% nejdůležitější funkcionality produktu, kterou zákazník skutečně potřebuje (autor)
Průzkumné testování		Návrh testů a exekuce zároveň. Je to opak skriptovacího testování (kdy jsou definované kroky před exekucí testů) (33)
Service Oriented Architecture	SOA	Spodní struktura podporující komunikaci mezi službami. Nejběžnější implementace SOA se stala komunikace mezi webovými službami. (41)
Refaktoring		Znamená *„přepracovat zdrojový kód, případně i architekturu tak, aby funkce zůstala nezměněna, ale udržovatelnost a čitelnost kódu (kvalita) byla výrazně vylepšena“* (38)
Tacitní znalost		*“Tacitní znalost je soubor dovedností, zkušeností, intuice, pravidel, principů, mentálních modelů a osobních představ konkrétního člověka”*(23)
Test-Driven Development	TDD	Technika vytváření testů a zároveň jádro agilních testovacích praktik (37)
Uživatelský scénář (use case)		Popis, jak by měl uživatel postupovat v systému, aby splnil úlohy. Popisuje sekvenci interakcí uživatele se systémem bez specifikace uživatelského rozhraní. (68)

11 Přílohy

Příloha 1: Dotazník

Question 1: Do you think that the testing process is faster thanks to Lean software development?

- Yes
- Yes, a little
- No, not much
- Not at all

Question 2: Do you think that our regular meetings have any good reason?

- Yes
- No

Question 3: Do you think that testing of our application is faster thanks to Lean software development?

- Yes
- Yes, a little
- No, not much
- Not at all

Question 4: Do you think that the application is tested better (more tests) thanks to Lean software development?

- Yes
- No

Question 5: Are you satisfied with testing process according to Lean software development?

- Yes
- Yes, a little
- No, not much
- Not at all

Question 6: What advantages did Lean software development bring you? (open question)

Question 7: What disadvantages did Lean software development bring you? (open question)

Question 8: Was it difficult to accept Lean software developoment principles for you?

- Yes
- Yes, a little
- No, not much
- Not at all

Question 9: Do you think that the whole development process speeded up?

- Yes
- No

Question 10: What would you change in testing team after acceptance of Lean software development? (open question)