Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Systémové inženýrství a informatika

Diplomová práce

Testování vývoje bankovního systému

Bc. Blanka Macháčková

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Testování vývoje bankovního systému" jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 31.3.2016 _____

Poděkování

Ráda bych touto cestou poděkovala paní Ing. Petře Pavlíčkové, Ph.D. za její odborné konzultace a panu Ing. Josefu Pavlíčkovi, Ph.D. za vedení práce. Dále své rodině a svému příteli za jejich podporu, zvlášť Ing. Janě Hausserové za trpělivost s čtením a opravou gramatických chyb.

Testování vývoje bankovního systému

Testing during development of the bank system

Souhrn

Diplomová práce se zabývá testováním softwaru pro společnost, jejíž činností jsou finanční služby. Důraz je zde kladen zejména na současný model vývoje, který byl nedávno změněn z vodopádového typu na typ agilní. Tento přechod je zde popsán za použití porovnání. Práce je následně zaměřena i na ideální složení scrum týmu. Takovýto tým by pak snadněji vyvíjel nové funkčnosti. V neposlední řadě jsou zde popsány nástroje, které jsou nezbytné pro vývoj a testování.

Cílem práce je zanalyzovat a navrhnout zlepšení postupu testování vývoje nových funkčností pro finanční systém.. Výsledkem této činnosti je návrh na zlepšení postupu nejen testování, ale i procesu, který je před samotným spuštěním testů.

Výstup práce pak činí spojení těchto analýz a návrh postupu testování, který by firmě ušetřil náklady.

Summary

The thesis deals with software testing in company, whose activities are financial services . Emphasis is placed on the current development model , which was recently changed from waterfall type to the agile type . This transition is described using the comparison . The work is focused on the ideal composition of the scrum team. Such a team could more easily develop new functionality. Finally, described the tools that are necessary for development and testing .

The aim is to analyze and propose improvements in process development testing of new functionality to the financial system. As a result of this activity is a proposal to improve the process , not only testing, but also a process that is itself before starting the tests.

Output work is the connection of these analyzes and design process testing, which would save the company costs .

Klíčová slova: testování, systém, analýza, metodika, testovací scénář, testovací set, agilní vývoj, scrum, bussines zadání, tester, proces

Keywords: testing, system, analysis, methodic, test case, test set, agile, scrum, bussines proposal, tester, process

Obsah

Slovník pojmů

ANSI	American National Standards Institution; americká standardizační org.
AST	Association for Software Testing
Backlog	Přiřazené nevyvinuté tiketů v systému JIRA; nashromážděná nedokončená práce
BSI	British Standards Institution; Organizace pro technicé standardy
Business	Oddělení firmy zabývají se vymýšlením nápadu pro větší spokojenost zákazníků a vyšší zisky firmy
Bug	Označení pro zadanou chybu v systému JIRA
DDTS	Distributed default tracking system; systém pro sledování chyb pro UNIX
DEMO	Demonstration; schůzka se zadavatelem za účelem představení vyvíjeného projektu
EDSAC	Electronic Delay Storage Automatic Calculator; počítač z poloviny 20. století
FIPS	Federal Information Processing Standard; standardy vyvinuté federální vládou Spojených států pro použití v počítačových systémech od nevojenských subjektů a vládních dodavatelů
IEEE	Institute of Electrical and Electronics Engineers; Společnost pracovníků v elektrotechnice a elektronice
ISO	International Organization for Standardization; mezinárodní organizace zabývající se tvorbou norem
ISO/IEC	Označení mezinárodní normy
JIRA	Vývojový management systém
MD	Man Day, práce vykonaná jedním člověkem za den (většinou 8hodin)
RAD	Rapid Application Development
Release	Vydání nové verze systému
SaaS	Software as a Service
SAST	The Swedish Association for Software Testing
SPICE	Software Process Improvement and Capability Determination

STaaS	Software Testing as a Service
Spira	Test management systém
Test result	Výsledek testovacího scénáře, kroku, setu
UML	Unified Modeling Language;
UNIX	operační systém

Seznam obrázků

Úvod

V dnešní moderní době si život bez technických vymožeností, všudy přítomnému internetu a služeb, které lze provozovat online, nedokážeme představit. Počítač, notebook, smartphone, tablet a další moderní pomůcky máme doma, pracujeme s nimi v práci, používáme je v běžném životě. Aby vše fungovalo tak jak má, jsou zde testy, které mají za úkol odhalit problémy a chyby, a ty se k běžnému člověku nedostaly.

Testování je nedílnou součástí běžného života každého z nás. Testují se výrobky, než jdou na prodejní pulty a podobným způsobem se testuje také software, který nám usnadňuje každodenní úkony

Již druhým rokem jsem zaměstnaná jako tester jedné středně velké firmy zabývající se finančními službami. I přes střední velikost je její systém velmi komplexní a propracovaný. Testování každého nového vývoje je velmi náročné. Proto bych se ráda v této práci zaměřila na zlepšení procesu testování nových funkčností.

V teoretické části se věnuji obecně tématu testování. Různým pohledů na testování, co vše je potřeba pro vývoj systému a následný proces testování. V hlavní části práce se zaměřuji na porovnání současného procesu vývoje a testování nových funkčností a mnou navrhnutého postupu. Popis tohoto procesu bude co nejvíce zobecněn z důvodu zachování firemního tajemství.

Při psaní této práce vycházím z praktických zkušeností a z teoretických poznatků týkajících se softwarového testování. Zároveň porovnávám testování v klasickém vodopádovém vývoji, které jsem absolvovala po nástupu do práce oproti agilnímu vývoji, kterého jsem momentálně součástí. Porovnání se tedy bude týkat původního typu vývoje s agilním vývojem nových funkčností do stávající platformy.

V práci je zároveň zmíněna důležitost propracovaného zadání projektu ze strany zadavatele. A také je zde okrajově zmíněna o spolupráce s businessem a hlavně o důležitosti business testování pro správné nastavení produktu.

1. Metodika a cíl práce

Metodika diplomové práce je založena na studii odborné literatury zabývající se řízením vývoje softwaru, metodikami testování a agilními metodikami. Tyto získané poznatky jsou následně využity v návrhové části. Ta je zastoupena analýzou současného stavu pomocí vlastních zkušeností a řízenými rozhovory se spolupracovníky.

Cílem práce je návrh zefektivnění postupů vývoje nové funkčnosti za využití současných zdrojů firmy. Stěžejní část práce se poté věnuje testování od vytvoření test analýzy, testovacích scénářů až po využití nástrojů pro efektivnější testování. V neposlední řadě je výstupem mé práce návrh postupu testování v přehledných diagramech.

2. Teoretická východiska/Přehled problematiky

2.1. Testování

Testování je široká oblast a důležitá součást lidského života. Dokonce by se dalo říci, že nejen toho lidského, ale i života výrobku, služby či softwaru. Testům podrobujeme, vědomě či nevědomě, všechny věci ve svém okolí již od útlého věku.

Testování může mít mnoho podob, odvíjejících dle odvětví, v kterém dané testování probíhá. Dle slovníku českého jazyka je jednou z definic následující text "*testování systému = vyhledávání chyb v systému, jeho prověřování; mat., stat. testování hypotéz = verifikace (uznání) nebo falzifikace (odmítnutí) hypotézy, testuje se tzv. nulová hypotéza proti alternativní hypotéze;"* (13). Jedná-li se o vědecké odvětví, může být testování nazýváno experimentem, pro produkty se zas může jednat např. o certifikaci a v době moderních technologií je stále více využívané testování softwaru. Pro účely práce je popsána pouze tato oblast.

2.1.1. Historie testování softwaru podle období

I když své okolí testujeme již od počátku vývoje lidstva, tak softwarové testování se datuje do mnohem bližší doby. O klasifikaci softwarového testování napsali v roce 1988 autoři D. Gelperin a B. Hetzel knihu *The Growth of Software testing* (17), kde popisují jak byla jednotlivá období v testování orientována.

Dle jejich knihy je první článek o kontrole programu napsán Alanem Turingem v roce 1949. Článek se dle uvedných zdrojů jmenoval *Checking a Large Routine* a byl publikován v *Report of a Conference on High Speed Automatick Calculating machines* na stranách 67 - 69 (5).

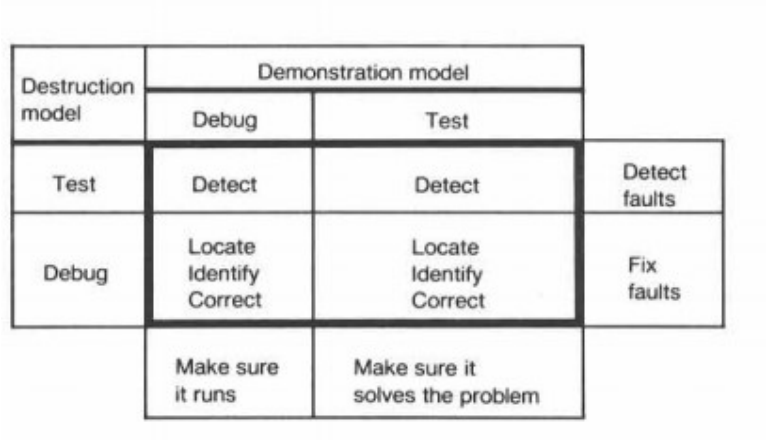
V tomto raném období se testování zaměřovalo více na hardware, než-li na samotný software. Ačkoliv se chyby softwaru objevovaly, tak až do roku 1956 bylo toto období orienotované na ladění (Debugging-Oriented) systému (5).

Mezi lety 1957 a 1978 se nachází období demonstrativní (Demonstration-Oriented), kdy roku 1957 Charles Baker oddělil ladění systému od testování v knize *Digital Computer Programming*. Podle jeho knihy měli programátoři dva úkoly: " Ujistit se, že program

běží" ("*Make sure the program runs")* a "Ujistit se, že program řeší problémy" ("*Make sure the program solves the problem"*). Pro testování se obrat "Ujisti se" stal cílem toho, že software splňuje své požadavky. V tomto období, kdy se software začal více využívat, uživatelé a manažeři firem stále více kladli důraz na "lepší testování", kdy chtěli efektivněji hledat problémy před distribucí softwaru. (5)

Během tohoto období význam testování a ladění (debugging) zahrnoval objevení, lokalizování, indentifikování a opravu chyb (viz. Obr. 1 Zdroj: The Growth of Software Testing (5). Dnes se ladění spíše využívá při testech koreknosti (sanity tests). (5)

Obrázek 1 Rozdíly v definicích "Test" a "Ladění"



Zdroj: The Growth of Software Testing (5)

Navazujícím krátkým obdobím dle Gelperina a Hetzela je destruktivně orientované období (Destruction-Oriented) mezi lety 1979 a 1982. V roce 1979 popsal G. J. Meyers v *Art of Software Testing* destruktivní model testování. Meyers definoval testování jako "proces exekuce programu za účelem nalezení chyb". Díky této definici se stalo primárním cílem testování nacházení chyb. Meyersovou myšlenkou bylo to, že pokud je za cíl ukázat to, že program je bez chyb, tak může tester podvědomě vybrat taková data, která nezpůsobí selhání programu, je-li ale za cíl vyhledání chyb, testovací data mají mnohem větší pravděpodobnost tyto chyby odhalit. (5)

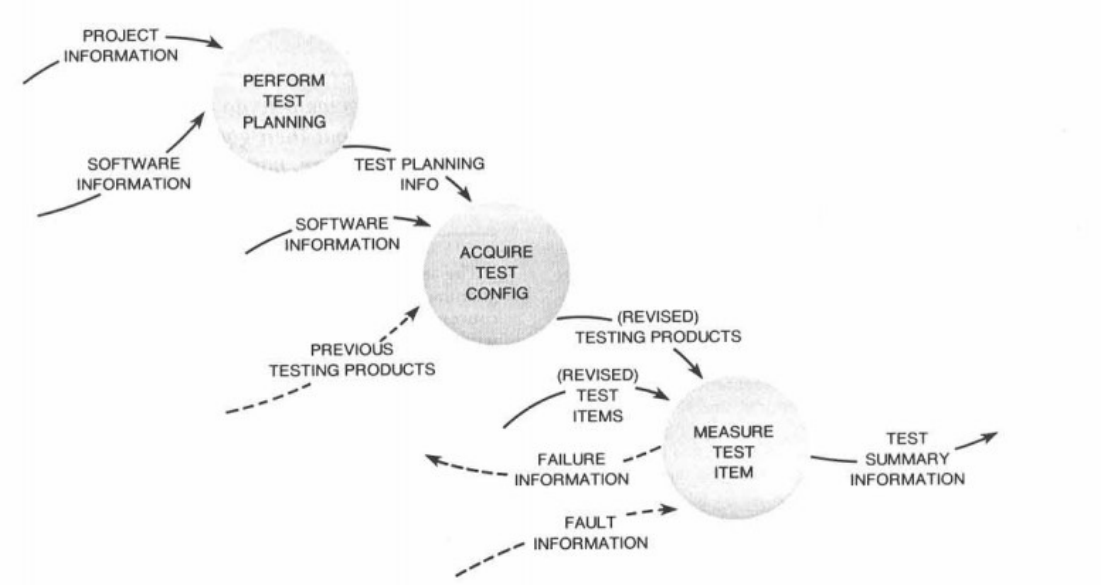
Další krátké období mezi lety 1983 a 1987 začalo po vydání příručky *Guideline for Lifecycle Validation, Verification and Testing of Computer Software* od Institutu pro počítačové vědy a technologie v National Bureau of Standards a bylo orientováno

vyhodnocování (Evaluation-Oriented). Příručka se zaměřovala na federální normu pro zpracování informací (FIPS) a popisovala její metodologii. Metodologie obsahovala analýzu, revizi a testování, které zajišťovalo ohodnocení softwaru během jeho životního cyklu. Příručka také doporučovala tři skupiny hodnotících technik (základní, komplexní a kritická). (5)

Posledním obdobím popsaným autory D. Gelperin a B. Hetzel je rok 1988, kdy kniha vyšla. Toto období nazvali obdobím orientovaným na prevenci (Prevention-Oriented). Testy měly prokázat nejen, že software splňuje zadané specifikace, ale také detekovat chyby a tím zabránit jeho poruchám.(5)

Testování se dostalo více do popředí a byly zformulovány i metodiky, jak na něj nahlížet. Jednu z možností jak pozorovat základní tok dat v procesu testování (viz. Obr. 2) uvedli autoři ve své knize, kde se pokusili shromáždit metody pro nadcházející období.(5)

Obrázek 2 Základní tok dat v procesu testování



Zdroj: The Growth of Software Testing (5)

2.1.2. Historie testování softwaru dle vývoje

Další možností jak se dívat na historii testování softwaru, je dle milníků vývoje a to nejen přístrojů, pro které testování je prováděno, ale samotné komunity testerů, kteří se s

hardwarového testování přesunuli na testování softwarové. V průběhu let pak na základě vyvinuté technologie zlepšovali metodiky pro testování.

18. století

Tato historie začíná okolo roku 1820 vynálezem diferenciálního motoru (rok 1822), který byl jedním z prvních mechanických počítačů. Čas plynul a další milník se objevil až 40 letech 18. století, kdy roku 1843 John Stuart Mill publikoval ve svém díle *Systém deduktivní a induktivní logiky* (*Methods of inductive reasoning)* uvádí pět metod induktivního uvažování. Při indukci jsou v premisách zachyceny všechny předměty množiny. Do 40 let 18. století se významně též zapsaly poznámky k dílu *Analytický stroj* (*Analytical Engine*) od autorky Ady Lovelace. Tyto poznámky obsahují metody programování a program pro výpočet posloupnosti Bernoulliho čísel. (9)

Historie se odmlčela až do roku 1878, kdy Thomas Alva Edison pojmenovává chybu v systému označenám bug (angl. brouk) ve svých dopisech Theodorovi Puskasovi. Za necelých 20 let později, roku 1896 je založena IBM. Tento název dostala mnohem později. Její původní název je Tabulating Machine Company by Herman Hollerith v Broome Contry. Svůj současný název dostává až roku 1924. (9)

Léta 1900 - 1959

Začátkem nového století italský ekonom Vilfredo Pareto vytvořil matematický vzorec, kterým popsal nerovnoměrné rozdělení bohatství, tzv. Paretův princip (1906). Rok 1939 pak přináší tzv. Shewhartův cyklus od autora Waltera Andrewa Shewharta. Ten později zpopularizuje Edward Deming a rozvinul původní Shewhartovu myšlenku do oblasti řízení kvality. (9)

Období 2. světové války přineslo mnoho nových vynálezů a k jejímu konci (rok 1944) byl postaven první plně automatizovaný počítač s označením Harvard Mark I, známý též jako IBM Automatic Controlled Calculator (ASCC). Další rok (1945) John von Neuman publikoval svůj popis logické konstrukce počítače. Na návrhu von Neumanovi architektury je dnes založena většina počítačů. (9)

Roku 1946 je na konferenci v Londýně založena mezinárodní organizace pro standardizaci (ISO) sloužením organizací ISA (International Federation of the National Standardizing Associations) a UNSCC (United Nations Standards Coordinating Commettee). Alan Turing ukončil toto desetiletí svým příspěvkem *Checking a large routine*, kde navrhuje, jak je možné kontrolovat rutinu. (9)

Padesátá léta minulého století pokračovala ve znamení Alana Turinga. Krátce před svou smrtí představil koncept testování umělé inteligence, tzv. Turingův test. Tento test spočíval v pokládání otázek počítači a vyhodnocení, zda je počítač schopen „myslet“ (přesáhnout svůj původní kód). Roku 51 byly pak představeny knihy jako *Juranova příručka kontroly kvality (Juran's Quality Control Handbook)*, zabývající se kavalitou řízení (quality managemnet) nebo kniha *Total Quality Control* od Armanda Vallin Faigenbauma, kde popisuje, že produkt by měl uspokojit zákazníka jak v jeho aktuálních potřebách, tak v jeho očekávaných potřebách. Posledním dílem dle Meertse, který ten rok zasáhl do vývoje a formování testování byla knihla *First practical text on programming* od autorů Maurice V. Wilkse, Davida J. Wheelera a Stanleyho Gillse, kde autoři popsali své zkušenosti z navrhování EDSAC. (9)

IBM roku 1952 vydává *The IBM 701 Defense Calculator*, který odstartoval úspěšnou a populární sérii IBM 700. Tento počítač obsahoval dvě elektrostatické jednotky a elektrickoou analyti kou a řídící jednotku. Jednotka se pronajímala okolo 16 000 dolarů za měsíc. Trvalo 4 roky než byl vydán manuál pro IBM počítač, konkrétně IBM 704, s názvem *The first Fortran manual*, který popisuje promněné, funkce, výrazy a základní syntaxy jazyka. Rok 57 pak přinesl přinesl článek zabývající se přímo testování, kdy Charles L. Baker vydal recenzi ke knize *Digital Computer Programming* od autora Dana McCrakena s názvem *Program testing vs debugging*, ve kterém oddělil testování od ladění. Kniha D. McCrakena by se dala považovat za jeden z prvních ucelených návodů o programování. Ve své knize doporučoval zavádět tzv. pauzy (break points), které mohly ověřit funkčnost pouze části programu. (9)

První testovací tým vznikl roku 1958 a byl zformulován Geraldem M. Weinbergem. Tento tým pracoval jako management operativního vývoje pro projekt Mercury, což byl první program pro let člověka do vesmíru ve spojených státech. Padesátá léta ukončují díla od D.G. Malcolma, J.H Rosebooma, C.E. Clarka a W. Fazara, kteří představili svoji metodu

PERT (Program Evaluation an Review Technique) a autoři James E. Kelley a Morgan R. Walker ve své práci *Critical-Path Planning and Sheduling* nabídli pohled na plánování a organizaci kompexních projektů. Kritická cesta pak byla navržena jako optimalizace přímých nákladů a plánování. (9)

60. léta až 80. léta 19. století

Šedesátá léta 19. století pak představovala velký rozkvět v technické literatuře, tak ve vývoji počítačů. Kniha *Computer Programming Fundamentals* od autorů Gerald Weinberg and Herbert Leeds z roku 1961 obsahuje kapitolu o softwarovém testování. Autoři se v ní dohadují, že testování by mělo dokázat schopnost počítačového programu se adaptovat místo schopnosti zpracovávat informace. Téhož roku vyšel článek *Tabular Form* od autora Burtona Grada, který představil rozhodovací tabulky. (9)

Roku 1963 popsal Robert J. Rossheim standardy symbolů vývojového diagramu. Jeho práce byla publikována pro *American Standards Association* (pozdější ANSI). Rok po této publikaci matematikové John. G Kemeny a Thomas E. Kurtz rozběhli první programovací jazyk s názvem Basic na Dartmouthské koleji. Basic je zkratkou pro ***B**eginner's **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode.* Roku 1967 je představen pojem softwarové inženýrství. Tento termín se zavedl na konferenci NATO, kde se řešilo téma "problém softwaru", kdy Brian Randell a Peter Naur vybrali toto spojení jako provokativní výraz toho, že i software musí být vyráběn (manufacture) na základě teoretických základů a praktické disciplíny, které jsou tradiční pro odvětví strojírenství. (9)

Rok poté se další zpráva z NATO zmiňuje o jakosti softwaru (Software Quality Assurance). Je v ní uveden dokument od autora Foberta W. Bemera s názvem *Kontrolní seznam pro plánování výroby softwarového systému* (*Check list for planning software system production*), ve kterém Bemer mimo jiné má v dotazníku vedenu otázku: "*Je výrobek testován pro to, aby byl co nejužitečnější pro zákazníka než aby odpovídal funkčním specifikacím" ("Is the product tested to ensure that it is the most useful for the customer in addition to matching functional specifications?").* (9)

Stejný rok je založena i společnost Intel Gordonem E. Moorem a Robertem Noycem a jejich produktem se stává statická paměť s náhodným přístupem (SRAM) a Fiedrich

Ludwig Bauer zavádí termín "Softwarová krize", která se odkazuje na velkou obtížnost psaní správných a srozumitelných a ověřitelných počítačových programů. Conwayův zákon, který je zformulovám téhož roku, který říká "*Každá organizace navrhující systém nevyhnutelně vyprodukuje návrh, jehož struktura bude kopií komunikační struktury této organizace."* autorem tohoto článku je Melvin E. Conway. Konec šedesátých let pak přináší článek *Datová struktura diagramů* (*Data structure Diagrams*) od autora Charlese W. Bachmana, kde zavání grafickou techniku pro modelování entit a jejich vztahů. (9)

Sedmdesátá léta otvírá Winston Royce svým popisem modelu vodopádu, který publikuje v novinách *Řízení vývoje velkých softwarových systémů* (*Managing the Development of Large Software System*). Ve svém článku zároveň Royce zmiňuje nedostatky tohoto sekvenčního procesu vývoje softwaru. Rok 1971 pokračuje publikací dalšího modelu, tentokrát modelu relačního od autora Edgara F. Codda, který pracuje pro IBM. Ve svém příspěvku *Relační model dat pro velké sdílené databanky* (*A Relational Model of Data for Large Shared Data Bank*) dává základy pro budoucí relační databáze. (9)

Rok 1971 přináší článek Ruicharda Liptona s názvem *Diagnostika poruch počítačových programů* (*Fault Diagnosis of Computer Programs),* kdy autor navrhl počáteční metodiku testování mutací pro jednotlivé testy. Za tyto mutace jsou považovány malé části kód, které jsou zmněny za účelem testování kvality. (9)

Autor William Hetzel publikuje po symposiu v Chapel Hill knihu s názvem *Program testování metod* (*Program Test Methods*), kdy se tato kniha zaměřuje na problém testování a validace. Ve stejném roce je založena společnost Compuware Corporation se sídlem v Detroitu, která se zaměřuje na zveřejňování testovacích nástrojů a technických pomůcek. Další událostí roku 1973 byl příspěvek L. Nolana s názvem *Stádia růstu modelu* (*Stages of growth model*), kdy tento model popisuje stádia, kterými společnost pochází při zavádění informačních technologií do svého obchodního procesu. (9)

Roku 1975 se vedle IBM objevuje nová společnost, založená Billem Gatesem a Paulem Allenem s názvem Miscrosoft a stává se jednou z největších softwarových společností na světě. V roce 2009 zaměstnává přes 10 000 testerů. (9)

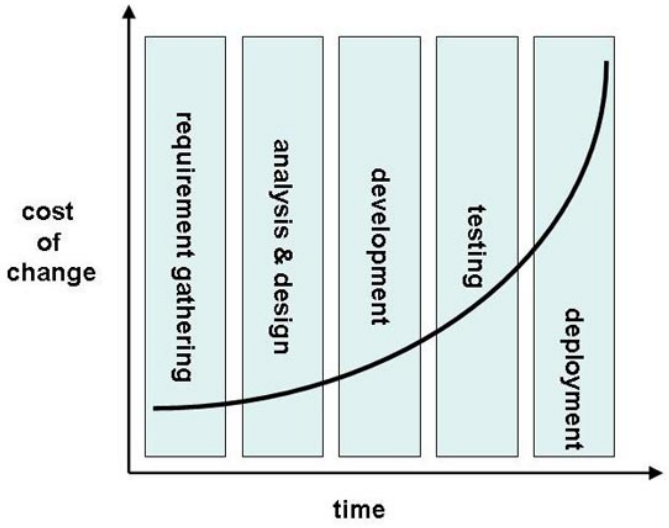
Téhož roku publikovány *Zákony nespolehlivosti* (*Law of Unreliabilty)* od Toma Gilba, kdy jako jeden z prvních definoval vztah mezi lidskou chybou a chybou systému a *Brooksův zákon* (*Brook's Law*) od Freda Brookse, který popisuje, že pokud má projekt zpoždění, může příchod nového člena na výpomoc paradoxně tento skluz dále prohloubit. (9)

Kromě "zákonů" vyšla i kniha Geralda M. Weinberga *Úvod do obecného systémového myšlení* (*An Introduction to General Systems Thinking*), kde autor nabízí jasný pohled na teorii systému. Podle Weinberga je jako pokus "učit správné myšlení, když chybí nebo jsou zavádějící štítky (labbles)." (9)

K velkým firmám na trhu se roku 1976 připojuje Apple založený Steavem Jobsem, Stevem Wozniakem a Ronaldem Waynem. Stejný rok navrhuje Michael Fagan z IBM zavendení inspekcí kódů (revizí), aby se snížily náklady na přepracování. V oblastní testů Thomas J. McCabe zavádí základní testovací cesty jako je testovací technika "bílí box" (White box). Glendford Myers zas ve své knize *Softwarová spolehlivost:Principy a praxe* (*Software Reliability: Principles and Practices*) mimo jiné zmiňuje to, že "cílem testerů je, aby program selhal".

Barry Boehm pak ve svém příspěvku *Softwarové inženýrství (Software Engineering)*, zveřejněném v prosinci roku 76, vydává jeho nákladovou křivku změně (Cost-of- change curve), která, jak ukazuje Obrázek **3** Křivka nákladů na změnu, roste v čase exponenciálně. (9)

Obrázek 3 Křivka nákladů na změnu



Zdroj: Improving Application Quality Using Test-Driven Development (10)

Kniha *Sotwarové metriky* (*Software Metrics*) od Toma Gilba je považována za zdroj pro velké množství těchto metrik. Autoři Alexander a Sára Ishikawa a Murray Silverstein ve své knize *Návrhový vzor: Města, budov, stavba Christopher* (*A Pattern Language:Towns, Buildings, Contructruction Christopher*), vydáné roku 1977, představují pojem "návrhový vzor" (design pattern), kdy tento vzor je obecný a může být opakovaně použit pro řešení běžně vyskytujícího se problému v softwarovém designu. (9)

Sedmdesátá léta ukončuje tvorba Glendorda Myerse. Meyers ve své knize *Umění testování softwaru* (*The Art of Software Testing)*, která je považována za první knihu, která je pouze o softwarovém testování, představuje koncept testování černé skříňky (black box). (9)

Roky 1980 - 1999

IBM vydává roku 1981 první osobní počítač IBM 5150, který nastavuje standardy pro trh s počítačovými produkty. Téhož roku James Martin a Clive Finkelstein představují svouji CRUD (Create- vytvořit, Retrieve - obnovit, Update - aktualizovat, Delete - smazat) analýzu jako metodu pro analýzu designu. (9)

Roku 82 je publikována kniha autora Wiliama Edwarda Deminga *Z krize: Kvality, produktivity a konkurenční pozice (Out of the Crisis: Quality, Productivity and Competitive Position*). Deming zde navrhuje, že kvalita musí být definována z hlediska spokojenosti zákazníků. (9)

První verze standardu pro dokumentaci softwarového testování IEEE 829 (Standard for Software Test Documentation) je vydána roku 1983, kdy tato norma specifikuje podobu osmi forem dokumentů pro použití ve stejném počtu fází softwarového testování. O rok později vydává James Martin *Manivest informačního systému* (*An Information Systems Manifesto*), kde ukazuje, že více než polovina chyb má kořeny ve špatně definovaných požadavcích. (9)

Rok 1985 ovládl Microsft vydáním první Excel, kdy tato verze byla napsána pro Macintosh od společnosti Apple. Až za dlouhé dva roky poté (1987) se dočkal i systém Windows. MS Excel se stal díky své popularitě vlajkovou lodí společnosti a i dnes se často využíva jako testovací nástroj. (9)

První edice knihy *Testování počítačového softwaru (Testing Computer Software)* od autora Cema Kanera zavedla nové standary v testování a zároveň v ní autor poprvé užil termín "průzkumné testování" (exploratory testing). Téhož roku je vyvinut Qualtrakem systém DDTS (Distributed default tracking system) pro UNIX trh. (9)

Rok 88 také přinesl Spirálový model od Barryho Boehma, který mohl být alternativní řešením k vodopádovému modelu. Barton Miller zase představil termín 'fuzz' (česky chmýří) testování, kdy za použití náhodných, nestrukturalizovaných dat je testována bezpečnost systému. (9)

Mercury Interactive je založena Amnon Landam a Arye Finegold rouk 1989 v Kalifornii. Společnost vydává mnoho produktů pro automatické testování nebo na výkonostní testování, jako je nástroj LoadRunner, než se v roce 2006 stává akvizicí společnosti HP (Hewlett Packard). (9)

V devadesátých letech provádí Boris Beizer svoji taxonomii - klasifikaci chyb v druhém vydání knihy *Techniky testování softwaru (Software Testing Techniques*), kde zároveň zavádí obrat "pesticidový paradox", kterým popisuje fenomén podobný užívání pesticidů proti broukům (angl. bug). Tento paradox říká, že čím více je software testování, tím se stává více imuní vůči testům. (9)

Roku 1991 je založené fórum za účelem diskuze o testovacích nástrojích, technikách a zkušeností mezi NASA, průmyslem a akademickou obcí. James Martin představuje ve své knize *Rapid Application Development* stejnojmený princi vývoje, kdy tento typ vývoje (RAD) je schopen vyvíjet mnohem rychleji a s kvalitními výsledky, než klasický životní cyklus vývoje. (9)

Jeff Sutherland, John Scumniotales a Jeff McKenan poprvé nazývají vývoj v týmu termínem Scrum (1993). Martin Pol, Ruud Teunissen a Erik van Veenendaal zveřejňují roku 1995 svůj přítu k řízení testů u strukturovaného testování. Ve Švédsku je zformována Annou Eriksson a Sigrid Eldhem Asociace pro softwarové testování (SAST), jako platforma pro výměnu zkušeností a znalostí z oblasti testování. (9)

Nejnavštěvovanější internetové stránky zakládá Larry Page společně se Sergeyem Brinem a dávají jim název Google. Autoři David Cohen, Siddharta Dalal, Jasse Parelius a Gardner Patton ve svém příspěvku *The Bominatorial Design Approach to Automatic Test Generation* (1996) představuji pairwise testy, které jsou založeny na kombinatoriční teorii. (9)

Rok 1997 přináší založeno dalšího společenství TestNet pro výměnu znalosti o softwarovém testování, James Rumbauch, Grady Booch a Ivar Jacobson vytvářejí UML (Unified Modeling Language) jazyk. (9)

První možnost získat evropský certifikát pro sofwarové testery, je vytvořena Britskou zkouškovou komisí pro informační systémy roku 1998. Stejný rok je vydána i populární

verze nástroje, pro zaznamenávání chyb Bugzilly, která je původně vyvinuta Terrym Weissmanem vznikající projekt Mozilla.org. Jsou vydány standardy pro komponety softwarového testování a vychází verze 1.0 Apache jMeter, který je volně dostupný (open source), jako nástroj pro testovní výkonu a webových služeb. (9)

V posledním roce před začátkem nového tisíciletí Cem Kaner, Jame Bach, Brian Marick a Bret Pettichord zakládají školu testování na základě kontextu (Context-Driven school of testing), Graham a Fewster vydávají knihu *Automatizace testů softwaru(Software Test Automation)*, kde se zabývají problémem automatických testů a byla vydána metodologie TestFrame Hansem Buwaldou, Dennisem Jansenem a Iris Pinksterovou. (9)

Nové tisíciletí (2000-2012)

V roce 2000 publikuje Jonathan Bach v časopise *Software Testing and Quality Engineering* metodu relačního testování. Metoda kombinuje odpovědnost s průzkumným testováním, londýnské nakladatelství *Test Publishing Limited* vydává první publikaci magazínu *Professional Tester*, kdy tento časopis byl opětovně vydán až v roce 2010. (9)

Na konferenci YAPC (Yet Another Perl Conference) zařazuje Mark Jason Dominus tzv. Bleskovou řeč (Lightning talk) do programu konference. Na této konferenci byla dodržována tato blesková komunikace, která znamenala pěti minutové diskuze, které byly ukončeny gongem. (9)

Agilní manifest (Agile Manifesto) je zveřejněn na konference v Utahu v roce 2001, kde 17 zástupců různých vývojových metodik definují 12 zásad agilního softwaru, James Bach, Cem Kaner a Bret Pettichord vydávají své zkuše s testy řízenými na základě kontextu Ponaučení vyplývající z testování softwaru (Lessons Learned in Software Testing). (9)

Téhož roku je poprvé použita zkratka SaaS pro Software as a Service (Software jako Služba) v článku *Software as Service: Strategic Backgrounder*, kdy model Software as a Service definuje distribuci aplikací přes internet a OpenSTAT se stává první verzí otevřeného testovacího nástroje pro testování zátěže a výkonu webu, která je vydána francouzskou skupinou Cyrano SA. (9)

V listopadu 2002 je založena v Edinburghu Rada pro mezinárodní kvalifikaci softwarového testování - ISTQB (The International Software Testing Qualifications Board) a Australská společnost Atlassian Software vydává první verzi později populárního nástroje pro vyhládávání chyb JIRA (JIRA 1.0.). Software stále více čelí útokům a tak své knize *Jak prolomit software: Praktický průvodce testování* (*How to Break Software: A Practical Guide to Testing*) James Whittaker popisuje celkem 23 útoků k odhalení softwarové chyby. Útoky jsou navrženy jak pro uživatelské rozhraní, tak pro celý systém. (9)

Testování se po vydání knihy Testy řízený vývoj (Test-Driven Developmentt: By Example) od Ken Becka mění tak, že testy jsou nyní psány předtím než je funkcionalita naprogramována a Ward Cunningham vytváří rámec pro integrační test (Framework for Integrated Test), jako otevřený nástroj pro automatizované uživatelské testy. Nástroj integruje vývoj testů do specifikací a spuštění testů do programování za pomoci příslušenství. To dává zákazníkům a programátorům možnost precizně komunikovat o svém software. (9)

Roku 2003 Bret Pettichord vydává dokument, ve kterém klasifikuje myšlenky testování softwaru do pěti škol, je založena Asociace pro testování softwaru (Association for Software Testing) - AST jako nezisková organizace zaměřená na podporu rozvoje profesionality v testování softwaru. V říjnu 2003 je vydána první část ISO/IEC, známá také jako SPICE (Software Process Improvement and Capability Determination - Zlepšení procesů softwaru a schopnosti určení), která je rámcem pro hodnocení procesů. (9)

Popularní webový nástroj Selenium je vyvinuto Jasonem Hugginsem v ThoughtWorks v roce 2004. Stejný rok vydání má Confluence 1.0, kterou vytvořila Australská společnost Atlassian 25. března 2004. Aplikace je v podstatě wikipedie pro podnikový intranet. (9)

Test Maturity Model Integration (TMMi) je vydán nadací TMMi. Tento model slouží jako standard pro hodnocení a zlepšení procesu testování. Tento rok (2005) přináší také vydání SoapUI na SourceForge v září 2005 a brzy na to se stává oblíbeným nástrojem pro testování webových služeb. (9)

První CAST (Conference of the Association for Software Testing - Konference sdružení pro testování softwaru) konference se koná v červnu 2006 v Indianapolis. Jejími hlavními řečníky jsou James Bach a Cem Kaner. (9)

V květnu 2007 ISO formuje pracovní skupinu pro vypracování nového standardu o testování softwaru - ISO 29119. Norma nahrazuje množství stávajících IEEE a BSI standardů pro softwarové testování. (9)

Softwarové testování jako služba (STaaS - Software Testing as a Service), tento termín vytváří Leo van der Aalstem v prezentaci na konferenci softwarové kvality. STaaS je model testování softwaru používaného k testování aplikace jako služby poskytované zákazníkům přes internet. (9)

Při svém úvodním proslovu na konferenci Google Test Atomation 2011, nazvané Test is dead (test je mrtvý), Alberto Savoia uvádí, že tradiční testování softwaru je mrtvé nebo na ústupu. Navrhuje, že začínající společnosti kladou menší důraz na testování a naopak větším důraz na uvolňování software rychle a rovnoměrně. (9)

Ve své knize *Specifikace příkladem* (*Specification by Example*), Gojko Adzic navrhuje řadu procesních vzorů, které usnadňují změnu v softwarových produktech k zajištění účinného dodání produktu. (9)

2.1.3. Chyba softwaru

Snad žádný software se nevyrobí bez chyby. Historie dává tomu tvrzení za pravdu, když zaznamenala hned několik významných chyb. Již v roce 1947 potřeboval počítač s označením Mark II neustálou péči programátorů, aby jej udrželi v provozu. Svůj velký debakl si zažila i společnost Disney, kdy roku 1944 vydala hru pro děti s názvem *The Lion King Animated Story book*. Po obchodní stránce byla hra úspěšná, ale pouze do doby než ji zákazníci spustili. Disney tehdy opomenul testování na různých model tehdy dostupných počítačů a hra fungovala pouze na typech počítačů, na kterých byla vyvinuta a testována. (11)

Chyby se tedy nevyhýbají ani špičkovým společnostem, jako je např. NASA, kdy se 3. prosince 1999 pokusila přistát na povrchu Marsu. I když jednotlivé funkčnosti modulu fungovali bez chyb, tak zde byl opomenut faktor integrace a tzn. end-to-end test. (11)

Jak je výše již naznačeno, chyba je selhání softwaru. I když je tento popis chyby příliš obecný. Z pohledu testů může být selhání způsobeno mnoha faktory. Těmito faktory mohou být následující možnosti (11):

- Vada/ závada/ selhání - nemusí být přímo chybou softwaru
- Problém / omyl / chyba - nastávají lidským faktorem vývoje
- Událost / odchylka / anomálie / nekonzistence - tyto faktory mohou být způsobeny neočekávanou reakcí mezi vyvíjenými částmi

Autoři knihy *Foundations of software testing: ISTQB certification* (6) rozčlenili chyby a defekty (error and defect) do přehledného diagramu, kdy každý sloupec s označením Requirement (požadavek) představuje průběh vývoje a výskyt chyby v něm.

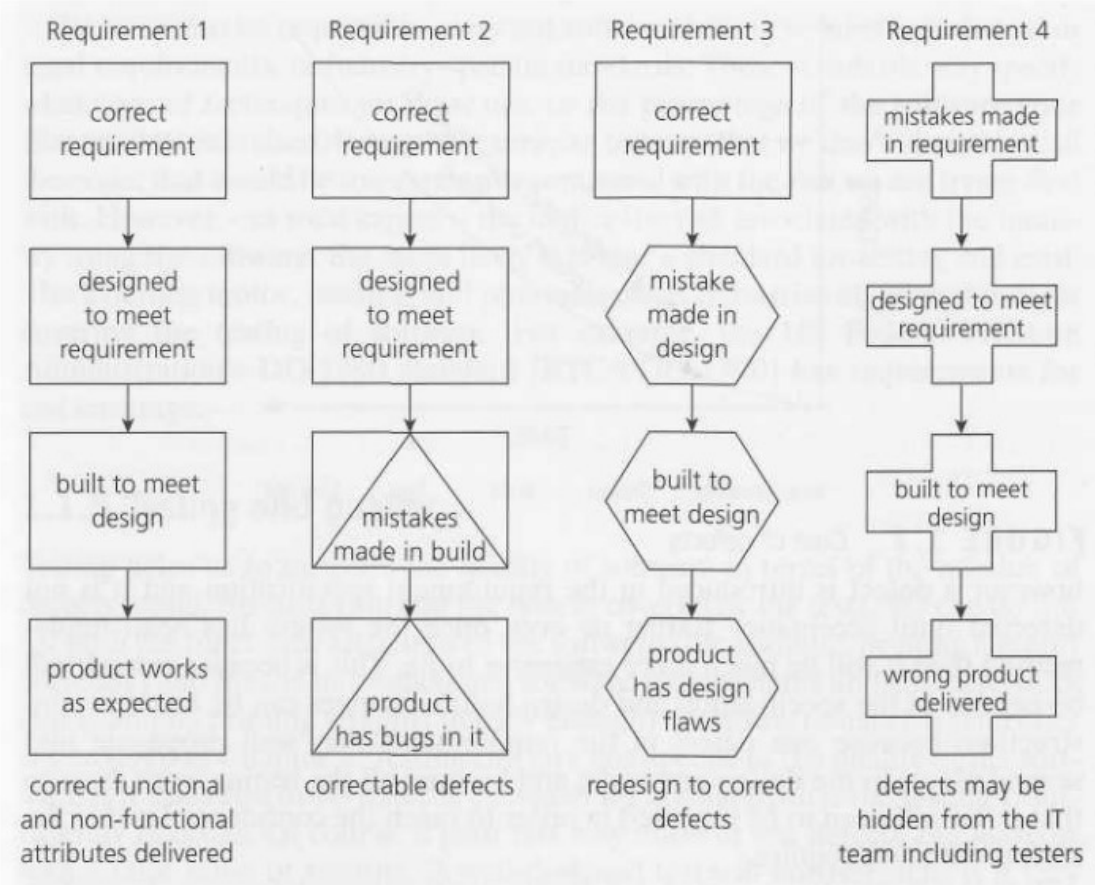
Requirement 1 představuje stav, kdy je software naimplementovaný správně - zákazníkovi požadavky jsou splněny, dle jeho očekávání. V procesu se nevyskytla žádná chyba. Produkt funguje podle zadaných specifik.

Další sloupec již vykazuje chybu. Počáteční kroky jsou splněny korektně, požadavky klienta jsou společností pochopeny, návrh odpovídá těmto specifikacím. Problém nastávám až při finálním sestavením produktu (tzv. build). Díky tomu software obsahuje chyby a je třeba tyto chyby odstranit.

Předposlední zobrazená situace ukazuje scénář, kdy kromě počátečního zadání, které chyby neobsahuje, je zbytek projektu naimplementován dle chybného pochopení požadavků klienta. Nejsou-li specifikace dodrženy, následný produkt odpovídá pouze návrhu od řešitelské společnosti a dodaný software pak nemusí klientovi vyhovovat. V tomto případě je třeba projekt upravit (redesignovat) na původní požadavky a opravit vzniklé defekty.

Requirement 4 je pak ukázkovým případem, jak by neměl vývoj vypadat. Chybu již obsahuje samotný požadavek klienta, ta se přenáší celým procesem, až k finálnímu produktu. Ten neodpovídá potřebám klienta, ale odpovídá jeho požadavků. Při integraci se pak mohou vyskytnout problémy, které mohli být ukryty jak před vývojovým týmem, tak před týmem testerů.

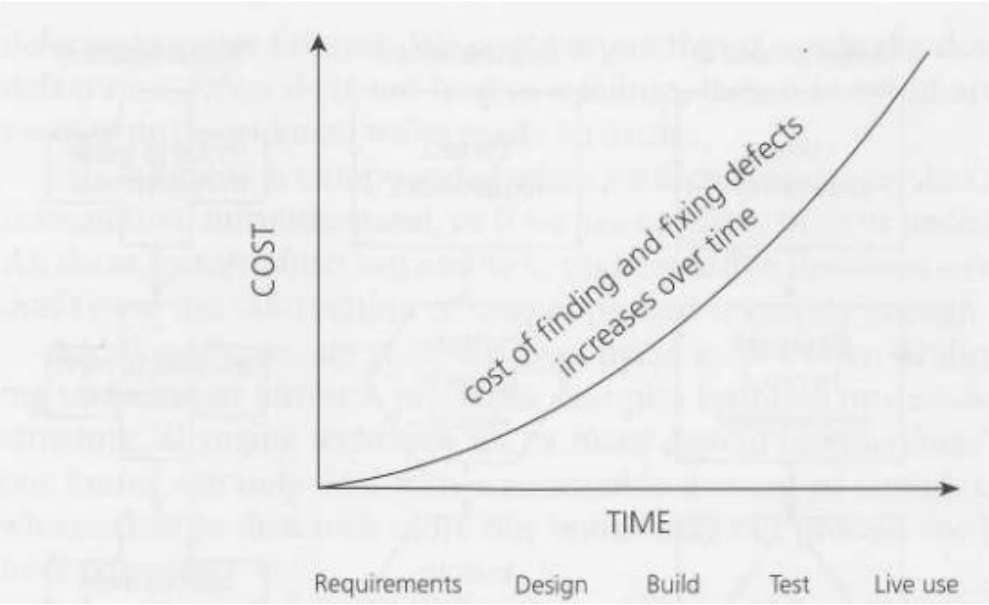
Obrázek 4 Typy chyb (error) a defektů



Zdroj: Foundations of software testing: ISTQB certification (6)

Vezmeme-li v úvahu, že náklady na nalezení a opravu těchto defektů se v průběhu vývoje zdražují, tak je velmi důležité, aby se celý projekt řídil podle prvního sloupce, kdy náklady dosahují minima. V takovém případě každý, kdo se podílel na daném projektu, a to od samotného zadání (vytvoření specifikace), odvedl svoji práci

Obrázek 5 Náklady na defekt



Zdroj: Foundations of software testing: ISTQB certification (6)

2.1.4. Test designér, tester a test manažer

Jak z předchozí kapitoly 2.1.3vyplývá, tak snahou firem je eliminovat co nejvíce chyb před vydáním svého softwaru. Pro tuto činnost si najímají test designéry a testery, kteří by měli software co nejlépe zanalyzovat a co nejpřesněji vydefinovat možné scénáře použití softwaru v praxi. Prací test designéra je na základě zadání a dokumentace vytvořit testovací scénáře, které mohou nastat při využívání softwaru. Tyto scénáře se netýkají pouze běžného využívání, ale hlavně testují alternativy, které může uživatel softwaru z jeho funkčnostmi provádět. Zaměřují se na to, aby dle jejich scénářů mohl tester co nejlépe otestovat odolnost a flexibilitu softwaru.

Tester následně při testech prochází tyto scénáře od nejpravděpodobnějších - tím testuje, zda je naprogramovaný kód natolik stabilní, aby mohl začít testovat jeho odolnosti alternativními scénáři. Ideální tester by měl zvládat i roli test designéra a tím pádem velice specifický typ myšlení. Měl by se tedy vyznačovat tím, že:

- má velkou fantazii, co se týká možností využití testovacího softwaru
- se snaží objevit skryté možnosti. Nejsou mu cizí otázky typu "Co se stane, když zkusím...?", "Pokud zvolím tuto nesmyslnou kombinaci, jak se software zachová?"
- je zvědavý a má dobré logické i technické myšlení. Nemusí být odborník přes programování, ale měl by si rozumět jak se zadavatelem, tak s vývojářem.
- je pečlivý a upozorňuje na možné nedostatky. Nedostatky nejen samotného softwaru, ale i třeba jeho návrhu.
- je komunikativní. Dokumentace k projektům neobsahuje vždy všechny potřebné informace a proto je třeba tyto informace vyhledávat, komunikovat se zadavateli či konzultovat s vývojáři.
- je progresivní. Neustrne ve svém vývoji, jelikož technologie jdou stále dopředu a je třeba za nimi nezaostávat.
- je psychicky odolný. Nese totiž odpovědnost za přehlédnuté chyby a zároveň by měl pravdivě vylíčit stav softwaru i např. vedení firmy.

Test manažer následně dohlíží na práci test designérů a testerů. Sleduje termíny a vývoj testů. Rozděluje své podřízené do projektů na testy softwarů dle kapacit svého oddělení. Měl by mít všechny vlastnosti testera a navíc k nim:

- přehled o systémech firmy / o vyvíjeném projektu
- manažerské schopnosti
- personální schopnosti

2.1.5. Testování softwaru

Na základě analýzy od test designéra a vytvořených testovacích testů jsou spouštěny jednotlivé testy. Tyto testy nemusí vycházet pouze z analýzy, ale mohou být napsané samotným vývojářem daného kódu, který se takovýmito testy odstraňuje první možné chyby. Jeho kód by správně měl být poslán i k revizi jinému vývojáři, aby se omezilo výskytu příliš složitého kódu nebo se odhalila logická chyba kódu. Tyto revize kódu zároveň zaručují, že je software vyvíjen dle standardů firmy.

Jak je již asi z předchozího odstavce zřejmé, existuje spousta typů testů, kterými se dá software otestovat. Testy, které jsou využity, se liší dle specifikace softwaru. Každé testování by ale mělo zajistit verifikaci, validitu, kvalitu a spolehlivost.

Při verifikaci se vyhodnocují zadané specifikace. Validací se určuje, zda software vyhovuje požadavkům uživatele. Mohlo by se zdát, že tyto pojmy jsou stejné, avšak zadané specifikace ne vždy mohou vyhovovat uživateli. Společnosti může vyvinout například účetní software přesně dle svojí specifikace, avšak pro uživatele - účetní je tento software nepoužitelný jelikož obsahuje špatné typy sazeb.

Předchozí příklad se hodí i pro další pojem, kterým je kvalita. Pokud účetní program neobsahuje správné typy sazeb, nedá se považovat za kvalitní. Kvalita nám určuje, jaký stupeň dokonalosti dosahuje dodaný software u zákazníků. Spolehlivost pak určuje, jak je dodaný produkt stabilní. Jako příklad může sloužit i událost z roku 2015, kdy se mezi nejočekávanější hry zařadil Batman: Ark ham Knight. Již prvních pár dní po jeho vydání se začali objevovat stížnosti. Hra se potýkala s technickými, ale i s výkonnostními problémy. Její vydavatel byl nakonec nucen hru z prodeje dočasně stáhnout. Faktory kvality i spolehlivosti byly tedy na nízké úrovni, stejně jako i její validace.

2.2. Způsoby testování softwaru

Pro dosáhnutí kvality, spolehlivosti, ale i validace a verifikace, slouží testy. Testy lze na základě jejich spuštění dělit na:

- Manuální
- Automatické
- Poloautomatické

2.2.1. Manuální testování

Dělení testů na manuální a automatické je jedním z nejzákladnějších dělení. Ať se použije jakýkoliv ty testu, tak je buď proveden manuálně, nebo automaticky. Manuální testování je zcela na zodpovědnosti testera, který na jeho základě hledá možně chyby softwaru. Toto testování může provádět dvěma způsoby:

- na základě definovaných testovacích scénářů

- tzv. free testy

Testuje-li tester na základě definovaných scénářů, není k tomu nutná přílišná znalost softwaru, za předpokladu, že jsou testovací scénáře napsány kvalitně. K takovéto činnosti je třeba pouze trpělivost a spořádanost. Trpělivost pro procházení kvant scénářů, které je třeba otestovat i když se liší třeba pouze minimálně. Spořádanost pro pravdivé zobrazení stavu po testech. Není-li tester při manuálním testování pečliví a to jak při čtení scénáře, tak při jeho vyhodnocení, tak se přehlédnutá chyba z manuálního testování může objevit až zákazníka.

Testy na základě definovaných scénářů jsou navrženy tak, aby odhalily co nejvíce chyb v rámci fungování softwaru. Na druhou stranu free testy zase odhalují slabiny softwaru. Pro tuto činnost musí být již tester splňovat celý set vlastností z kapitoly 2.1.4. Při těchto testech se zkouší odolnost při nečekaném chování. Tester se při free testech nedrží nalinkovaných scénářů, ale testuje spíše metodou "co když". Při takovém testování si pokládá stále dokola otázky typu: "Co se stane, když....". Tyto typy testů by se daly také přirovnat ke zběsilému klikání po obrazovce a zkoumání toho co se stane. Člověk musí mít velkou fantazii a vědět toho dost o softwaru, aby testování mohl řádně provést.

Manuální testování jako celek je značně náročně na čas, ale mnohem flexibilnější k vnějším faktorům, než je testování automatické. Na druhou stranou jeho časová náročnost může vést před vydáním softwaru k větší chybovosti. Tato chybovost je zapříčiněna omezenou kapacitou času testerů a blížícím se termínem vydání. Z každou opravou by se měla otestovat minimálně dotčená oblast, nejlépe by se však měl spustit celý proces znovu. Řešením takovýchto situací může být automatizování testů.

2.2.2. Automatické testy

Pro zrychlení procesu testování. Zatím co, pracovní doba testera je (většinou) 8 hodin, automatické testy mohou pracovat nepřetržitě. Na základě zadaných parametrů mohou projet jeden typ scénáře se všemi variantami, neunaví se, neudělají chybu. Pracují tak, jak jsou naprogramovány.

Na základě tohoto popisu by se dalo říct, že každá firma by měla investovat pouze do automatizace a nikoliv do práce testerů. Toto však není pravda. Ne každý proces se dá automatizovat, stejně tak se nedá automatizovat každý scénář.

S rostoucím kódem softwaru se zvyšuje počet testů a náklady na jejich údržbu. Pokud by firma investovala pouze to automatizace testů, stále by zde museli být test designéři, kteří by navrhli, jak testy se mají chovat, vývojáři, aby je naprogramovali, tester, který by řešil jejich chyby.

I při perfektně odladěných automatických testech je každé přidání kódu rizikem, že se celé testy musí znovu upravovat/ programovat. Spuštění takových testů vyžaduje značnou stabilitu testovacího prostředí.

Automatizace tím pádem vyplatí u procesů (softwaru), který se repetitivní, je třeba při něm testovat mnoho variant při stejném postupu nebo slouží k ověřování funkčností. Automatické testy mohou být využity i třeba pro přípravu dat je-li to výsledkem spuštěného softwaru. Dá se s nimi rychle udělat tzv. smoke test prostředí, kdy pokud tyto testy projdou, je prostředí stabilní pro testy.

Programátoři je často využívají ve formě UNIT testů, kdy pomocí takovýhle testů ověřují správnost vyvinutého kódu. UNIT test je napsaný na vždy na samostatně testovatelnou jednotku a je nezávislý. Dobře napsané testy se dají použít i pro rychlejší testování regrese, nebyla-li vývojem měněna.

2.2.3. Poloautomatické testy

Tento typ testů je kompromisem mezi manuálními a plně automatickými testy. Jsou koncipovány tak, že část testu je provedena manuálně a část automaticky. Nejedná se tedy o testy, kdy jsou data připravena automaty, ale může jít třeba o tzv. end to end (E2E) testy, které slouží pro otestování celého procesu, kdy část procesu je zautomatizována a část nikoliv. Tento typ testů se dá dále využít v případě rozšíření funkčností, pro nadstavbu funkčností nebo pro pokračování procesu, kde byla jeho první část již zautomatizována.

Poloautomatické testy stejně jako automatické testy by měli jít zopakovat manuálně. Jejich výstup by měl být sledován a zaznamenán a jejich funkčnost by měla být popsána testovacími scénáři stejně jako je to u manuálních testů. Jsou-li tyto skutečnosti splněny, lze pak snáze při pádu testu odhalit jeho příčinu, není-li pro danou sekci kódu napsané logování, které popisuje do zobrazené výjimky popis chyby.

Bohužel ne vždy se i přes napsané logování dá zjistit, kde test padá, a proto je důležité, aby zde byla možnost manuálního průchodu pro kontrolu a zjištění chyby.

2.2.4. Dělení testů

Kromě rozdělení testů na manuální, automatické a poloautomatické lze testy dělit i podle jejich funkčností na (14):

- Load and Performance Testing (výkonnostní testy)
- Installation Testing (instalační testy)
- Stress & Volume Testing
- Compatibility & Migration Testing (testy kompatibility a migrační testy)
- Data Conversion Testing (testy konverze dat)
- Security Testing (Application Security, Network Security, System Security)

Testy se mohou dále dělit na (6):

- **Integrační**
  - Incremental Integration Test
- **Systémové testy**
  - Recovery testy
  - Performance testy
  - Security testy
  - Stress test
  - Instalační testy
- **Akceptační testy**
  - Alpha test
  - Beta test
  - Acceptance test

Ať již vezmeme v úvahu jakékoliv rozdělení, tak by měl každý vývoj softwaru projít minimálně funkčními, výkonnostními, bezpečnostními, akceptačními, recovery a instalačními testy. Vývojář by měl na každý ucelený kousek napsat UNIT test, další testy by měly být vytvořeny na základě nalezených chyb. Takové testy zabrání jejich opakování. (18)

2.3. Atlassian JIRA

Atlassian JIRA je systém pro podporu vývojářských týmů během všech fází dodávky. Zahrnuje řízení a sledování úkolů a požadavků v projektu (task and project management),

nabízí možnost vytváření tzv. workflow a širokou nabídku rozšiřujících modulů. Díky této nabídce může každá společnost do značné míry zakoupený systém personalizovat dle svých potřeb.

JIRA neslouží pouze vývoji, ale i tester může do ní zaznamenávat nalezené chyby pomocí tzv. Bugů. Tyto Bugy se mohou lišit svojí konfigurací firmu od firmy, přesto by měly mít společné prvky, jako jsou:

- Název - krátké výstižné pojmenování chyby
- Popis - pole pro výstižný a podrobný popis
- Prostředí - pole pro označení prostředí, na kterém chyba nastala
- Verze - verze, na které se defekt vyskytl, nebo ve které verzi bude opraven

Kromě Bugů existují v systému JIRA i tikety, které slouží pro zaznamenání práce vývojáře či testera. Tikety mohou mít různé úrovně, kdy pod větším a komplikovaným tiketem může být jeho rozmělnění na menší části, které se mohou distribuovat více lidem.

Všechny tikety a Bugy, které nejsou momentálně nikým řešeny, jsou umístěny v Backlogu, který tak obsahuje přehled práce, kterou je třeba ještě vykonat.

2.4. SpiraTest

Stejně jako Atlassian JIRA je pro řízení vývoje, tak SpiraTest od společnosti Inflectra je nástrojem pro řízení testů. Na jednom prostředí jsou zastřešeny požadavky, testy, releasy (vydání nové verze systému), iterace, úlohy, chyby a incidenty.

Strukturovaná forma, podobná vnoření jednotlivých složek v systému Windows, slouží pro přehlednost vytvořených testovacích scénářů. Tyto scénáře pak mohou být sestaveny do jednotlivých iterací, které lze jednoduše filtrovat pro následnou exekuci. Výsledky spuštěných testů se mohou na základě této iterace vyhodnotit. Pro každé nové kolo testů, by měl být vytvořena nová iterace pro větší přehlednost výsledků.

Exekuce (spuštění) testu se dá ve SpiraTest pozastavit, nebo nastavit do příslušného stavu s propojením na nalezený problém.

2.5.Vývoj

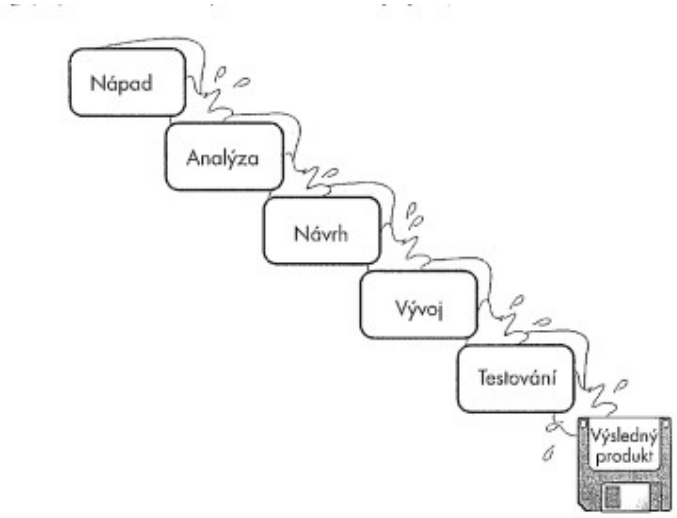
2.5.1. Model vodopádový vývoje

Tento model je velmi systematický. Všech jeho pět částí jde po sobě a není zde přílišný prostor pro úpravy či změny. U vodopádového vývoje se postupuje od nápadu, což je první podnět, přes analýzu, která zjistí realizovatelnost tohoto nápadu. Lze-li nápad realizovat

tak se vytváří třetí část vývoje a to je návrh. Návrh se předloží vývoji, který s ním pracuje až do samotného konce vývoje, kde je předán k testům a otestován.

Výhodou tohoto přístupu je naprosto jasná a detailní specifikace. Při samotném vývoji a testování není pochyb o žádném detailu, neboť vše bylo předem dohodnuto a přesně specifikováno. Výsledek bývá velmi kvalitní. Nevýhody z pohledu testování lze spatřovat ve skutečnosti, že testování probíhá pouze na konci a také v tom, že specifikace produktu je velmi náchylná ke vzniku chyb.

Obrázek 6 Model vodopádového vývoje



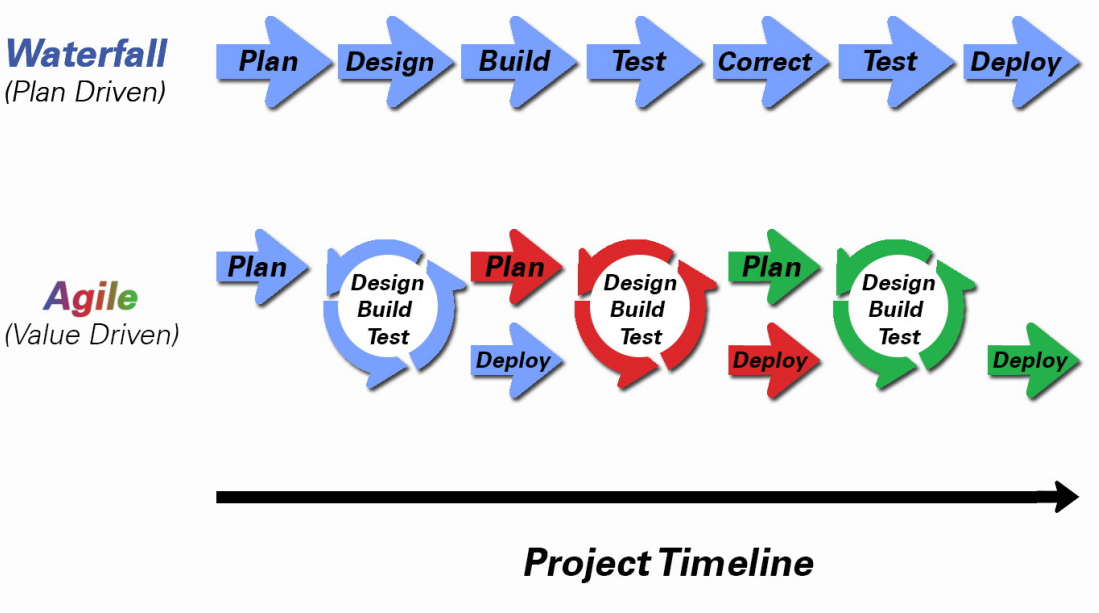
Zdroj: Testování softwaru, (11)

2.5.2. Agilní vývoj (agilní metodiky)

Dynamičtější způsob vývoje tvoří agilní metodiky, které jsou založeny na inkrementálním a interaktivním vývoji. Díky postupnému vývoji je možné rychleji reagovat na změny požadované zadavatelem. Větší interakce zároveň zaručuje, že klient, po dokončení vývoje, dostane přesně takový produkt, na jehož vývoji spolupracoval. (7)

Agilní postup se využíval již i dříve, jako ustálený výraz vznikl až v únoru 2001 po sepsání *Manifest agilního programování*, kde se definují přístupy k vývoji, nyní známém jako agilní programování (8)

Obrázek 7 Porovnání agilních metodik s vodopádovými



Zdroj: *Why PE And Agile Haven't Always Gone Together* (16)

2.5.3. Další možnosti vývojových metodik

Prototypování

Tento typ vývoje se zabývá vytvářením prototypů. Nedochází při něm k úplnému dovývoji softwaru. Není používán samostatně, ale spíše jako doplněk větších metodiky (tj. spirála, nebo RAD -Rapid application development).

Prototypování se snaží snížit nebezpeční rizik spojených s projektem, tím, že ho rozdělí na menší části, které tak umožňují snadnější provedení změn v průběhu procesu.

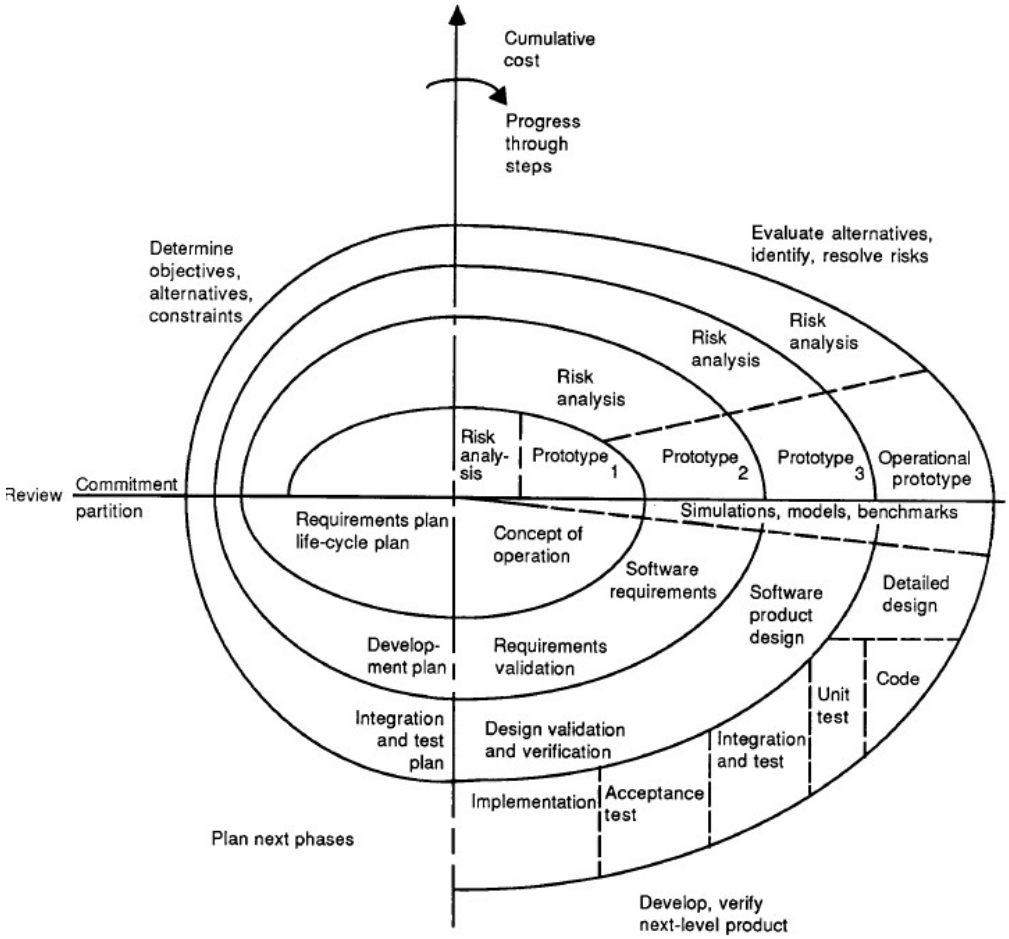
Spirálový přístup

V roce 1985 definoval Barry Boehm spirálový model softwaru (3), kdy tento model definuje 4 opakování cyklu složených vždy ze 4 kroků. Tyto kroky jsou:

- určení cílů, alternativ, omezení
- hodnocení alternativ
- vývoj a ověření další úrovně produktu
- plánování další fáze

Tyto cykly jsou zakončeny revizí, kdy pokud by byl tento přístup kombinovaný s prototypováním, tak by prototyp vznikl ve druhém kroku po analýze rizik.

Obrázek 8 Spirálový model



Zdroj: Wikiversity. Dostupné z: http://www.ics.uci.edu/~wscacchi/Software-Process/Images/Spiral-Model-Boehm-1987.gif (15)

Inkrementální přístup

Je založen na kombinaci sekvenční a iteračních metodik. Stejně jako u prototypového přístupu i zde je snaha omezit projektová rizika rozdělením projektu na menší části. Díky této segregaci je vývoj možné mnohem efektivněji řídit i ve vodopádovém modelu, kdy je model aplikován vždy pouze na vyvíjenou část.

Scrum

Používá se k řízení vývoje složitých produktů od začátku 90 let 19. století a je založen na teorii řízení empirických procesů. Scrum je: (12)

- jednoduchý
- srozumitelný
- extrémně obtížný pro dokonalé zvládnutí

2.6. Ostatní role v agilním vývoji

Scrum master

Náplní práce Scrum mastera je primárně pomáhat týmu dosáhnout jeho cílů. Aby toho dosáhl, tak za pomoci koučovacích principů podporuje tým a jednotlivce v jejich rozvoji, odstraňuje vzniklé překážky vývoje. Tím zajišťuje vyšší efektivitu, kvalitu a motivaci k práci.

Business vlastník

Je často nazýván také Product Owner a jak z překladu tohoto anglického slova vyplývá, je vlastníkem produktu. Jeho prací je vytváření zadání projektů na základě vizí společnosti. Jeho návrh by se měl opírat o potřeby zákazníků, zaměstnanců a přinášet firmě zisk nebo úsporu nákladů.

Business analytik

Vypracovává projekty na základě zadání od bussines vlastníků, kdy za pomoci business analýzy, tzn. sběr informací pro vytvoření a následnou správu analytické dokumentace, vytvoří koncept zadání, který zohledňuje systémy firmy a zkoumá dopady, které může mít tento projekt na jejich fungování.

2.7. Shrnutí kapitoly 2

Kapitola nabízí teoretické objasnění pojmů, které se dále vyskytují v diplomové práci. Je zde uvedena historie vzniku testování, jeho koordinace v rámci softwaru SpiraTest, řízení nalezených chyb v rámci softwaru JIRA. Kapitola obsahuje popisy rolí, které se účastní vývoje a vlastností, které by měl mít tester. Dále popisuje jednotlivé metodiky vývoje a typy testů.

3. Základní údaje

Jak jsem již v úvodu zmínila, pracuji pro jednu ze středně velkých firem zabývajících se finančními službami. Její nabídka služeb je oproti větším firmám omezena, přesto je i pro tyto služby potřeba spravovat komplexní soustavu vzájemně propojených systému na pozadí. Některé ze systémů jsou kritické a jiné nekritické. Při výpadku kritických systémů nelze provést žádnou operaci a tento výpadek omezí zákazníka. Nekritické jsou pak ty, které přímo chod společnosti neohrozí a klient se o nich často ani nedozví. Samozřejmostí je, že s rozšířením produktové řady se často vyvine úplně nový systém, kterému se následně musí určit daná kritičnost. Pokud všechny systémy fungují, jak mají, tak vývojáři systému a testeři odvedli svoji práci na výbornou. Bohužel, ať je sebelepší vývojář, či tester, jsou zde vždy situace, na které nepomyslí. Z tohoto důvodu se vytvářejí různé typy testů a to jak za vývojáře (např. integrační, unit testy) nebo pak celé sady testovacích scénářů, vytvořených testerem (test analytikem), které se snaží možnosti vzniku chyby v systému co nejvíce eliminovat.

3.1.Současný stav

Díky zrychlování životního stylu chce klient vědět, že i firma, která se mu stará o finance dokáže být dostatečně rychlá, pružná a adaptibilní, aby včas mohla reagovat na vývoj trhu. Jen taková firma dokáže klientovi přinést takovou míru upokojení jeho potřeb, aby i nadále zůstal jejím klientem. Proto společnost, ve které pracuji, přechází v současné době z modelu vodopádového vývoje více na model vývoje agilního. Tyto modely jsou značně protikladné, a proto se vytvářejí nové metodiky a upravuje se původní koncept procesů ve firmě. Veškeré metodiky a procesy, které byly nastaveny pro vodopádový vývoj, který nepřinášel tolik chtěnou rychlost, pružnost a adaptibilitu, jsou postupně měněny a vytvářeny pro agilní vývoj. Jelikož přepnout z jednoho typu vývoje na jiný není tak jednoduché, tak nějaký čas firma oběma způsoby.

Za mého nástupu byl pouze vývoj vodopádový, ale dostala jsem jedinečnou příležitost se zúčastnit počátků agilního vývoje a testování. Díky této příležitosti mohu porovnat tyto dva způsoby vývoje.

Při vodopádové metodice bylo jednoznačné zadání, do kterého se (nebylo-li to právně, systémově nebo koncepčně nutné) nedělaly žádné změny. Vývojový tým dostal poklady, které si objasnil se zadavatelem a analytikem, pak již jen ukazoval výsledky své práce. Oproti tomu je agilní vývoj více živý. Změny a myšlenky se do značné míry mohou dodávat během vývoje, což je pro mne jako testera mnohem větší výzva.

Při původním stylu vývoje byly testovací scénáře upravovány jen zřídka a většinou se jednalo o kosmetické úpravy validací vzhledu. Nový styl vývoje je náročnější na udržování přehledu změn. Tyto změny je potřeba do projektu zaznamenat, posoudit, zda jak moc ovlivní již vyvinuté funkčnosti a napsané scénáře. Navíc business analytiků není nazbyt a test analytik je zároveň tester. Momentálně je v našem týmu 5 vývojářů a 2 testeři, kteří dělají test analýzu, napíší testovací scénáře a následně je v rámci dokončení vývoje otestují. Po otestování, že vyvinutá funkčnost funguje dle zadání, předávají projekt regresní části testovacího týmu.

Zde vidím asi největší slabinu celého procesu. Jedna osoba problém zanalyzuje, popíše do testovacích scénářů a ty následně, společně s druhým testerem, otestuje. Ač je zde jistý dohled mentora, tak tento mentor má pod sebou více lidí. Musí udržovat znalosti z rozdílných úkolů, na kterých jeho svěřenci momentálně pracují a má-li pod sebou vícero testerů z agilních týmů, tak zde vzniká další riziko, že se k němu nedostanou všechny informace.

Ve firmě již existuje navrhnutá metodika vývoje, ale není zde úplně dotažena metodika testování, na které se nicméně usilovně pracuje. Většina testerů tedy test analýzy, své poznatky a poznámky k projektu udržují v poznámkových souborech či na papíře a nikoliv na nějakém místě, kde by k nim měl přístup i druhý tester z týmu, což může značně ztížit práci na projektu např. v případě nemoci. Udržování přehledu domluvených změn a úprav je při agilním vývoji nezbytné, jelikož tester (test analytik) by měl vždy mít dostupné nejnovější informace k projektu a na jejich základně dělat test analýzu, psát či upravovat testovací scénáře.

Pro agilní vývoj je více než u vývoje vodopádového nutnost upravovat jednotlivé kroky napsaných testovacích scénářů. Nevýhodou je, že se jednotlivé kroky scénáře nedají jednoduše hromadně upravovat, avšak zatím jsem se nesetkala s nástrojem, který by toto

umožňoval. Drobnosti (i velké věci), které se během vývoje v agilní metodice mění, nelze tak snadno zapracovat do již vytvořených scénářů a takovýto nástroj značně usnadnil práci, jak propagovat tyto změny jednoduše do již vytvořených testovacích scénářů. V současnosti tak musí tester vždy projít jeden scénář za druhým a každý ze kroků upravit ručně.

Tím vzniká velká časová náročnost na tyto úpravy, zvlášť jsou-li již scénáře napsány pro různé typy distribučních kanálu (např. jako je průchod službou s klientem přes call centrum, na pobočce, nebo přes webové rozhraní), kdy se tyto scénáře často liší pouze v počátečních krocích. Navíc úpravy v projektu nejsou při agilním vývoji výjimečné.

Často se tak stane, že vznikají velké nesrovnalosti mezi původní myšlenkou a současným stavem. Tyto nesrovnalosti mohou být například nedomyšlené věci v zadání, nebo některé popsané problémy mohou být již vyřešeny jiným realizovaným projektem. Naštěstí jsou systémy řádně popsány analytiky v přehledné architektuře a ke koordinaci zadání slouží JIRA.

Občas ale i zde mohou vzniknout duplicitní tikety. Každý tým má k sobě ekvivalentně vytvořený projekt, Úkoly pro daný tým jsou na schůzkách rozřazeny do těchto projektů, kde si pak členové týmu mohou zakládat, po domluvě se svým vedoucím, své tikety, které potřebují pro řešení daného úkolu. Pro tým je tento způsob přehlednější, avšak se může stát, že se díky tomu objeví, již zmíněný duplicitní tiket. Tomuto stavu se snaží jak projektový manažer, tak jednotlivé týmy zabránit tím, že jednotlivé tikety propojují mezi sebou. Některé týmy však pracují nezávisle, například aplikační podpora, která řeší všechny chyby, které se dostanou až na produkci, může vyřešením chyby částečně vyřešit i některý z vyvíjených tiketů. Tyto nesrovnalosti se pak následně snaží rozklíčovat vývojář spolu s testerem, bohužel však někdy nezávisle na sobě.

Jedním z mála společných prvků, které má současný agilní vývoj s původním vodopádovým vývojem je začátek tvorby projektu, kdy na začátku tvorby projektu je uspořádaná schůzka s business vlastníkem (product owner), analytikem a vývojovým týmem a to včetně testera a projekt je představen. Před schůzkou si vývojový tým řádně pročte zadání a sepíše případné dotazy na zadavatele.

Dalším ze společných prvků jsou pak schůzky, kde jsou jednotlivé části vyvinutých funkčností projektu představeny zadavateli. Zde se může bussines vlastník, ale i budoucí uživatelé této funkčnosti/systému vyjádřit k dosavadnímu vývoji. Toto vyjádření je pak již rozdílné, zatím, co u vodopádového vývoje se vyjádření týkalo pouze toho, jestli se vývojový tým striktně držel zadání, či nikoliv, u agilního vývoje je tento prostor spíš otevřen připomínkách, nápadům na optimalizaci a vylepšení. Ne každý nápad je realizován, ale je zde větší volnost pro zadavatele a budoucího vlastníka k customizaci výsledného projektu.

Abych to shrnula, současný stav má několik málo nedostatků, které jsou ale většinou způsobeny růstem firmy a její adaptací na tento růst. Díky tomuto přechodu na nový typ vývoje zde vzniká prostor pro vylepšení pracovních postupu, kterým se intenzivně věnují zaměstnanci, mající to v popisu práce. Jelikož není tento postup ještě definitivně popsán, ráda bych touto prací přispěla k jeho zhotovení na základě svých dosavadních zkušeností.

3.2.Vývoj a testování

Vývoj a testování patří neodmyslitelně k sobě, avšak nelze příliš dobře provádět tyto činnosti na stejném místě (prostředí). Díky rozvoji firmy rostou i nároky pro údržbu těchto prostředí. V současné době bohužel není dostatek místa, aby každý tým měl jak vývojové, tak testovací prostředí. Jednotlivé týmy tak pracují na svých verzích systému a tam jsou tyto verze otestovány. Jednotlivé větve vývoje se pak sejdou na společném prostředí, kde se testují následně i regresní dopady.

Tím zde vzniká jisté omezení jak pro testery, tak pro vývojáře, kteří si nemohou s prostředím "dělat, co chtějí", ale musejí brát ohledy, kromě jiných vývojářů také na testery, kteří na tomto prostředí testují jednotlivé části projektu. U agilního vývoje je to ještě komplikovanější tím, že zde nejsou pevně určeny hranice, kdy již zadavatel nemůže měnit a upravovat zadání. Tento nedostatek, v podobě chybějícího testovacího prostředí, by měl být v blízké době odstraněn.

3.3.Struktura test týmu

Jelikož je k testování i ve středně velké firmě je třeba mnoho lidí, celá struktura funkčního testování rozdělena na čtyři důležité členy týmu. Celý testovací tým vede jeden vedoucí,

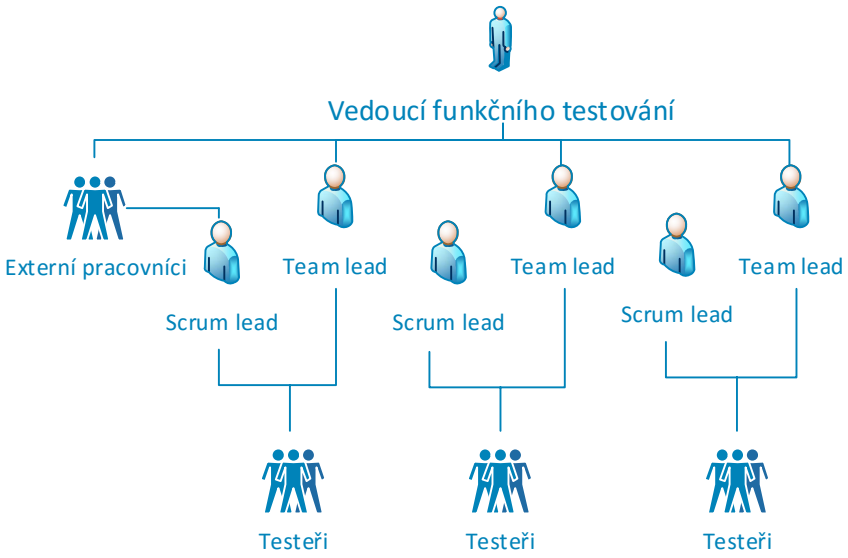
pod kterého spadají všichni externí a interní testeři. Ten zaštiťuje veškeré testování systému, kromě výkonnostních (performance) a bezpečnostních (security) testů.

Pod tímto vedoucím je pak dvoudimenzní řízení samotných testerů. První dimenzi zajišťuje liniové vedení a druhou pak projektové vedení (scrum).

Liniové vedení zajišťují tzv. test leadi, kteří vykonávají personální operativu ohledně testerů a zajišťují odborné vedení. Dále kontrolují kvalitu vykonané práce, zajišťují další vzdělání a zvyšování znalostí v rámci systému.

Projektové vedení zase na druhou stranu zajišťuje koordinaci testů v rámci projektu a provádí případné dočasné navýšení kapacit v případě rozsáhlejší oblasti testování.

Obrázek 9 Hierarchické rozdělení testování



Zdroj: vlastní tvorba

Na výše uvedeném schématu je znázorněno hlavní hierarchické rozdělení v rámci firmy. Kromě test leadů, jsou zde dále ještě koordinátoři nových a regresních funkčností. Ti mají primárně na starost společná prostředí, jejich prvotní otestování po nasazení a informování testerů o problémech, které mohou nastávat. Na tyto koordinátory a vedoucího funkčního testování se obrací projektové řízení (tzv. scrum master) v případě, že kapacita přidělených testerů nestačí pro otestování nové funkčnosti. S regresním koordinátorem je také probírán dopad do regrese.

3.4. Popis rolí a jejich funkčnosti

Koordinátor za regresní testy

Primární náplní jeho práce je revize regresních scénářů, zapracovávání chyb po nasazení vyvinutých funkčností do regresních testů. Mezi jeho další činnosti patří rozdělování práce mezi testery, sestavování skupiny scénářů pro testování, reportování výsledků z testování a koordinace zpracování nových scénářů do regresních. Dále také případná urgence opravy chyb a jejich retestů.

Koordinátor za testy nových funkčností

Náplní práce koordinátora za nové funkčnosti (NF) je správa plánu přípravy testů, reporty z těchto testů, rozdělení testů NF, koordinace testů s případnými třetími stranami, průběžná kontrola zadaných chyb a urgence dlouho neřešených či neretestovaných chyb. Dále úzce spolupracuje s koordinátorem za regresní funkčnosti ohledně kapacit testerů, kteří mohou být přiděleni na výpomoc.

Vedoucí oddělení funkčního testování - liniový vedoucí

Účastní se výběrových řízení na nové testery a s tím související případné dohody s dodavateli externích testerů. Schvaluje výkazy podřízeným pracovníků na živnostenský list. Vymýšlí spolu s test leady plány rozvoje pracovníků. Schvaluje dovolené test leadům. Má vynikající přehled o jednotlivých systémech a na základě těchto znalostí ladí vnitřní procesy pro testování.

Nedílnou součásti jeho práce je podávání zpráv o průběhu testování vedení firmy a zpracování odhadů délky trvání testování. Sledování stavu testů a rozšiřování získaných vědomostí, které jsou nezbytné pro testování, na test leady a následně na testery. Definuje i zadání pro reporty od test koordinátorů.

Test leadi

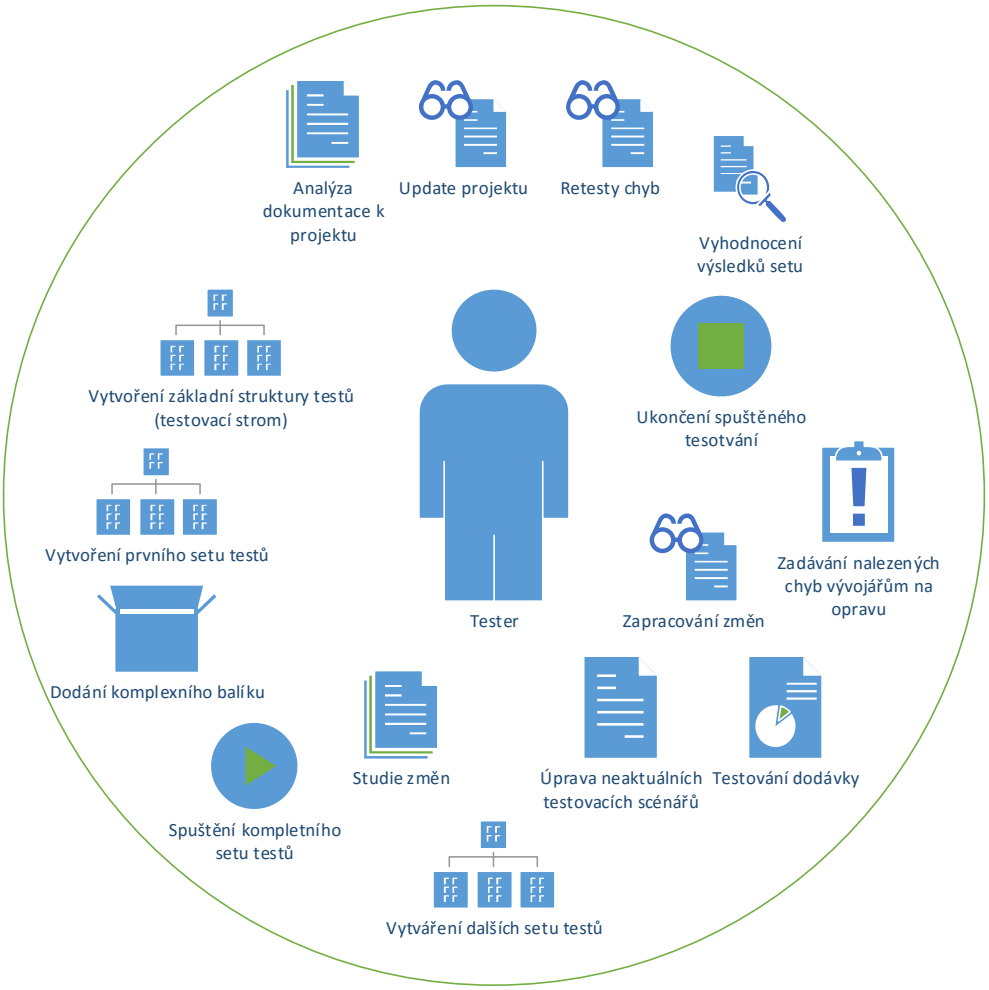
Test lead spravuje a dohlíží na docházky, schvaluje dovolené. Stanovuje spolu s testerem kvartální cíle. Mentoruje testera a pomáhá mu ve zlepšování jeho znalostí a dovedností. Zajišťuje u vedoucího požadavky na školení pro testery a přebírá zpracované projekty z nových funkčností. Dohlíží na práci testera a případně opravuje nalezené chyby.

Popis práce testerů

Práce testera je velice rozmanitá, od studie dokumentace, přes vytvoření analýzy až po sestavení testovacích scénářů a otestování projektu. Hlavní činností je hlavně zmíněná analýza projektu a podle ní vytvoření testovacích scénářů, které co nejlépe otestují vyvíjenou oblast.

Další významnou oblastí práce testera je pak samotné testování a reportování chyb, kdy je třeba co nejpřesněji popsat danou chybu, aby mohla být odstraněna. Po jejím odstranění ověřit, že je opravdu odstraněna. Hlídat si přidělené tikety v systému JIRA a zlepšovat znalosti v rámci kvartálních cílů.

Obrázek 10 Činnosti testera

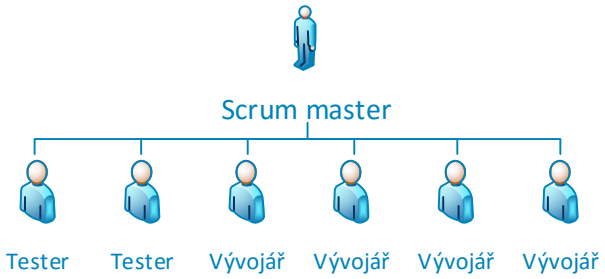


Zdroj: vlastní tvorba

3.5. Složení Scrum týmu

Týmy mají různé složení, podle složitosti projektů, které dostávají přiděleny. Tým, ve kterém pracuji já, se momentálně skládá ze čtyř vývojářů, dvou testerů, jednoho scrum mastera a business analytik je přidělený na projekt. Tým většinou pracuje na jednom zadaném projektu, ale může se stát, je-li projekt menšího typu, že se pak pracuje na dvou paralelních projektech zároveň.

Obrázek 11 Hierarchické rozdělení testování



Zdroj: vlastní tvorba

3.6. Shrnutí kapitoly 3

Kapitola popisuje současný stav ve společnosti, ve které pracuji. Jsou zde z praktického hlediska popsány role a jejich povinnosti v rámci firmy. Kromě rolí je zde uveden přechod firmy z vodopádového modelu vývoje na model agilní a dopady tohoto přechodu.

4. Fáze projektu

V této kapitole se budu zabývat současným stavem řešení jednotlivých projektů, které se v rámci týmu vyvíjely. Popíši jednotlivé fáze od představení projektu, přes jeho analýzu, psaní test casů až po samotné testování a předání nasazovacímu týmu. Cílem této kapitole není navrhnout lepší řešení, ale co nejpřesněji, jak je to pro zachování citlivých dat možné, popsat stav, který je podkladem pro samostatné vylepšení.

4.1. Zadání projektu

Zadání projektu začíná schůzí business vlastníků, kde jsou zvoleny projekty, které jsou momentálně pro firmu nejpřínosnější. Tyto projekty se nacení a dle tohoto nacenění jsou pak na prioritizační schůzce rozděleny do týmů dle jejich časových kapacit. V úvahu se bere počet pracovních dní vývojářů (tzv. man day - MD), které má daný tým k dispozici.

Obrázek 12 Rozdělení projektů do týmů

Zdroj: vlastní tvorba

Po prioritizační schůzce se pak následně vytvoří backlog týmu, kde jsou uvedeny jednotlivé projekty, které byly na této schůzce přiřazeny týmu. Backlog je uspořádaný v hierarchii, v jaké se jednotlivé projekty budou vyvíjet. Toto pořadí se může měnit dle okolností, jako je např. být rychlejší/pomalejší vývoj oproti odhadu, čekání na dodávku od jiného týmu, nedomyšlenost projektu.

Když má scrum master uspořádaný backlog, je jasné, který projekt začne tým vyvíjet jako první, je svolána schůzka se zadavateli. Před touto schůzkou tým studuje zadání projektu a

to jak v JIRA, kde je základní popis (někdy i bussines popis), tak také na vnitropodnikové wikipedii, kde bývá rozsáhlejší bussines analýza.

Na schůzce se zadavateli je následně bussines vlastníkem představen základní koncept projektu - tzn. proč je tento projekt třeba, jaké problémy řeší, jaký přínos bude mít pro bussines, kdo jej bude využívat atd. Analytik upřesní podrobněji, jak je daný projekt zakomponován do systémů firmy, kde má dopad a jaké nástroje je třeba využít při implementaci.

Schůze končí vytvořením předběžných odhadů za vývojový tým, které se mohou od původních odhadů lišit z několika důvodů, např. z důvodu větší složitosti, než bylo v předchozím zadání předpokládáno.

Obrázek 13 Upřesnění zadání

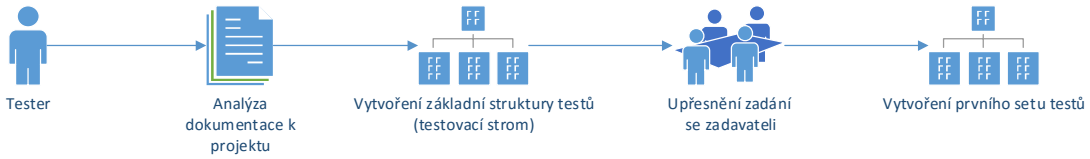
Zdroj: vlastní tvorba

4.2.Analýza projektu

Po schůzce se zadavateli jsou vytvořeny vývojové tikety do JIRA, kde pod zastřešující Story jsou vytvořeny tikety pro jednotlivé oblasti vývoje. Jelikož není test analytik, tak tester začíná analyzovat dokumentaci projektu a to nejen na základě dokumentace, ale také na základně poznatků se schůzky a vývojových tiketů.

Během analýzy tester vytvoří základní strukturu testů a občas přijde i na nedomyšlené části zadání a proto je třeba další schůze se zadavateli k upřesnění zadání. Je-li zadání dostatečně upřesněno, začíná tester psát první set testů.

Obrázek 14 Test analýza

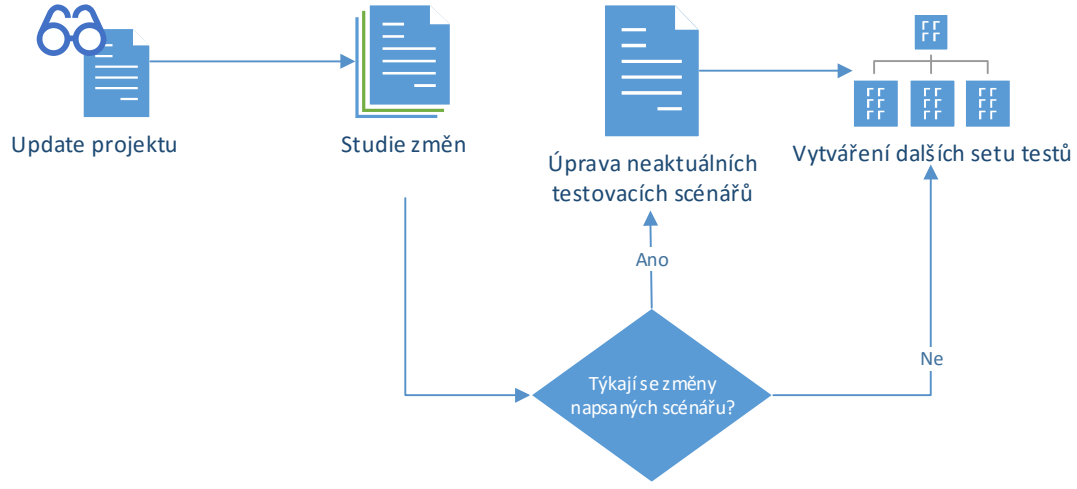


Zdroj: vlastní tvorba

Podle velikosti projektu jsou tyto testy psány buď pouze na základě dokumentace, nebo je-li již alespoň část projektu vyvinuta, tak na základě vyvinutých částí v porovnání s dokumentací, kdy zároveň je již vyvinutá část testována tzv. free testy.

Při vývoji se následně přijde na další možné komplikace - nemožnost vývoje komponenty, nesrovnalost současného stavu s popisem zadání, nedomyšlení možného rozbití jiných funkčností implementací, přílišná pracnost vyvinutí nepodstatné části projektu atd. Znovu se konzultují poznatky s analytiky a je-li zásah do projektu příliš velký, tak také i s bussines vlastníkem. Po domluvě s vývojovým týmem je následně proveden update projektu, kde jsou zaneseny domluvené změny.

Obrázek 15 Update projektu



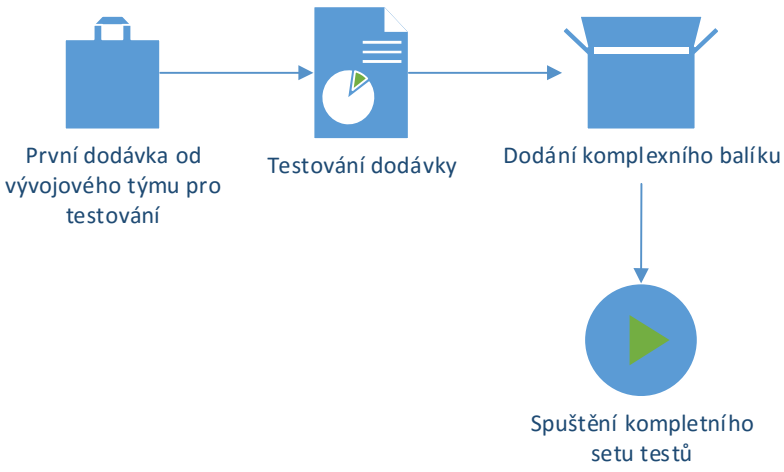
Zdroj: vlastní tvorba

Aktualizace projektu znamená pro testera studii změn a rozhodnutí, zda se dané změny týkají již vytvořených scénářů či nikoliv. Pokud ano, tester upravuje neaktuální scénáře a pokud ne, pokračuje ve vytvářené dalších test setů.

4.3.Testování projektu

Po dopsání všech testovacích scénářů, které vyplynuly ze zadání a vývojových tiketů, dodává vývojový tým první dodávku vyvinuté funkčnosti, tato dodávka je první v rámci vývojových tiketů otestována, dle postupu testování uvedeném v daných tiketech. Pokud dodávka projde těmito testy, jsou následně spuštěny kompletní sety testů, týkající se dané funkčnosti.

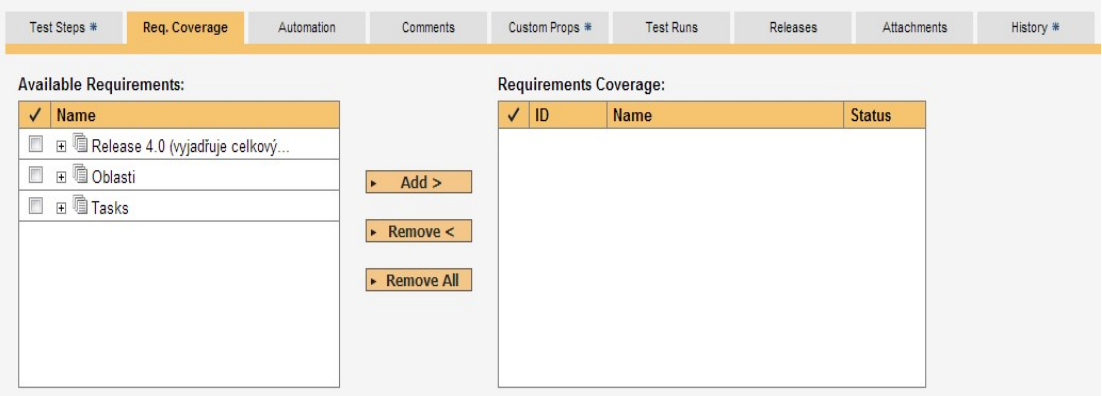
Obrázek 16 Dodávka projektu



Zdroj: vlastní tvorba

Testy jsou spuštěny (tzv. „execuovány“) ve SpiraTest v jednotlivých iteracích. Které odpovídají dodávce projektu. Nalezené chyby jsou zaznamenány do Test result a je vytvořený Bug v týmovém projektu JIRA.

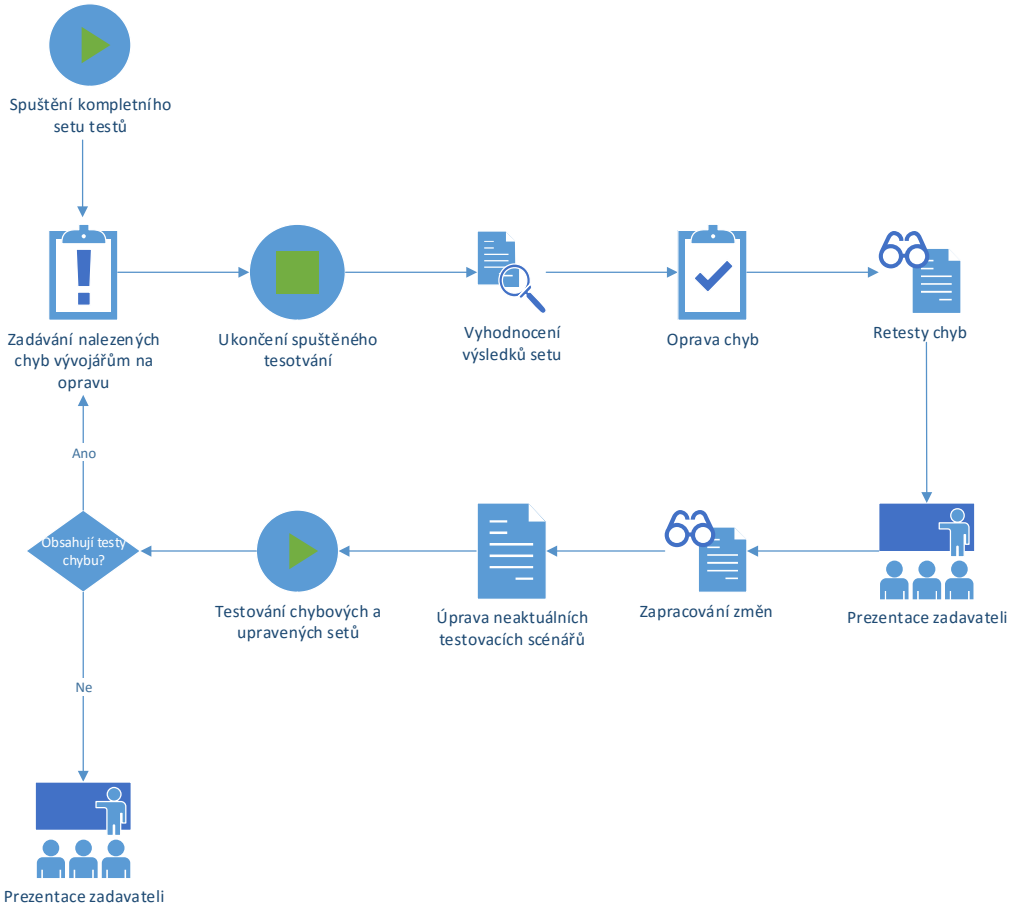
Obrázek 17 Přidání iterace na test ve SpiraTest



Zdroj: Print screen z programu SpiraTest

Po opravě bugu vývojářem je následně Bug otestován dle popisu, je-li opraven, jsou retestovány i všechny testy navázané na tento Bug.

Obrázek 18 Testování projektu



Zdroj: vlastní tvorba

Kromě chyb se při testování objeví nesrovnalosti testovacích scénářů s naimplementovanou skutečností. Tyto nesrovnalosti jsou ověřeny dle dokumentace, s vývojáři a i s analytikem projektu. Jedná-li se o špatný testovací scénář, je upraven a následně testován ve své upravené formě.

Pokud testy prošly alespoň na 80% je svoláno DEMO pro prezentaci zadavateli. Na této schůzce je prezentován projekt nebo jeho část, která je již vyvinuta (tento případ je pro rozsáhlejší projekty). Zadavatel se vyjádří k dodávce, zda je naimplementována dle původního zadání, resp. podle posledního platného zadání.

Případně dodá připomínky k implementaci, které jsou následně posuzovány dle několika faktorů:

- Požadavek byl v zadání
- Požadavek nebyl v zadání
    - Jedná se o malou úpravu max. 0,5 - 1 MD
    - Jedná se o větší úpravu do 2 MD
    - Jedná se o velkou úpravu nad 2 MD
- Změna původního návrhu

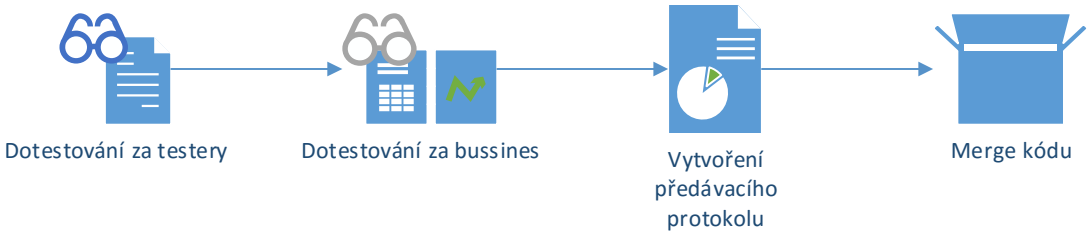
Pokud byl požadavek v zadání a zadavatel trvá na jeho implementaci, dodělává vývojový tým tento požadavek. Nebyl-li požadavek v zadání, tak zde velmi rozhoduje doba implementace vývoje. Většinou platí, že do 1 MD jsou požadavky v rámci agilního vývoje dodány. Nad 2 MD již pak musí vzniknout doplňující tiket a musí být schválen vedením vývoje, jelikož je pak o tento čas zdržen vývoj dalšího projektu.

Po DEMU je pak dána možnost businessu k otestování dané funkčnosti na vývojové prostředí, kde byla funkčnost testována i testerem. Tester zatím zapracovává případné připomínky ze schůzky se zadavateli do testovacích scénářů.

4.4. Doručení projektu

Po otestování dodaného kódu jak za testy, tak za bussines přechází projekt do své závěrečné fáze, kdy je vytvořen předávací protokol pro nasazení a kód je zakomponován (tzv. "zmergován") do hlavního kódu, který se následně nasazuje na stabilní testovací prostředí. Zde se následně podle času testerů daného týmu testuje další iterace na sloučeném kódu nebo se toto testování přenechává již regresnímu týmu, který zapracovává testy do regresních scénářů.

Obrázek 19 Předání projektu



Zdroj: vlastní tvorba

4.5. Shrnutí kapitoly 4

V kapitole 4 je popsán současný stav vypracování projektu od zadání, vytvořeném na základě myšlenky business vlastníka a zpracované business analytikem, až po předání vyhotoveného projektu zpět zadavateli. Každou jednotlivou fázi doplňují přehledné UML diagramy, které vizuálně doplňují napsaný text.

5. Návrh řešení

V kapitole 3 a kapitole 4 je popsán, co nejpřesněji současný stav, kde vidím pár míst, kde se může proces zefektivnit a tím pádem lépe odpovídat růstu nabízeného portfolia a spletitosti systému. V této kapitole bych se ráda navrhla některá zlepšení a to nejen dosavadního testování, ale i složení týmů a postupu při vývoji. I když se nedá navrhnout pouze jedno řešení, protože jak týmy, tak pracovníci v nich jsou značně dynamickou jednotkou. Můj návrh naznačí možnosti, které sice budou ze začátku nevýnosné pro firmu, ale delším časovém horizontu se jí tato snaha několikanásobně vrátí.

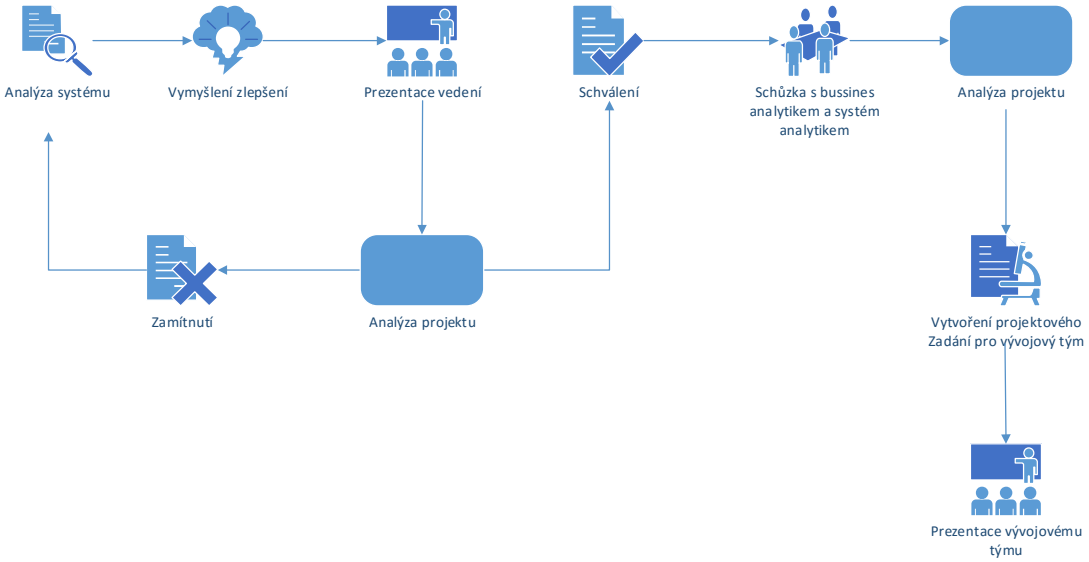
5.1. Před vývojová fáze

I když se může zdát, že tato fáze by neměla patřit do této práce, je nezbytně nutná pro proces a od kvality zadání se odvíjí rychlost jeho implementace. Nevznikaní následná nedorozumění při vývoji. Zadání by měl bussines věnovat větší pozornost před prioritizační schůzkou. Čas by neměl být vyhrazen pouze novým nápadům, ale i revizi starých projektů, které nebyly ještě realizovány.

Nové nápady na vylepšený systém pracují s aktuálními možnostmi a technologiemi, které jsou dostupné, avšak starší nápady mohou být již částečně realizovány, nebo již být naprosto zbytečnými díky jiným řešením, které existují. Postup vytvoření nového projektu by měl být následující:

- Analýza současných implementací
- Vytvoření návrhu na zlepšení
- Prezentace tohoto nápadu vedení
- Dle rozhodnutí vedení následně opravit nápad nebo schůzka se business analytiky, kteří dále mohou navrhnout zakomponování projektu do současné architektury.
- Na základě požadavků vývoje, je vybrán/ sestaven nejvhodnější tým pro jeho vývoj.
- Vývojový tým odhadne náklady na vývoj, případně řekne, co z projektu je možné vyvinout za stanovený odhad.

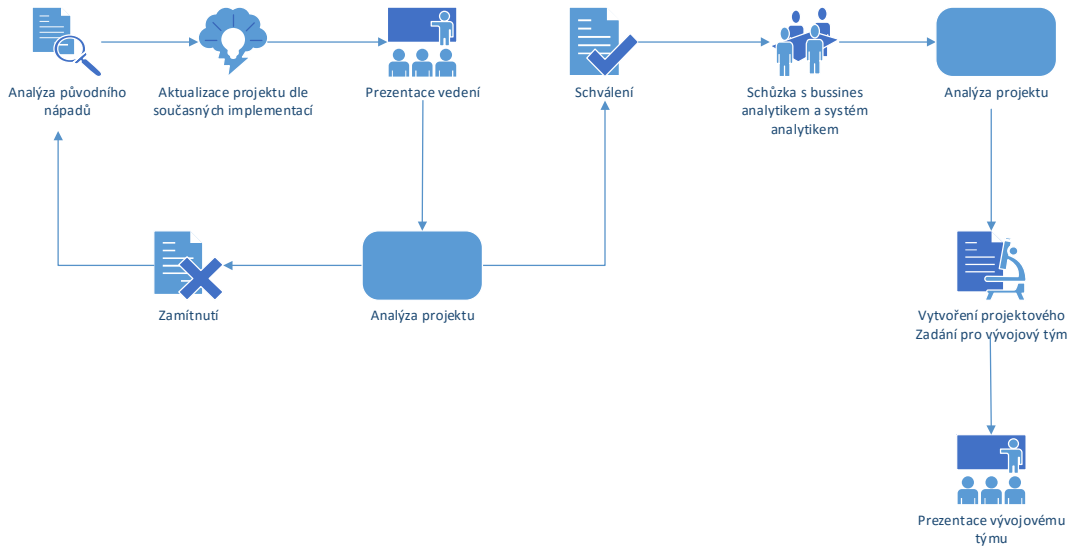
Obrázek 20 Proces vytvoření nového projektu



Zdroj: vlastní tvorba

Při postupu aktualizace staršího projektu je postup trošku složitější, jelikož si zadavatel musí zjistit řádně aktuální stav systémů. Dohledat, jestli části projektu nebyly implementovány v minulosti (např. jako hotfix), zda funkčnost má ještě smysl vyvíjet a není třeba naimplementované již lepší řešení. Je to rozhodně mnohem více práce na začátku, ale ušetří to čas při vývoji, kdy vývojový tým již tolik nezkoumá, zda neexistuje dané řešení, což se u pár let starých projektů může stát. Navíc tyto starší projekty nemusí počítat z nově vyvinutými systémy, které mohou zastávat části navrhnutého řešení.

Obrázek 21 Proces aktualizace staršího projektu



Zdroj: vlastní tvorba

Pokud se k vývojovému týmu dostane jednoznačné, srozumitelné a promyšlené zadání projektu má pak tento tým mnohem více času na vývoj a případné úpravy při aktualizaci zadání, než když je zadání narychlo schválené, protože se na něm mělo pracovat již např. před měsícem.

5.2.Vývojový tým

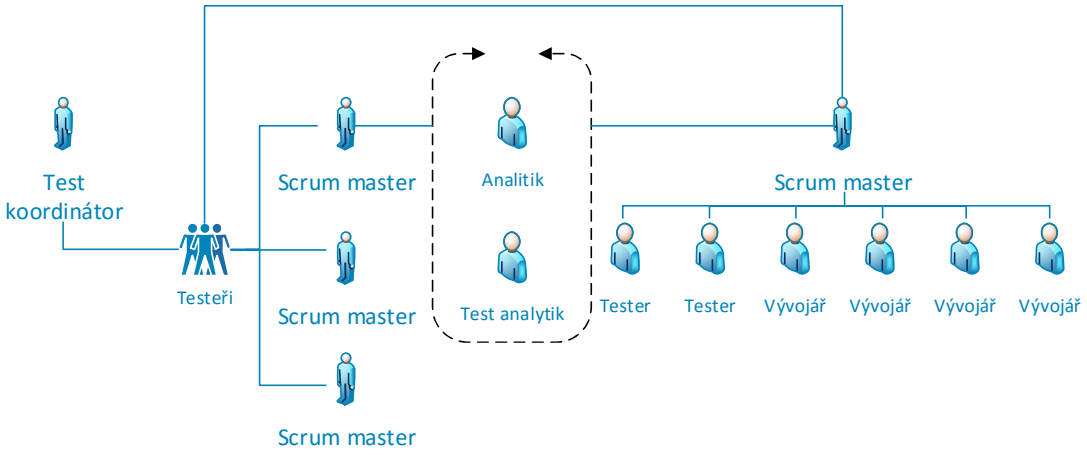
Je-li dodáno plnohodnotné zadání, je třeba sestavit tým, kterému by mělo toto zadání připadnout. Tým složený z pevného základu a proměnlivých členů. Vedení týmu a koordinaci práce, včetně domluvy případné výpomoci má na starosti scrum master. Základními členy dále jsou vývojáři a testeři dle velikosti projektu a zaměření projektu. Proměnlivou složku tvoří analytici, a to jak business, tak test analytik. Tito analytici jsou sdíleni maximálně mezi dvěma týmy.

Business a test analytik se věnují rozboru zadání a řešení nesrovnalostí v rámci projektu. Zapracovávají změny a úpravy zadání do popisu architektury a do testovacích scénářů, které udržují aktuální. Zároveň odhadují dopady případných změn v zadání na náročnost vývoje a dopad do testovacích scénářů.

Poslední proměnlivou složkou jsou testeři, kteří vypomáhají s rozsáhlejším testováním, ale jejich primární práci je např. regresní testování a zapracování testů do regrese. Tyto testery řídí primárně test koordinátor, který jim dává oblasti k zpracování/otestování. V tomto rozdělení je tým schopen efektivně řešit rozpory v zadání a případně další požadavky zadavatele.

Analytici mají komplexní přehled o projektu a jsou mnohem lépe schopni určit dopady těchto požadavků na vývoj. U menších projektů, čímž se rozumí do 10 MD, není tak důležitá funkce test analytika, jelikož i samotný tester musí být částečně analytik a vědět o projektu dostatečné informace, aby pomohl s psaním testovacích scénářů a při testování nenásledoval jen slepě napsané scénáře. Neměla by však vznikat situace, kdy tester napíše scénář a sám jej otestuje.

Obrázek 22 Složení vývojového týmu



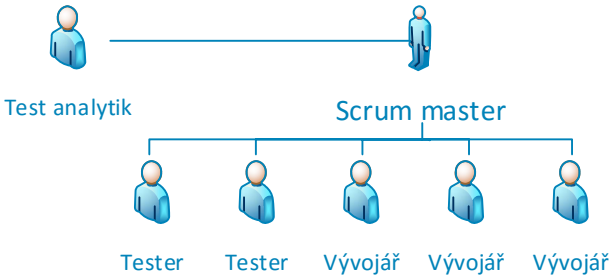
Zdroj: vlastní tvorba

Po napsání a zpracování dokumentace do podoby testovacích scénářů a jejich verifikace, nastupuje tým automatického testování. Tento tým spravuje a udržuje automatické testy, které mohou využívat všichni testeři pro zrychlení své práce (např. přípravu dat z podmínek testů). Náplní práce tohoto týmu je automatizace testů, které mohou sloužit pro přípravu dat, dále se zaměřuje na údržbu testů a aktualizaci kódů pro současný systém. Testy rozděluje dle funkčností:

- testy k ověření stability prostředí

- testy k základní přípravě dat (např. založení zákazníka, založení produktu dle parametrů)
- automatizace/ částečná automatizace náročných úkonů (např. založení zákazníka s převodem hypotéky)
- automatizace/ částečná automatizace nových funkčností

Obrázek 23 Složení týmu pro automatizaci



Zdroj: vlastní tvorba

5.3. Metodika testování

Je-li správně složen tým pro vývoj projektu, zbývá správně využít zdroje pro testování. Firma již nyní má částečně automatizované test. Vývojáři při svém vývoji píší UNIT testy a při spojování verzí je testována funkčnost celku integračními testy. Dále se využívají zátěžové testy, volné testování funkčností a regresní testování pro ověření integrity sytému.

Stejně jako narůstá složitost a délka zdrojového kódu jednotlivých aplikací, narůstá s ní i komplexnost testů a počet testovacích scénářů. Při testování a odhalování chyb je třeba postupovat dle firemní metodiky. Tato metodika jasně dává instrukce, jak se v daných situacích zachovat. Návod na formu (šablonu), jak psát testovací scénáře, jak zadávat objevené chyby do tiketovacího systému a jak řešit opravené chyby.

Metodika však nenavrhuje zlepšení a efektivitu práce, pouze poskytuje manuál pro jednodušší sjednocení práce různých typů lidí. Mnou navrhovaná metodika pro zlepšení efektivnosti práce je popsána v následujícím textu.

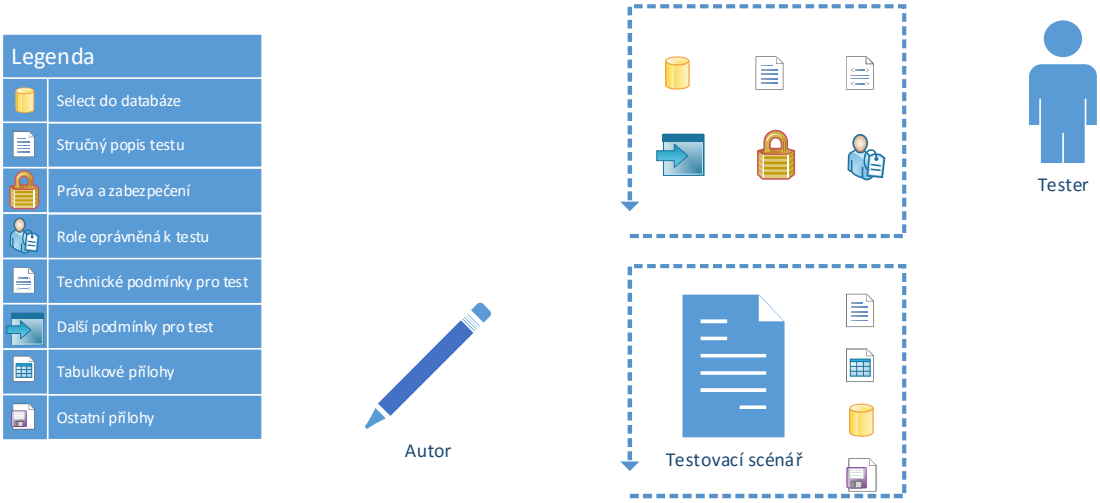
5.3.1. Psaní testovacích scénářů

Každý napsaný scénář obsahuje v popisu jednu, až dvě věty primárního popisu scénáře tzn., pokud se scénář např. zabývá založením klienta v bance, pak zde v úvodu je popsáno

o jakého klienta se jedná a co nám pomáhá tento scénář ověřit. Popis dále obsahuje podmínky vstupu testu a to včetně možných přístupů, práv a znalostí systémů, které jsou k testu potřeba. Z popisu by mělo být testerovi jasné, jaký je účel testu, jaké systémy, práva a podmínky jsou třeba pro testování a zda je kvalifikován takovýto test provést.

Jsou-li pro test vyžadována další data nebo odborné znalosti, jsou tato data umístěny do přílohy testu nebo test obsahuje odkaz pro získání těchto dat/znalostí. Jedná-li se o test, který může provádět pouze specifická osoba z firmy, je tato osoba uvedena jako tester nebo je v popisu napsána pozice takovéhoto zaměstnance.

Obrázek 24 Podmínky a podklady pro testera sepsané autorem scénáře



Zdroj: vlastní tvorba

Kroky jednotlivých scénářů by pak měli obsahovat vždy popis situace a její očekávaný výsledek, případně podkladová data nebo pomocné údaje (např. select do databáze, odkaz na získání podkladů pro provolání webové služby pomocí SOAP nebo vzorec pro pomocný výpočet).

Popis jednotlivých kroků je vždy srozumitelný. Je zde uvedeno, v jakém systému se pracuje, za jakou roli se testuje, co se přesně má udělat. Očekávaný výsledek je pak popsán

co nejpřesněji. Pokud možno nejsou výsledkem možnosti jako např. OK nebo zkontrolováno a to ani u validačních testů.

Je-li některá z částí testu společná po více testů, může být napsána jako samostatný test, který je pak odkázán do těchto testů. Jedná se většinou o opakující se činnosti typu zasílání zpráv v jistém formátu či jiné úkony, které nejsou ovlivněny typem scénáře, který je testován.

Kroky dělíme na co nejmenší fragmenty, aby bylo možno nejpřesněji zkontrolovat funkčnosti (případně zkontrolovat validitu např. testované obrazovky). V příliš dlouhých popisech kroků se lze lehce ztratit.

Výsledkem popisu kroku je pak očekávaná akce nebo očekávaný obraz na monitoru, který je popsán. I když se jedná pouze o popsání zobrazení, tak i zde by měla být, jako popis výsledku, alespoň jedna věta, která potvrzuje tento výsledek (např. Zkontrolováno, tlačítko pro odeslání formuláře se nachází v pravém dolním rohu.)

Testy jsou psány, co nejpřesněji, zároveň ale i co nejobecněji. I když si toto tvrzení zdánlivě odporuje, lze ho na psaní testů aplikovat. Toto pravidlo je důležité z důvodu měnění textace během vývoje. U validačních textů se, kromě struktury obrazovky, kontrolují textace, které se mají nacházet na dané obrazovce. Vývojem aplikace, zákonů, trendů a dalších faktorů se mohou tyto textace měnit. Pokud v testu je uvedena přesná textace, která má být kontrolována, pak při každé změně textace je třeba upravit všechny testovací scénáře, kde se daný text vyskytuje. Tato premisa nám naznačuje, že přímé texty není dobré psát do scénářů z důvodu náchylnosti na jejich změny.

Je-li třeba uvést text, u kterého si zněním ještě v dané fázi vývoje není tester jistý, lze použít spojení: 'je zde text např."XXX YYY"', čímž naznačuje, že text se může slovně lišit, ale zároveň přesně popíše hlavní myšlenku, kterou by měl vyjadřovat.

Obrázek 25 Možnosti jednotlivých kroků testovacího scénáře

1.	Systém XY...	Systém je...	
2.	Kontrola zobrazení...	Zobrazení ...	
3.	Kontrola funkčností..	Funkčnost...	Data pro fukčnost
4.	Zaslání upozornění...	Upozornění...	
5.	Zobrazení obrazovky...	Obrazovka ...	

Zdroj: vlastní tvorba

Je-li třeba uvést text, u kterého si zněním ještě v dané fázi vývoje není tester jistý, lze použít spojení: 'je zde text např."XXX YYY"', čímž naznačuje, že text se může slovně lišit, ale zároveň přesně popíše hlavní myšlenku, kterou by měl vyjadřovat. Další možností jak texty nahrazovat jsou číselníky textů, na které by test odkazoval. Tato varianta je citlivá na možné dělení textace v rámci variant - pokud pro původní text sloužila pouze jedna formulace pro služby A i B, může se stát, že v průběhu času se textace pro službu B více specifikuje a je třeba nahradit odkaz v testu na novou specifikaci testu, která platí pro službu B.

Centrální údržba textu není ale tak nereálná, jelikož tyto texty jsou často generovány a jsou tím pádem zaznamenány v systému (tzn. tato databáze ve firmě existuje). Navíc tato možnost je mnohem méně náchylná na chyby, než je ruční upravování textace v testech nebo hromadné úpravy textací jednotlivých kroků v testech.

5.3.2. Převod scénářů na automatické testy

Údržba testovacích scénářů je s rozvojem firmy stále náročnější. Proto velkou časovou úsporou může být využití automatických scénářů. Tato úspora je nejen z hlediska času na provedení samotného testu, ale i na jeho údržbu. Tyto scénáře mohou být kdykoliv spuštěny v rámci testování a tím verifikována jejich správnost po zásahu do kódu vývojáři.

Další podstatným faktem pro převod manuálních testů na automatické je příprava dat. Zatím co testerovi, který se danou oblastí příliš nezabývá, trvá založení služby A cca 2 hodiny a často se musí ptát ostatních kolegů na radu z důvodu třeba i drobných změn při zakládání služby A, automat, je-li správně naprogramován, stejnou činnost zvládne při plné funkčnosti prostředí za 1 hodinu a bez zátěže dalších pracovníků. Firma tak může ušetřit v průměru 0,5 MD/den na testera u přípravy dat služby A. Tato hodnota je brána s ohledem

na možnost výpadků systému, které mohou zapříčinit pád testu. V tomto případě je třeba zanalyzovat výstup z testu a zadat chybu, nebo vyčkat na větší stabilitu testovacího prostředí.

Ne všechny testy jsou vhodné pro automatizaci. Automatizovat by se měli pouze testy, u kterých se neočekává v brzké době změna funkčnosti, mohou sloužit pro přípravu podkladových dat pro testery a jejich smyslem nejsou kontrolní výpočty nebo validace textů. Tyto testy také mohou sloužit pro kontrolu prostředí po nasazení nových verzí - tzv. smoke testy.

Další výhodou převodů testů na automatické je úbytek testovacích scénářů. Tento úbytek je důsledkem variant, pro které musely být před automatizací napsány scénáře zvlášť. Tyto možnosti lze po automatizaci sloučit do jednoho testu, kdy varianty jsou rozepsány v popisu testu. Takto slučovat testy lze pouze v případech, kdy jsou testy identické, jedinou změnou je pouze taková proměnná, která neovlivňuje průběh testu, kromě této proměnné jsou testy identické.

Testeři a vývojáři, kteří spravují tyto testy, mají obsáhlý rozhled o systémech firmy a zároveň jsou proškoleni ohledně bezpečnosti softwaru, aby tyto testy nemohly být nijak zneužity.

Pro webové služby se pak mohou použít testovací sety přímo v programech pro jejich testování. Jedním z možných programů pro testování webových služeb je SOAPUI. Tento program nabízí širokou možnost pro vytvoření vlastních testovacích setů, tzv. Test Suitě.[1] Tyto Suity obsahují jednotlivé testovací scénáře, které mohou být použity pro funkční. Zátěžové, bezpečností a jiné testy jako např. pro testování maximálních odpovědí služby, provolání služby se specifickými atributy atd. Dalšími možnostmi pro testování webových služeb je např. open source JMeter nebo komerční nástroj Smart Bear.

5.3.3. Školení testerů

Pro práci testera v navrhnutém modelu je třeba zajistit i odpovídající vzdělání pro dostupný software, se kterým by pracoval v rámci testů. Tester by musel ovládat základy SQL

[1] https://www.soapui.org/getting-started/functional-testing.html

jazyka, minimálně základní znalost čtení a úpravy proměnných JAVA kódu, orientaci ve vytváření test suit v programech jako je např. SOAPUI.

Primární znalostí by měla však být metodika psaní testovacích scénářů, jejich spuštění, zadání nalezených chyb do systému a vyhodnocení výsledků testování. Tester by si dále měl se svým nadřízeným domluvit rozšiřování znalostí o systémech firmy a v rámci udržení těchto znalosti i pořádat pro své kolegy školení ohledně jeho získaných znalostí z vyvíjeného projektu. Touto interakcí by se udržovaly informace o systémech mezi zaměstnanci. Tyto informace by mohli pomoci při analýzách projektů k odhalení nejasností, které mohou projekt prodloužit a prodražit i o několik pracovních dní.

Školení by zároveň vytyčila kontaktní osoby za dané oblasti, na které by se tester mohl obrátit v případě dotazů za daný systém. Výstupem školení by pak byly materiály uložené na vnitropodnikové síti, kde by se staly návody pro nově nastupující zaměstnance.

Pravidelnost školení by zároveň zaručovalo udržování materiálů aktuálních v rámci vývoje, takže by tyto informace byly relevantní pro nastudování zkoumané oblasti v rámci analýzy projektu.

Technické školení testerů by dále přispělo k vyššímu využití nástrojů pro testování, které může mít firma k dispozici. Pokud by měl tester dostatečné vzdělání v SQL a JAVA jazyce a dále ovládal programy, jako jsou např. SQL developer, SOAP UI, Eclipse či IntelliJ IDEA, usnadnilo by mu to hledání podkladových dat v databázích, vytvoření testovacích suit pro opakující se testy webových služeb či úpravu napsaných automatických testů, dle jeho potřeb.

5.3.4. Výstup testů

Na základě pečlivě napsaného scénáře, při řádném využití zdrojů správně proškoleným testerem jsou pak možné dvě možnosti výstupu testů. Test prošel bez chyby nebo test obsahoval chyby. Pro zaznamenání možnosti dvě slouží mnoha firmám systém JIRA. Každá společnost ho má definovaný pro své potřeby. Pro testera však jsou potřeba pouze tikety popisující jeho práci mimo testy a pak Bugy, které zaznamenávají chyby v testech. Tyto chyby se dělí na:

- blokující

- nezávažné
- závažné

Blokující

Tyto chyby zabraňují spuštění testu. Nemusí to být pouze chyby vyvíjené funkčnosti, ale jsou to většinou chyby dodávky, prostředí nebo systému. Test nemůže být spuštěn, jelikož nelze splnit podmínky pro jeho spuštění. Výsledkem testu je stav Blocked a test není vůbec spuštěn.

Výjimkou je pokud se při spuštění testu objeví krok, který nelze provést, pokud nebude dodána některá část systému. V takovémto případě lze připravit data, lze spustit test, avšak test nelze dokončit.

Nezávažné

V takovém stavu lze test spustit. V jeho průběhu se pak může objevit nezávažná chyba, která nebrání jeho dokončení. Za nezávažné chyby se považují chyby v textacích, kosmetické chyby na obrazovce (např. odesílací tlačítko zarovnané na střed místo na pravý okraj). Výsledkem testu je stav Caution, důvod je zaznamenán v systému JIRA a odkázán do systému Spira.

Závažné

Test lze spustit, avšak některý z jeho kroků padá na závažnou chybu. Přes tuto chybu nelze dále v testu pokračovat. Test do opravy chyby končí ve stavu Failed a chyba je zadána s kritickou či vysokou prioritou do tiketovacího systému.

Jsou-li všechny chyby opraveny nebo testy neobsahují žádné chyby s kritickou či vysokou prioritou je možnost projekt ukázat zadavateli, který se poté může zapojit do testování.

5.4. Business testování

Tento typ testování je velmi důležitou, i když bohužel ve firmách často opomíjenou složkou testování nových funkčností. Zadavatel by měl na základě domluvy s vývojovým týmem otestovat projekt a říct, zda mu takto navržený projekt vyhovuje a splňuje jeho očekávání na základě dodaného zadání.

Na testování by měli být zvolení takový zaměstnanci, kteří se systémem pracovali, jedná-li se o inovaci systému, nebo takový zaměstnanci, kteří v budoucnosti budou systém využívat a mají představu, jak by měl vypadat.

Toto testování je prováděno tzv. volnými testy, kdy bussines testuje své strategie a pohled na projekt a je tím pádem neovlivnění test analýzou. Toto nezávislé testování pak vede buď k odhalení nesrovnalostí, nebo k potvrzení vyvinuté funkčnosti.

V případě těchto nesrovnalostí je vývojový tým informovaný o rozsahu a navrhované nápravě, či původní myšlence, která nemusela být ze zadání zřejmá. Její oprava/náprava je pak podrobena stejné interakci, jako je to popsáno v kapitole 4.3 Testování projektu.

Všechny schválené připomínky z bussines testování jsou zaneseny do testovacích scénářů, případně jsou tyto scénáře vytvořeny, pokud zde jsou nepopsané možnosti, které mohou způsobovat komplikace při využívání funkčnosti.

5.5. Shrnutí kapitoly 5

Kapitola 5 Návrh řešení obsahuje vypracovaný cíl práce, kde je zde popsán mnou navrhnutý postup vypracování projektu od zadání, vytvořeném na základě myšlenky business vlastníka a zpracované business analytikem, až po předání vyhotoveného projektu zpět zadavateli. Každou jednotlivou fázi doplňují přehledné UML diagramy, které vizuálně doplňují napsaný text.

Tento návrh řešení je navržen podle "Požadavku 1" z knihy *Foundations of software testing: ISTQB certification* (6), kdy základním principem je pečlivě vytvořený požadavek business vlastníka, od kterého se pak odvíjí celý další proces a to jak sestavení vývojového scrum týmu, tak následné nižší náklady na opravu změn zadání.

Kapitola se také zabývá předáváním znalostí v rámci testovacího týmu, které je navrženo pomocí školení a jejich výstupu na firemní síti, kde by takové informace byly vždy dostupné a díky pravidelnosti pořádání těchto akcí i aktuální.

6. Závěr

Každý den testujeme nové služby či produkty. Za tyto produkty a služby vydáváme peníze a ty co neutratíme, chceme mít bezpečně uložené ve své bance, investované do fondů či akcií. Nechceme však za ochranu takto uložených peněz platit horentní sumy a proto je třeba, aby společnosti dbaly na nízkých nákladech, ale ne za cenu bezpečnosti či rizika ztráty.

Zefektivnění práce a to nejen u testů může firmě ušetřit peníze, zvýšit spokojenost jak zákazníků, tak zaměstnanců. Tato spokojenost se odvíjí od kvality poskytované služby, a aby kvalita zůstala zachována, tak o to se starají testeři spolu s vývojáři. Vedení firmy s businessem se pak stará, aby společnost stále nabízela to, co si zákazníci přejí a využijí.

Při zajišťování kvality je třeba mít řádně sepsaný postup a při minimalizování nákladů je třeba řádně využívat zdroje. Proškolený personál s propracovanou metodikou, který pracuje pro firmu rád, by měl být základem každé firmy, která chce být na trhu úspěšná.

Jako jeden z cílů práce jsem si stanovila návrh ideálního postupu při zpracování projektu s velkým zaměřením na testovací část, kterým jsem popsala v kapitole 5. Žádné testování ale nemůže začít, není-li první dodáno promyšlené zadání, sestavený správně specializovaný vývojový tým a určený analytik pro konzultaci případných nejasností.

Samotné testování by nemělo ustrnout pouze na popisování zanalyzovaných skutečností do testovacích scénářů, ale mělo by se dále vyvíjet směrem k automatizaci, která šetří čas a omezuje případnou chybu lidského faktoru.

Tester samotný by se měl ve svých znalostech v rámci systému firmy, pro kterou pracuje dále rozvíjet za podpory svých nadřízených. Svou práci dělat zodpovědně a přehledně dle stanovených metodik.

Metodiky by měly být nastaveny tak, aby co nejpřesněji objasňovaly současné pochody ve firmě a doplněním těchto metodiky by pak byly výstupy ze školení, pořádaných zaměstnanci, které by byly pravidelné. Materiály ze školení by byly zveřejňovány na vnitropodnikovou wikipedii, kde by k nim měl tester přístup a mohl by si nastudovat oblast, kterou analyzuje.

Další cíl, který jsem definovala pro tuto práci, je splněn pomocí přehledných UML diagramů, které jsou v práci zobrazeny v jednotlivých kapitolách. Tyto diagramy popisují jak současný stav ve firmě, tak i navržený postup a doplňují tak text diplomové práce.

Navržené řešení počítá spíše se stabilnější základnou zaměstnanců a příliš se nehodí pro firmy, kde je velká fluktuace zaměstnanců. Pro takovýto typ firem je lepší pracovníky vyškolit pouze pro typ práce, na kterou jsou najatí.

Citovaná literatura

(1) BUCHALCEVOVÁ, A. 2005. *Metodiky vývoje a údržby informačních systémů*. Praha : Grada, 2005. ISBN 80-247-1075-7.

(2) BUCHALCEVOVÁ, A., STAVOVSKÁ, I., a ŠIMŮNEK, M. 2002. *Základy softwarového inženýrství - základní témata*. Praha : Oeconomica, 2002. 80-245-0346-8.

(3) BUCHALCEVOVÁ, A. 2009. *Metodiky budování informačních systémů*. Praha : Oeconomica, 2009. ISBN 978-80-245-1540-3.

(4) DEBONO, Mark. *Functional vs Non Functional Testing*. In: Reqtest [online]. Apr 17, 2012 [cit. 2016-03-13]. Dostupné z: http://www.reqtest.com/blog/functional-vs-non-functional-testing/

(5) GELPERIN, D. a HETZEL, B. 1988. *The Growth of Software Testing*. s.l. : CACM, Vol. 31, No. 6, 1988. ISBN 0001-0782.

(6) GRAHAM, Dorothy, VAN VEENENDAAL, Erik a EVANS, Isabel. 2008. *Foundations of software testing: ISTQB certification*. s.l. : Cengage Learning EMEA, 2008.

(7) KNESL, Jiří. 2012. *SprintMethod. agilní metodika vycházející ze Scrumu*. [Online] SprintMethod, 2012. [cit.: 201-03-01]. Dostupné z: http://www.sprintmethod.cz/.

(8) JIŘÍČKOVÁ, Martina a ONDEK, Štefan. 2015. *Agilní metody v projektovém řízení*. ITMforum. [Online] ITMforum, 06 04, 2015. [cit.: 2016-03-27]. Dostupné z: http://www.itmforum.cz/rubriky/rizeni-projektu-a-jejich-portfolia/agilni-metody-v-projektovem-rizeni/. 2336-582X.

(9) MEERTS, Joris a GRAHAM, Dorothy. *The History of Software Testing*. Testing References. [Online] Testing References. [cit: 2016-03-25]. Dostupné z: http://www.testingreferences.com/testinghistory.php.

(10) MURPHY, Craig. 2005. *Improving Application Quality Using Test-Driven Development* (TDD). METHODS & TOOLS. Global knowledge source for software development professionals, 2005, Vol. 1, 13.

(11) PATON, R. 2002. *Testování softwaru*. Praha : Computer Press, 2002. ISBN 80-7226-636-5.

(12) SCHWABER, Ken a SUTHERLAND, Jeff. 2013. *Průvodce Scrumem*. Scrum Guides. [Online] 07 2013. [cit.: 2016-03-30]. Dostupné z:. Dostupné z:

http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-CS.pdf.

(13) 2011. *Slovník současné češtiny*. Brno : Lingea, 2011. ISBN 978-80-87471-27-2.

(14) SOFTWARE TESTING STUFF. *Functional Testing Vs Non-Functional Testing*. Softwartestingstuff.com [online]. [cit. 2016-03-18]. Dostupné z: http://www.softwaretestingstuff.com/2007/10/functional-testing-vs-non-functional.html

(15) 2016. *Software testing/History of testing - Wikiversity*. Wikiversity. [Online] MediaWiki, 02 26, 2016. [cit.: 2016-03-26]. Dostupné z: https://en.wikiversity.org/wiki/Software_testing/History_of_testing.

(16) VOGEL, Jeffrey. 2014. *Why PE and Agile haven't always gone together*. Tech MVP. [Online] Tech MVP, 07 04, 2014. [cit.: 2016-03-24]. Dostupné z: http://techmvp.net/2014/07/04/agile-pe/.

(17) WEINBERG, Gerald. 2001. *An Introduction to General Systems Thinking.* Dorset : House Publishing, 2001. 10: 0932633498.

(18) ZAFAR, Rehman. *What is software testing? What are the different types of testing?* In: Codeproject [online]. Mar 20, 2012 [cit. 2016-03-18]. Dostupné z: http://www.codeproject.com/Tips/351122/What-is-software-testing-What-are-the-different-ty