TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Vývoj a testování stavebního softwaru Aspe

Bakalářská práce

Studijní program: B6209 – Systémové inženýrství a informatika
Studijní obor: 6209R021 – Manažerská informatika

Autor práce: **Marek Kozák**
Vedoucí práce: Ing. Vladimíra Zádová, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Economics

Development and testing of building software Aspe

Bachelor thesis

Study programme: B6209 – System Engineering and Informatics
Study branch: 6209R021 – Managerial Informatics

Author: **Marek Kozák**
Supervisor: Ing. Vladimíra Zádová, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Marek Kozák**
Osobní číslo:	**E13000030**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Manažerská informatika**
Název tématu:	**Vývoj a testování stavebního softwaru Aspe**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

1. Přístupy k testování: charakteristika a vyhodnocení
2. Možnosti softwarové podpory v rámci stavebnictví
3. Vývoj a testování aplikace
4. Zhodnocení přínosu řešení

Rozsah grafických prací:

Rozsah pracovní zprávy: **30 normostran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. Řízení kvality softwaru: průvodce testováním. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.
SPILLNER, Andreas, Tilo LINZ a Hans SCHAEFER. Software testing foundations: a study guide for the certified tester exam: foundation level, ISTQB compliant. 4rd ed. Sebastopol, CA: Distributed by O'Reilly Media, 2014. ISBN 978-1-937538-42-2.
PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. Jak testuje software Microsoft. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5.
SVOZILOVÁ, Alena, Ken JOHNSTON a Bj ROLLISON. Projektový management. 2., aktualiz. a dopl. vyd. Praha: Grada, 2011. ISBN 978-80-247-3611-2.
Elektronická databáze článku ProQuest (knihovna.tul.cz).

Vedoucí bakalářské práce: **Ing. Vladimíra Zádová, Ph.D.**
Katedra informatiky

Konzultant bakalářské práce: **Ing. Zbyněk Lipavský**
IBR Consulting s. r. o. Praha

Datum zadání bakalářské práce: **31. října 2015**

Termín odevzdání bakalářské práce: **31. května 2017**

doc. Ing. Miroslav Žižka, Ph.D.
děkan

Technická univerzita v Liberci
Ekonomická fakulta
L.S.

doc. Ing. Jan Skrbek, Dr.
vedoucí katedry

V Liberci dne 31. října 2015

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Poděkování

Rád bych touto cestou poděkoval své vedoucí bakalářské práce paní Ing. Vladimíře Zádové, Ph.D. za čas, ochotu a cenné rady, které mi během zpracování práce poskytla. Dále bych rád poděkoval firmě IBR Consulting, s.r.o. za umožnění práce na vývoji Aspe 10.

Anotace

Bakalářská práce Vývoj a testování stavebního softwaru Aspe se zabývá procesem vývoje a testování nové verze stavebního rozpočtového softwaru Aspe 10. Práce je rozdělena do dvou částí. Teoretická část představuje techniky a přístupy využívané během testování softwaru a vymezuje jeho postavení v rámci vývoje. Dále jsou zde představeny hlavní metodické přístupy uplatňované v rámci vývoje softwaru a přiblížena současná situace a trendy ve využívání softwarové podpory ve stavebnictví.

Praktická část se podrobně zabývá vývojem programu Aspe 10, který je hlavním produktem firmy IBR Consulting. Stručně popisuje samotný program, jeho historii a průběh vývoje nové verze se zaměřením na proces testování. Popisuje interní proces spolupráce jednotlivých členů vývojového týmu a jejich role v rámci vývoje. Na závěr je celý proces zhodnocen a jsou navrženy změny pro jeho zefektivnění.

Klíčová slova:

Aspe, Stavební software, Testování softwaru, Agilní vývoj, Proces vývoje softwaru

Annotation

Bachelor thesis development and testing of building software Aspe is engaged in the development process of the new version of the construction management software Aspe 10. Work is divided into two parts. The theoretical part presents techniques and approaches used during software testing and define its position within the development. Furthermore, there are also presented the main methodological approaches used in the software development and the approach of the current situation and trends in the use of software support in the construction industry.

The practical part deals in detail with the actual development of Aspe 10, which is the main product of the company IBR Consulting. It briefly describes the program itself, its history and the course of development of the new version with a focus on the testing process. It describes the process of internal cooperation among the various members of the development team and their role in development. At the end of the whole process is evaluated and changes are designed to make it more effective.

Keywords:

Aspe, Construction management software, Software testing, Agile development, Software development

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek a značek

BI – Business Intelligence

CAD – Computer Aided Design

CRM – Customer Relationship Management (Řízení vztahů se zákazníky)

DMS – Document Management Systém (Systém pro správu dokumentů)

ERP – Enterprise Resource Planning

FURPS – metoda od firmy Hewlett-Packard pro ověření kvality dodávaného softwaru.

GUI – Graphical user interface (Grafické uživatelské rozhraní)

ICT – Information and Communication Technology (informační a komunikační technologie

IS – Informační systém

ISTQB – International Software Testing Qualifications Board

MIP – Manažerský informační portál

SQL – Structured Query Language

SQuaRe – Software Quality Requirements and Evaluation

UI – User interface (Uživatelské rozhraní)

XML – Extensible Markup Language (rozšiřitelný značkovací jazyk)

ZBV – Změna během výstavby

Úvod

Řízení kvality softwaru je v dnešní době nedílnou součástí procesu vývoje softwaru. V minulosti nebyla této oblasti vývoje softwaru věnována dostatečná pozornost, což vedlo k mnoha závažným problémům způsobeným počítačovými chybami. Na odstranění těchto chyb, pokud to bylo vůbec možné, byly vydány nemalé finanční prostředky. V souvislosti s tím lze vzpomenout na známý problém roku 2000 (Y2K) nebo havárii přistávacího modulu sondy NASA - Mars Climate Orbiter. (Patton, 2002)

Při vývoji rozsáhlých podnikových aplikací je kromě finančních nákladů na opravu chyb potřeba myslet také na celkový obraz firmy, kdy vydání řádně neotestovaného programu může vést až ke ztrátě zákazníka a poškození renomé firmy.

Tématem této bakalářské práce je vývoj a testování nové verze rozpočtového stavebního softwaru Aspe s číslem 10 určeného pro přípravu a realizaci staveb. Jelikož se jedná o vývoj rozpočtového programu, role testování je nezbytnou součástí vývoje. Tato práce se na tento proces zaměřuje a přibližuje tak jeho úlohu v rámci vývoje. Cílem této práce je vyvinutí spolehlivé a řádně otestované nové verze programu Aspe 10, která bude připravena k distribuci mezi zákazníky a plně nahradí předchozí verzi programu.

V teoretické části kvalifikační práce bude přiblížen proces testování softwaru a s tím související nejčastěji používané přístupy v rámci vývoje softwaru. Dále budou přiblíženy současné trendy a možnosti využití softwarové podpory v rámci stavebnictví.

Praktická část je zaměřena na samotný vývoj stavebního softwaru Aspe 10. Poskytne detailnější pohled na proces vývoje s důrazem na role a procesy v rámci vývojového týmu. Blíže popisuje testování aplikace v rámci jednotlivých fází vývoje. Na závěr je celý proces zhodnocen a doporučena řešení ke zlepšení celého procesu vývoje.

Zhodnocení současného stavu

Testování softwaru je dnes věnována stále větší pozornost a ve velkých organizacích se přibližuje úrovni programování a implementování technologií. Tento vývoj dal vzniknout i mezinárodní radě pro testování ISTQB (International Software Testing Qualifications Board), která utváří standardy pro certifikaci testerů. Na základě ISTQB velmi dobře popisuje samotný obsah testování Andreas Spillner a kol. v knize *Software Testing Foundations*, která slouží také jako příručka pro výše zmíněné certifikace.

Poslední desetiletí byla ve znamení velkého boomu v oblasti vývoje softwaru. Hlavním milníkem byl přelom milénia, kdy se začaly naplno prosazovat nové – agilní metodiky, které mění pohled na celý proces vývoje a testování softwaru. Velmi dobrý přehled agilních a rigorózních metodik s jejich porovnáním nabízí Alena Buchalcevová ve své knize *Metodiky budování informačních systémů.* Výběru metodiky s návazností na testování se ve své diplomové práci *Metodika testování podle mezinárodních praktik a standardů* věnuje Iveta Králová, kde se zabývá tvorbou metodiky pro české prostředí za využití mezinárodních standardů ISTQB.

V dnešní době lze sledovat hned několik trendů. Podle *Cigniti Predicts Software Testing Trends for 2016* je očekáván další nárůst v oblasti bezpečnostního testování z důvodu velkého nárůstu množství sdílených dat přes webové služby, mobily a cloudové aplikace. Obecně se očekává výrazný rozvoj v oblasti automatizace testů, spojených se stále se zrychlujícím vývojem. Zrychlující se vývojové cykly potvrzuje i uveřejněný průzkum *Testing trends in 2016: A survey of software professionals* od společnosti SauceLabs, který si ovšem všímá, že velká část firem stále nedokáže naplno využít výhody agilního vývoje.

1 Testování softwaru

Testování softwaru je obsáhlý pojem a pro přiblížení této problematiky je v úvodu kapitoly představen samotný termín *testování softwaru*. V další části kapitoly jsou popsány základní termíny užívané v rámci vývoje softwaru. Následně jsou rozebrány důvody důležitosti řádného testování z pohledu současnosti a jeho dopad na vyvíjený produkt. Ke konci kapitoly jsou představeny základní testovací techniky používané k zajištění kvality.

1.1 Pojem testování softwaru

Testování softwaru je staré jako vývoj sám. V minulosti ovšem bylo testování chápáno naprosto odlišně než dnes a nebylo rozlišováno jako samostatná část vývoje, což bylo zapříčiněno hlavně omezeností výpočetní síly a její ceny, ale také složitostí jednotlivých programů. V roce 1979, kdy již bylo testování vnímáno jako samostatná část patřící pod řízení kvality. Glenford Myers (2004, s. 6) testování definuje jako *„Testing is the process of executing a program with the intent of finding errors.“* V překladu: *„Testování je proces spouštění programu, za účelem hledání chyb“*, oproti tomu Patton (2002, s. 17) definuje cíle softwarového testera jako vyhledávání chyb v co nejkratším časovém úseku a zajištění jejich nápravy. Asi nejucelenější definici splňující novodobý pohled na pojem testování nabízí Graham (2008, s. 13), kde říká, že testování lze chápat jako: *„proces skládající se z aktivit během celého životního cyklu vývoje softwaru. To zahrnuje statické a dynamické testování*[1]*, plánování, přípravu a vyhodnocení softwarových produktů a souvisejících pracovních výstupů, s cílem určit, zda vše odpovídá specifikovaným požadavkům, vše plní svůj účel a v neposlední řadě s cílem odhalování defektů.“*

Z pohledu ANSI/IEEE 1059 standardu je softwarové testování definováno jako: *“ Testing is the process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate the features of the software item.“* V překladu: *„Proces analyzování softwaru za účelem odhalení rozdílů (chyb) mezi stávajícími a požadovanými podmínkami a následným vyhodnocením softwaru*“ (Sponsor, 1994)

[1] **Statické testování** – testování na úrovni specifikace (software není spuštěn)
Dynamické testování – testování spojené s používáním softwaru

Je patrné, že pohled na testování softwaru se v průběhu času měnil. Z počátku bylo testování vnímáno pouze jako činnost, která měla za cíl vyhledat chyby programu. Dnes již je testování chápáno jako komplexní činnost, které má za cíl dosáhnout kvalitního programu, což znamená nejen program bez pádů a chyb, ale také aby byl uživatelsky přívětivý a byla k němu vedena i kvalitní dokumentace.

1.2 Základní pojmy v testování

Jak uvádí Patton (2002), či Roudenský (2013): pro testování softwaru je nezbytná znalost základních a všeobecně užívaných pojmů a praktik. Je jasné, že pro jednotlivé stavy existuje mnoho synonymních výrazů, které se mohou mezi jednotlivými podniky lišit, ale jejich používání a striktní vyžadování je čistě na stanovených konvencích uvnitř organizace. Sjednocení pojmů může značně ulehčit komunikaci v rámci vývojových týmů a předejít tak mnohým nesrovnalostem.

1.2.1 Softwarová chyba

Pro vysvětlení pojmu *softwarová chyba* je nutné si nejdříve pomoci pojmem *specifikace produktu*. Specifikace je dohodou mezi vývojovým týmem daného softwaru. Definuje a podrobněji popisuje, jak bude daný produkt vypadat a jak se bude chovat. Zároveň stanovuje, jak by se výsledný program chovat neměl.

Pokud se aktuální chování programu jakkoliv liší od očekávaného chování programu, mluvíme o chybě. Kromě pojmu „chyba“ existuje nesčetné množství označení pro selhání programu, jako „defekt“, „závada“, „selhání“, „anomálie“ atd. Jejich použití závisí jen na konvenci v dané organizaci. V oblasti softwarového vývoje se chyba běžně označuje jako „bug“, v překladu brouk. Ke vzniku tohoto označení se pojí příběh z roku 1947 z Harvardské univerzity. V této době byl nejmodernějším zařízením počítač Mark II, obrovský sálový počítač, navržený Howardem Aikenem. Jednoho dne najednou stroj přestal pracovat. Když technici usilovně zjišťovali příčinu problému, našli uvnitř počítače zaklíněnou můru. Do systému ji nejspíše přilákalo světlo a teplo a při přistání mezi kontakty relé ji usmrtilo vysoké napětí. V záznamech o chybě se poté objevilo, že

v počítači byl nalezen „bug“. Obdobné přísloví používá i v češtině, kdy říkáme: „ještě to má mouchy.“ (Patton 2002, s. 9)

O softwarové chybě hovoříme tehdy, pokud je splněna alespoň jedna s následujících podmínek:

- software nedělá to, co je uvedeno ve specifikaci,
- software dělá něco, co by podle specifikace dělat neměl,
- software dělá něco, co není uvedeno ve specifikaci,
- software nedělá něco, co není uvedeno ve specifikaci, ale specifikace by se o tom zmiňovat měla,
- software není uživatelsky přátelský, je těžko srozumitelný, obtížně se s ním pracuje, je pomalý nebo se tester domnívá, že s jeho používáním bude mít koncový zákazník problém (Patton 2002, s. 14).

Poslední bod může zahrnovat prakticky vše. Tester funguje jako přechodový můstek na cestě softwaru k zákazníkovi. Proto je zde důležité, aby se tester vžil do pozice uživatele a dokázal odhadnout, co zákazník bude považovat za složité ovládání – mohou to být příliš malé ikony, nebo jen nepraktické uspořádání formuláře. Pro maximální efektivitu je vhodné, aby byl tester v kontaktu se zákazníkem a mohl s ním ihned konzultovat změny[2], které se dotknou uživatelského rozhraní aplikace (dále jen UI).

1.2.2 Verifikace a validace

V souvislosti kontroly zajišťování kvality softwaru se často setkáváme s pojmy verifikace a validace, někdy také označované pouze zkratkou V&V aktivity. I když tyto dva pojmy znějí velmi podobně, v oblasti vývoje softwaru zde existuje velmi důležitý rozdíl v chápání těchto pojmů.

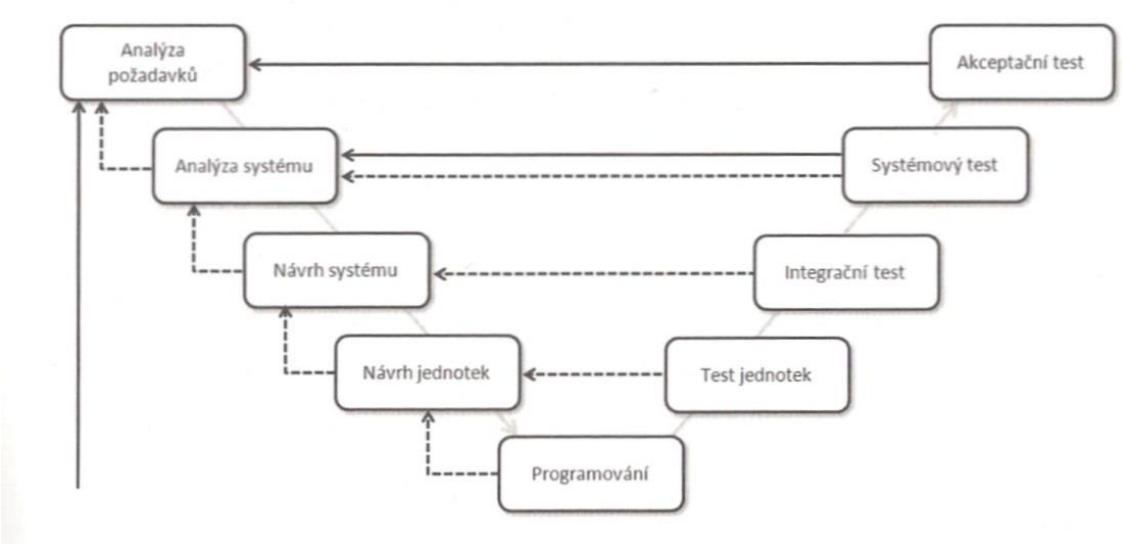
Verifikace je proces, který má za úkol ověřit, zda vyvíjený software odpovídá specifikaci (návrhu) a zda je správně vytvořen. Zpravidla se týká pouze jedné části vývojového

[2] Přímá komunikace se zákazníkem je jednou s výhod agilního přístupu

procesu. O konkrétních verifikačních aktivitách se více zmiňuje norma ISO/IEC 12207:2008. Oproti tomu *Validace* ověřuje výsledky vývoje, zda vyhovují původním požadavkům a jeho zamýšlenému použití (Roudenský 2013, s. 24-26), (Spillner 2014, s. 41). Barry W. Boehm tyto dva pojmy shrnuje v neformální definici jako:

- *„Verifikace: Vytvářím produkt správně?*
- *Validace: Vytvářím správný produkt?“* (Roudenský 2013, s. 25)

Verifikační a validační aktivity velmi dobře ilustruje tzv. V-model, který vychází z vodopádového modelu, který je blíže popsán v jedné z dalších kapitol. Hlavní myšlenkou V-modelu je chápání vývoje a testování softwaru jako souběžné a navzájem propojené aktivity se stejným významem. Levá strana reprezentuje proces vývoje, kdežto pravá strana integraci a testování.



Obr. 1: V-model

Zdroj: Roudenský (2013, s. 27)

1.2.3 Kvalita a spolehlivost

Kvalita a spolehlivost bývá často brána jako jedno a to samé. Ne zřídka panuje domněnka, že pokud se během testování programu dosáhne stavu, kdy je software dostatečně stabilní a

spolehlivý, bude tím pádem i kvalitní. To ale nemusí být vždy pravidlem, protože spolehlivost je pouze jedním z mnoha ukazatelů kvality.

Uživatel si může představit pod pojmem kvalitní software nepřeberné množství funkcí a vlastností, např. schopnost spustit program na starém počítači nebo na různých verzích operačního systému, možnost telefonické podpory zajištěné softwarovou společností, ale mohou to být i takové detaily, jako špatně zvolená velikost, či barva dialogových formulářů nebo nevhodně navržený design krabice se softwarem. *Spolehlivost* vyjadřuje pouze, jak často program havaruje nebo jak často dochází ke špatné práci s daty uvnitř programu. Tato vlastnost je samozřejmě velmi důležitá, ale je k ničemu, pokud se s programem špatně pracuje a není uživatelsky přátelský. K tomu, aby se dosáhlo nejen spolehlivosti programu, ale i vysoké kvality softwaru, je nutné, aby během celého vývoje byla prováděna jak odpovídající verifikace, tak i validace (Patton 2002, s. 42)

Snahy definovat pojem kvalita softwaru daly vzniknout standardům v rámci mezinárodních norem. V dnešní době je nejvíce používán systém norem ISO/IEC 25000-25099 v rámci projektu *SQuaRe* (*Software Quality Requirements and Evaluation* - v překladu „Požadavky na kvalitu softwaru a její hodnocení“) Kvalita softwarového produktu je zde tvořena z osmi charakteristik:

- Funkčnost
- Účinnost
- Kompatibilita
- Použitelnost
- Bezporuchovost
- Bezpečnost
- Udržovatelnost
- Přenositelnost

Každá charakteristika se dále dělí na další podcharakteristiky, které jsou předmětem měření. Jednou z měr je i testování. *Testování tedy poskytuje informace o kvalitě produktu*.

(Roudenský 2013, s. 21-23). Pro měření kvality existuje hned několik modelů kvality. Mezi nejznámější modely kvality patří FURPS[3], popř. v rozšířené podobě model FURPS+.

1.2.4 Tester

Cílem softwarového testera je především odhalovat chyby. Podle BussinesDictionary.com je tester *„Technik, který provádí předepsané testy softwarových programů a aplikací před jejich zavedením s cílem zajistit kvalitu, integritu designu a správnou funkčnost. Tester používá přísné testovací metody, včetně rozsáhlých simulací chování koncových uživatelů za účelem nalezení "chyb", které jsou následně opraveny programátory“.* Patton k jeho povinnostem dále přidává také důraz na rychlost objevení samotných chyb a jejich opravení v rámci vývoje. Dále také uvádí i seznam vlastností, které by dobrému softwarovému testerovi neměli chybět. Mezi nejdůležitější vlastnosti patří neúnavnost, kdy se dobrý softwarový tester neustále snaží znovu navodit chybu, která se vyskytla při nějaké činnosti, ale je velmi nejasné, z jakého důvodu nastala a velmi obtížně se znova navozuje. Další důležitou vlastností je kreativita, kdy je zapotřebí využít všechny běžné, ale i krajně neobvyklé postupy při práci se softwarem. Zároveň musí mít i dobrý úsudek, aby si dokázal naplánovat, co a jak bude testovat a rozpoznat, zda chování programu je skutečně chybou. V neposlední řadě musí být přesvědčivý a zároveň diplomatický, aby dokázal přesvědčit zbytek vývojového týmu o závažnosti chyby, která nemusí být z počátku vždy tak zřetelná. Není od věci, když tester má alespoň základní znalosti v programování, aby lépe pochopil funkčnost, a způsob jakým je software programován.

1.2.5 Bug report

Reportování nalezených chyb je nedílnou součástí softwarového vývoje. Po nalezení chyby je nezbytné tuto chybu ohlásit vývojovému týmu formou tzv. „bug reportu“, v češtině se můžeme setkat také s pojmem hlášení o chybě.

K organizaci takto nahlášených chyb je využíváno připomínkových systémů tzv. „Bug tracking systems“, ve kterých se shromažďují tyto pro vývojáře nezbytné informace. Aby mohla být chyba náležitě a včas opravena, je nutné ji také správně popsat. Existuje mnoho

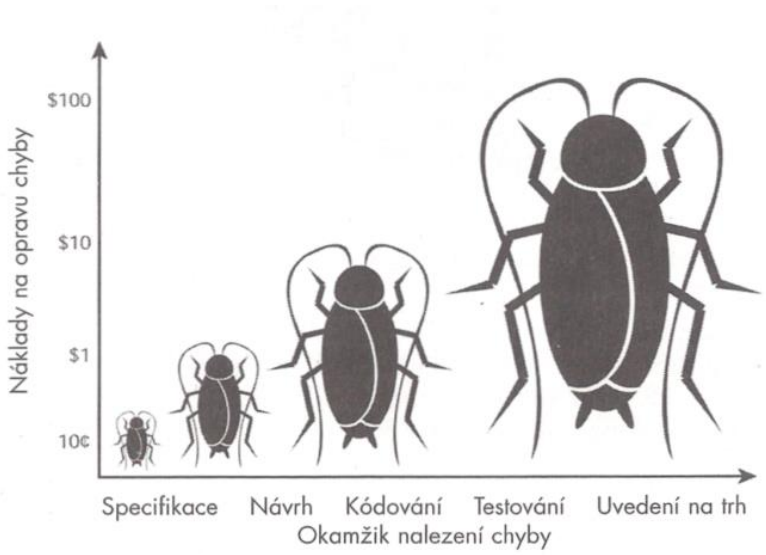
[3] Zkratka FURPS reprezentuje: F-funkčnost (functionality), vhodnost k užití (usability), spolehlivost (reliability), výkon (perofrmance), schopnost být udržována (supportability).

doporučení jak psát bug report, např. portál softwaretestinghelp.com popisuje vzorový bug report, který by měl obsahovat název chyby, což znamená popis činnosti, při které chyba nastala, dále tzv. „bug ID“ – tzn. číslo chyby, které je nejčastěji automaticky vytvořeno po uložení. Hlášení o chybě by dále mělo obsahovat verzi programu, vážnost chyby a prioritu k opravě většinou udávané škálou 1-5, typ chyby a vývojové prostředí. Současně by měl být přiložen bližší popis chyby, který muže obsahovat např. screenshot (foto obrazovky) a postup pro znovu navození chyby. Dále je zde popsáno mnoho dalších náležitostí, které se mohou v bug reportu objevit (Sample bug report, 2015). Zimmerann (2010) se ve své studii zabývá tím, jakým informacím z bug reportu přikládají vývojáři, ale i testeři největší váhu. Z průzkumu jasně vyplývá, že pro programátora je nejcennější, pokud je popsán postup pro znovu navození chyby, na dalších místech se umísťují tzv. Stack Trace, neboli informace o probíhajícím podprogramu s výpisem kódu, kde došlo k chybě, a popis testovacího scénáře, kde je popsána prováděná aktivita s očekávaným výsledkem. Na druhou stranu z průzkumu vyplývá, že mnohdy není důležitá verze operačního systému. Neméně zajímavým zjištěním je také hodnocení duplicit v bug reportech, kdy na rozdíl od všeobecně branného názoru, kde se má za to, že duplicitní hlášení plýtvají programátorským časem, mnoho programátorů uvedlo, že mnohdy jsou zde poskytnuty klíčové dodatečné informace.

1.3 Účel testování

Jak již bylo zmíněno, pohled na testování softwaru se v průběhu let velmi změnil. Dnes se již většinou nejedná o přehlíženou část vývoje softwaru, kdy se vývojové týmy k testování často dostaly až na konci vývoje, nebo vůbec a samotné testování připadlo až na koncového zákazníka. Výpočetní technika a programové vybavení dnes často díky své efektivitě nahrazují lidský faktor v mnoha odvětvích a jsou velmi významným faktorem celkového technologického růstu lidstva. Je těžké si představit, že by došlo k tak velkému pokroku, pokud bychom nevyužili výpočetních možností počítačů k simulaci výpočtů složitých matematických modelů nebo k automatizaci a řízení výroby. Ale i tyto technologie jsou produkty lidské činnosti, a proto nejsou bezchybné, takže je nutné klást důraz na důkladné otestování produktů, které jsou umístěny na trh. Neodchycené chyby, které se vyskytují ve větší míře, než je u podobných systémů obvyklé, mohou vést díky špatnému rozhodnutí projektového managementu k velkým finančním ztrátám nebo ztrátě

reputace. To může mít za následek ztrátu stávajících zákazníků, ale i ztrátu zájmu potenciálních zákazníků, či obchodních parterů o využití produktu a poskytovaných služeb. Na druhou stranu se může stát, že produkt bude na trh vypuštěn cíleně dříve s pravděpodobně vyšším obsahem chyb v rámci dosažení strategické výhody v konkurenčním boji o nové zákazníky. V tomto případě je počítáno s rizikem a je nabízen nadstandartní servis např. bezplatných záplat nebo aktualizací (Roudenský 2013, s. 48-49). Testováním se snažíme minimalizovat náklady na vývoj softwaru při zajištění požadované kvality, což Patton ilustruje na obrázku, kde je názorně ukázána cena opravy chyby v závislosti na fázi vývoje softwaru, kdy je objevena.

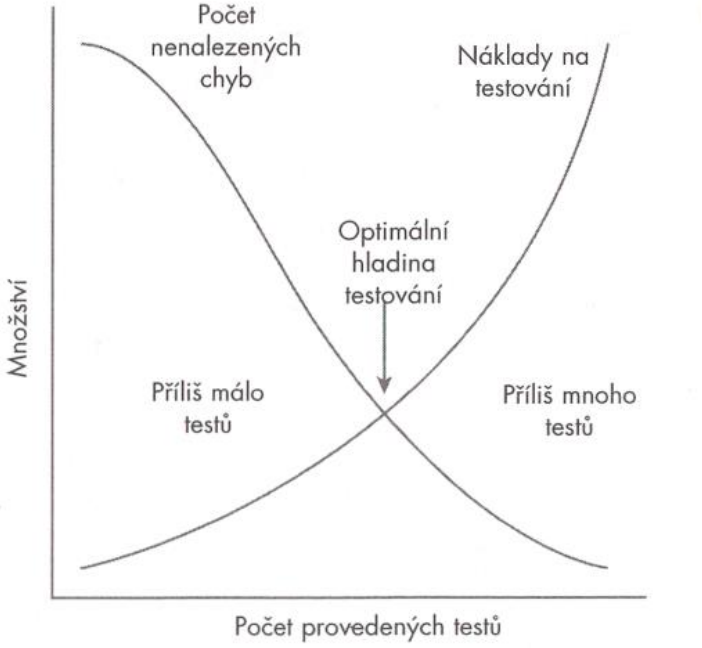


Obr. 2: Náklady na opravu chyby

Zdroj: Patton (2002, s. 16)

Na druhou stranu žádný software nelze otestovat kompletně a to hlavně z finančních a časových důvodů. Testovaný software je buď příliš velký, nebo existuje velké množství vstupů a výstupů, které není možné testováním pokrýt. Z tohoto důvodu můžeme prohlásit, že testování softwaru nikdy nemůže prokázat, že chyby neexistují. Testováním můžeme pouze prokázat, že chyby existují, ale ne vyloučit absolutní bezchybnost programu z důvodů samotné definice chyby, kdy chyba může být pouze v očích jednotlivce. Proto je testování postaveno na riziku, které musí vývojář podstoupit a najít optimální rovnováhu

mezi množstvím provedených testů a případném počtu odhalených chyb (Patton 2002, s. 33-35).



Obr. 3: Optimální hladina testování

Zdroj: Patton (2002, s. 36)

Zde platí pravidlo: *„Čím více chyb najdete, tím více chyb v softwaru je“*. Není výjimkou, že tester dlouho nemůže žádnou chybu najít, ale poté jich objeví hned několik za krátký okamžik. Důvodů může být hned několik – programátor může mít špatný den, nebo je náchylný k opakování určitého typu chyby, ale někdy jsou objevené a zdánlivě spolu nesouvisející chyby důsledkem vážnější chyby skryté v návrhu nebo architektuře softwaru. Nesmí se však zapomínat, že tvrzení může platit také naopak a program je opravdu velmi dobře napsán, takže obsahuje minimální množství chyb. Je potřeba si dát pozor na tzv. **Paradox pesticidů**, který byl prezentován v roce 1990 Borisem Beizerem. Autor zde porovnává softwarové chyby („bugy“) s opravdovým hmyzem a dochází k analogii, kdy opakované používání stejných testovacích postupů funguje podobně jako používání stejného pesticidu na hubení hmyzu, kdy se hmyz stává proti pesticidu časem imunní a je potřeba použít jiný. (Patton 2002, s. 35-37).

1.4 Testovací techniky

Tato kapitola si klade za cíl představit základní postupy využívané v rámci testování a zajištění kvality softwaru. Výběr techniky je závislý na konkrétních situacích a produktu, který je testován.

1.4.1 Statické a dynamické testování

Jedním ze základních rozdělení způsobu testování je dělení na *statické testování* a *dynamické testování.* Statické testování lze chápat jako testování nespuštěného programu, což zahrnuje zkoumání návrhové dokumentace, databázových modelů nebo zdrojového kódu tak, že program nespouštíme. Díky tomu může statické testování probíhat již v raných fázích vývoje softwaru a patří mezi typické verifikační aktivity. Tento způsob je velmi efektivní a náklady na opravení chyby jsou velmi nízké (Roudenský 2013, s. 28-29), (Patton 2013, s. 49-50). Oproti tomu dynamické testování zahrnuje práci se spuštěným programem v pozdějších fázích vývoje, kdy již je program dostatečně funkční. Nejedná se pouze o samotné testování, ale patří sem i analýza vytíženosti systému apod. V dynamickém testování se využívá testů *bílé, černé* nebo *šedé* skříňky.

1.4.2 Testování černé a bílé skříňky

Testování *černé skříňky* nebo také testování vstupů a výstupů je velmi hojně používaná strategie. Cílem testera je zjistit okolnosti, za kterých se program nebude chovat podle dané specifikace tak, že nemá přístup dovnitř programu ke zdrojovému kódu. Zná pouze vstupní údaj, který zadá jako vstup a předpokládaný výstup, dále už neřeší operace, které probíhají uvnitř programu.

Oproti tomu testování *bílé skříňky* (někdy také nazývané testování *průhledné skříňky*), jak už název napovídá, umožňuje testerovi nahlédnout do útrob programu – ke zdrojovému kódu. Díky tomu dokáže navrhnout testové případy tak, aby odhalil kritická místa programu, kde je vyšší pravděpodobnost havarování programu a pokud možno obsáhl všechny možné scénáře, které mohou nastat. Při tomto postupu existuje riziko, že tester přizpůsobí testovací scénáře činnosti programového kódu tak, že nedokáže objektivně software otestovat (Patton 2002, s. 42-43). Testování šedé skříňky je kombinací obou výše

zmíněných. V praxi to může znamenat např. situaci, kdy je software testován přes uživatelské rozhraní, a výsledky jsou ověřeny pomocí dotazů do databáze.

Každý z přístupů nalezne využití v rámci různých situací. Mezi největší výhody *černé skříňky* patří využitelnost v rámci rozsáhlých kódů a nižší nároky na testery z hlediska znalosti programovacích jazyků, oproti tomu dokáže obsáhnout pouze omezenou část programu a obtížněji se určují chybová místa. V neposlední řadě je velmi obtížná tvorba testových scénářů bez detailní specifikace k programu. Při testech *bílé skříňky* lze snáze identifikovat kritická místa programu a určit tak vhodné vstupy k efektivnímu testování. Dále pomáhá optimalizovat kód a odstranit neefektivní řádky kódu, které mohou způsobit „skryté chyby“. Oproti tomu je tento typ testování více nákladný, a to jak z hlediska časového, tak s nároky na znalosti samotných testerů. (Software Testing - Methods, c2016)

1.4.3 Testy splnění a selhání

Testování softwaru lze rozdělit také podle cíle, který je stanoven před započetím testu. V případě, že při ověřování základní funkčnosti programu není hlavním cílem nalézt co nejvíce chyb a dovést program k pádu, ale spíše otestovat minimální provozuschopnost a najít cestu, jedná se o *test splnění* (test-to-pass). Využití nalezne hlavně v raných stádiích vývoje, kdy program představuje spíše základní kostru a nejsou zdaleka ošetřeny všechny možné vstupy a procesy.

V pokročilejších fázích vývoje, ve kterých program již funguje za normálních okolností podle specifikace, se tester snaží dovézt program za hranice jeho možností, mluvíme o *testu selhání* (test-to-fail) neboli *vynucení chyb* (Patton 2002, s. 57-58).

1.4.4 Manuální a automatické testování

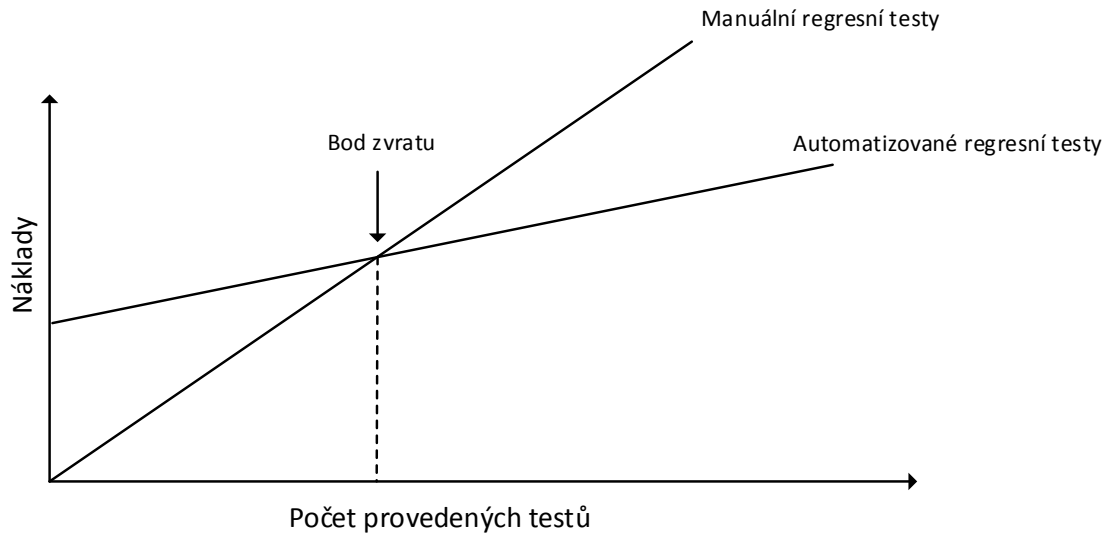
Manuální testování je prováděno bez automatických nástrojů nebo skriptů. Tester zde přebírá roli uživatele a hledá v softwaru chyby a neočekávané chování. Nenahraditelné je zejména v závěrečných fázích vývoje a při akceptačním testování u zákazníka. Nezbytnou součástí jsou zkušenosti a analytické myšlení testera. V případě testování rozsáhlých aplikací tento přístup zabírá velké množství času a je zde i vyšší riziko přehlédnutí nebo špatného vyhodnocení chyb.

Automatické testování si klade za cíl zefektivnit oblasti testování. Např. regresní testování v rámci iterativního vývoje, kdy je potřeba ověřit, zda v nových verzích programu došlo k opravě chyb, nebo zda nevznikly chyby nové. Automatické testování se nevyužívá pouze pro regresní testování, ale také pro provádění jednotkových[4], či bezpečnostních testů atd. Automatizací není možné plně nahradit manuální testování, jelikož některé oblasti se automatizací nedají pokrýt, anebo je to velmi náročné. Využití nalezne v oblastech, jako jsou různé přihlašovací formuláře nebo testování případů, kdy k softwaru přistupuje velké množství uživatelů. Dále je možné automatizaci použít na všechna grafická rozhraní, připojení k databázím apod. Hlavní výhodou automatického testování je časová úspora a s tím spojené snížení nákladů. Mezi nevýhody patří nutnost kontroly a úpravy skriptů během vývoje. V iterativním vývoji se nezřídka stává, že po naimplementování určité funkcionality dojde vývojový tým k závěru, že plně nevyhovuje požadavkům a je nutné ji přepracovat, nebo úplně vypustit. Další oblast, kde dochází často k změnám, je grafické rozhraní.

K automatickému regresnímu testování by se mělo přistoupit, hlavně pokud se jedná o rozsáhlé projekty v pozdějších fázích vývoje, kdy je software stabilní a nedochází k výrazným změnám. V rámci automatického testování je využíváno pomocných skriptovacích jazyků, jako je Python nebo Ruby. K vyšší efektivnosti se využívají pomocné nástroje jako je Visual Studio Test Professional nebo IBM Rational Functional Tester, díky kterým je možnost výrazně zrychlit a zjednodušit testování programu. Rozhodnutí o využívání automatického regresního testování by mělo být podloženo jeho finanční návratností. Náklady na testování v závislosti na časové náročnosti,

[4] Jednotkové testy (Unit tests) – z pohledu objektového programování se jedná o testování jednotlivých metod, či tříd. Zpravidla je prováděno samotným vývojářem.

reprezentované počtem provedených testů, jsou ilustrovány na obrázku níže. Je zde vidět finanční návratnost automatického regresního testování v delším časovém horizontu, kdy stoupá počet opakovaně prováděných testů.



Obr. 4: Návratnost automatizovaných regresních testů

Zdroj: Vlastní

2 Metodiky vývoje softwaru

Metodiky vývoje softwaru představují souhrn metod a postupů, které udávají rámec pro vývoj softwaru. Metodika vývoje IS/ICT podle Buchalcevová (2009, s. 17) *„definuje principy, procesy, praktiky, role, techniky, nástroje a produkty používané při vývoji, údržbě a provozu informačního systému, a to jak z hlediska softwarově inženýrského, tak z hlediska řízení.“*

V současné době se metodiky pro vývoj softwaru dělí na dva hlavní proudy označované jako rigorózní a agilní metodiky. Hlavním kritériem pro rozdělení je kritérium *Váha metodiky*, ale liší se i jinými kritérii jako *Přístup k řešení* atd.

V této kapitole jsou porovnány rigorózní a agilní metodiky vývoje a jejich historie. Větší důraz je kladen na agilní metodiky, které se neustále vyvíjejí, a jejich potenciál neustále roste. Blíže budou popsány nejznámější metodiky jak rigorózního, tak agilní vývoje, a pravidla Agilního manifestu, která představují základ agilního vývoje softwaru.

2.1 Rigorózní metodiky

Rigorózní, nebo také tradiční metodiky, se vyznačují přesným a podrobným popsáním procesů a činností. Vycházejí z předpokladu, že veškeré požadavky je možno naplánovat a definovat předem, a tím zabránit případným změnám v budoucnosti, ke kterým by v ideálním případě nemělo vůbec dojít. Rigorózní metodiky se využívají ve velkých programátorských týmech při tvorbě velkých projektů, nejčastěji založených na vodopádovém (sériovém) modelu, kdy lze z modelu jasně pochopit, ve které fázi produktového cyklu se program nachází, a tím včas odhalit případné problémy, které mohou nastat, a tím zajistit dodání produktu včas. Existují ovšem i rigorózní metodiky založené na iterativním a inkrementálním modelu. Mezi nejznámější rigorózní metodiky patří Object–oriented Process, Environment and Notation (OPEN), Rational Unified Process (RUP), či Enterprise Unified Process (EUP).

2.1.1 Metodika Rational Unified Process

Metodika Rational Unified Process vznikla „*spojením přístupu **Rational** a metodiky **Objectory Process** Ivara Jacobsona*“ (Buchalcevová, 2009, s. 121). Následně byla vytvořena zobecněná verze *Unified Process* popsaná v knize *„The Unified Software Development Process“*. Původně byla řazena mezi rigorózní metodiky, ale postupně byla doplněna o agilní praktiky a v současnosti je velmi univerzální a lze využít pro jakýkoliv typ projektů. Metodika RUP je založena na 6 praktikách softwarového vývoje (RATIONAL SOFTWARE CORPORATION, 2011).

- Iterativní vývoj softwaru *(develop software iteratively)*
- řízení požadavků *(manage requirements)*
- použití komponentových architektur *(use component-based architectures)*
- vizuální modelování softwaru *(visually model software)*
- ověřování kvality softwaru *(verify software quality)*
- řízení změn v softwaru *(control changes to software).*

Metodika RUP je přehledně charakterizována v (Buchalcevová, 2009, 121 – 127).

2.2 Agilní metodiky

Díky rychlému pokroku v technologiích a změnám požadavků okolního prostředí již tradiční metodiky nevyhovovaly k plnění požadavků. Požadavky na rychlé zavedení informačních systémů a komunikačních technologií (IS/ICT) a nutnost rychlé reakce na okolní trh vyústily ke změnám v metodikách vývoje. Z tohoto důvodu se od druhé poloviny 90. let začínají prosazovat metodiky, které umožňují rychlou tvorbu řešení a možnost pružné reakce na měnící se požadavky. Tyto metodiky se nazývají „agilní“, což znamená „hbitý“, „aktivní“ nebo „horlivý“. Na rozdíl od rigorózních přístupů, kdy je zákazník zapojen jen na počátku vývoje při vytváření specifikací, agilní metodiky si zakládají na zapojení zákazníka během celého vývoje. Z pravidla je zákazníkovi co

nejrychleji předložen program (nebo jeho část) již během vývoje a na základě zpětné vazby je možné provést úpravy během vývoje. Tento postup je velmi výhodný, protože zákazník nejlépe posoudí, co opravdu chce, a náklady na úpravy jsou znatelně nižší během vývoje, než po úplném dokončení produktu (viz kap. 1.3 Účel testování).

Pod pojmem „agilní“ se skrývá velké množství odlišných přístupů, které však mají společný základ v tzv. Agilním manifestu *(Manifesto for Agile Software Development).* S jeho vznikem je spojeno 17 softwarových vývojářů, kteří v roce 2001 vytvořili a podepsali základní teze agilního vývoje a vytvořili „Alianci pro agilní vývoj softwaru“. Mezi nejznámější agilní metodiky patří Scrum, Extrémní programování (XP) nebo vývoj řízený vlastnostmi (FDD), či vývoj řízený testy (TDD).

2.2.1 Manifest agilního vývoje software

„Manifest agilního vývoje software“ definuje hodnoty a principy agilního vývoje softwaru, vycházejících z reálných zkušeností autorů.

„Objevujeme lepší způsoby vývoje software tím, že jej tvoříme a pomáháme při jeho tvorbě ostatním. Při této práci jsme dospěli k těmto hodnotám:

- ***Jednotlivci a interakce*** *před procesy a nástroji*
- ***Fungující software*** *před vyčerpávající dokumentací*
- ***Spolupráce se zákazníkem*** *před vyjednáváním o smlouvě*
- ***Reagování na změny*** *před dodržováním plánu“* (Beck, 2001).

Na tyto teze navazuje 12 principů agilního vývoje softwaru, které stojí za Agilním manifestem:

1. Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.

3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.

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

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

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

7. Hlavním měřítkem pokroku je fungující software.

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

9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.

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

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

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

Tyto principy slouží jako základ většiny agilních metodik, které značně usnadní proces vývoje (Beck, 2001).

Mezi nejdůležitější faktory agilního vývoje patří *sebe-organizující se tým*, využívající *iterativní a inkrementální vývoj* spojený s *vysokou kvalitou softwaru*. Neméně důležitá je i spoluúčast zákazníka během celého procesu vývoje, ne jen na počátku jako tomu je u tradičních metodik.

2.2.2 Scrum

Scrum byl původně používán pro potřeby vývoje softwaru, ale dnes se s ním můžeme setkat také v mnoha jiných odvětvích produkt managementu, vzdělání, marketingu a dalších. V dnešní době patří k jedné z nejpoužívanějších a nejoblíbenějších agilních metodik využívaných v projektovém řízení. Poprvé byl představen v roce 1986 v *New Product Development Game* pány Hirotaka Takeuchi a Ikujiro Nonaka, o další rozšíření se zasloužili nejvíce pánové Ken Schwaber a Jeff Sutherland, kteří ze svých zkušeností sepsali *The Scrum Guide*. Název Scrum byl odvozen od rugbyového mlýna. Stejně jako v rugby, Scrum vyžaduje spolupráci různorodého týmu, který se po fázích snaží společně dostat za cílovou čáru. Scrum tým je samo-organizující se jednotkou, která má všechny kompetence k dosáhnutí výsledků bez zásahů zvenčí. Týmový model je navrhnut tak, aby optimalizoval flexibilitu, kreativitu a produktivitu.

Scrum, jako ostatní agilní metodiky, je postaven na iterativním přístupu. Na začátku vývoje vytvoří tzv. *product owner* uspořádaný seznam požadavků seřazený podle priorit k dokončení. Tento seznam se nazývá ***product backlog***.

Druhý krok začíná plánováním a výběrem požadavku s nejvyšší prioritou z product backlogu, kterým se v následující iteraci, nazývané ***Sprint*** a trvajícím většinou 2 – 4 týdny, bude tým zabývat. Požadavek je umístěn na tzv. „*TO DO list"*, kde je dále rozepsán na jednotlivé pod úkoly. Během sprintu probíhají každodenní porady celého týmu trvajících 15 minut, která slouží ke koordinaci a posouzení prací - tzv. *daily Scrum*. Tyto porady jsou řízeny *scrum masterem*, jehož hlavním úkolem, je udržet tým v plném soustředění na cíle sprintu. Na poradách účastníci odpovídají na tři hlavní otázky:

- Na čem jsem včera pracoval?
- Jaké mám úkoly na dnešní den?
- Jaká vidím omezení nebo překážky, které mi brání ve splnění úkolu?

Na konci této fáze by měla být již vybraná část produktu ve stavu schopném distribuce na trh. Nastává fáze přezkoumání sprintu, kdy za účasti všech zúčastněných stran, včetně zákazníků, je provedeno zhodnocení daného sprintu. Dochází k prezentaci odvedené

práce, případně problémů, které nastaly a zapříčinily nesplnění některých požadavků z product backlogu. Je diskutován také plán pro budoucí vývoj, rozpočet, či změny v obchodní strategii. Na konci dochází k úpravě product backlogu a následnému určení požadavků pro následující sprint tak, aby splnil nová očekávání.

Poslední fází je příležitost pro retrospektivní ohlédnutí za předchozím sprintem a zhodnocení jeho průběhu. Jsou identifikovány dobré praktiky, ale také slabá místa, a následně je vytvořen plán pro vylepšení slabých míst a celkové zlepšení průběhu dalšího sprintu. Cílem pro nový sprint je zlepšení vyvíjeného produktu a zvýšení efektivity scrum týmu. Po dokončení cyklu je opět vybrán další požadavek z product backlogu a nastává nový sprint. Produkt je tedy vyvíjen v iteracích s pravidelnými inkrementy. (Proces je znázorněn na obrázku „Fáze Scrumu“)

V rámci scrum týmu existují 3 základní role:

- **Product owner:** je zodpovědný za správu product backlogu a práci vývojářského týmu.
- **Development team:** samo-organizující se tým, který vytváří produkt.
- **Scrum master:** slouží jako řešitel problémů a přechodný můstek mezi vlastníkem produktu a vývojovým týmem. Pomáhá týmu k dosažení vytyčených cílů (Schwaber, 2013).

Obr. 5: Fáze Scrumu
Zdroj: www.scrumalliance.org

2.3 Modely životního cyklu softwaru

„Modely životního cyklu informačního systému představují rámec realizace procesů životního cyklu v časové posloupnosti.“ (Buchalcevová, 2009, s. 47) V současné době se při budování IS/ICT využívá hned několik modelů životního cyklu. Výběr správného modelu životního cyklu je významným kritériem pro zvolení optimální metodiky. V této kapitole jsou představeny některé využívané modely životního cyklu a popsány jejich výhody a nevýhody.

2.3.1 Model programuj a opravuj

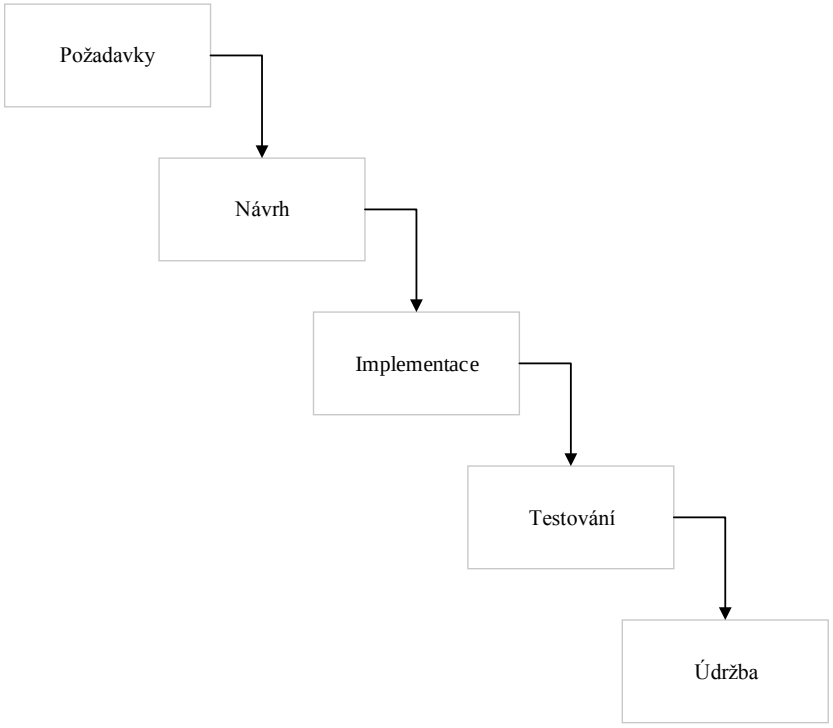
Model *programuj a opravuj* je nejjednodušším z modelů, který představuje jednoduchý a neformální způsob vývoje. Základem je neustálé opakování činností: programování, testování a oprava chyb. Tento model se vyznačuje velmi nízkou mírou plánování a dokumentace, kdy je dána přednost rychle vytvořit verzi programu, která je následně testována, a nalezené chyby jsou reportovány zpět vývojovému týmu. Pokud je rozhodnuto, že množství a úroveň chyb je na přijatelné úrovni, anebo je dosažen termín ukončení projektu, je vývoj ukončen a produkt uveden do provozu.

2.3.2 Vodopádový model

Vodopádový model patří mezi nejznámější zástupce tzv. tradičních přístupů k vývoji softwaru. Poprvé byl formálně popsán v článku *Managing the Development of Large Software Systems*, publikovaném v roce 1970 Winstonem W. Roycem.

Model je založen na postupném přechodu mezi fázemi projektu, kdy konec jedné fáze představuje zároveň začátek fáze následující. V rámci vodopádu není možný posun zpět, a proto je vždy postup na další fázi důkladně zhodnocen. Jak je z obrázku „Vodopádový model“ patrné, k samotnému testování funkcionality se přistupuje až po kompletní implementaci systému. V dnešní době není tento model mezi vývojáři příliš oblíben, a to hlavně z důvodu své nepružnosti. Avšak v rámci disciplinovaného vývojářského týmu, který klade velký důraz na specifikaci produktu na začátku vývoje, se stává velmi efektivním nástrojem.

Nevýhodou ovšem zůstává časová náročnost přípravy projektu a s tím spojené nároky kladené na členy vývojového týmu.



Obr. 6: Vodopádový model

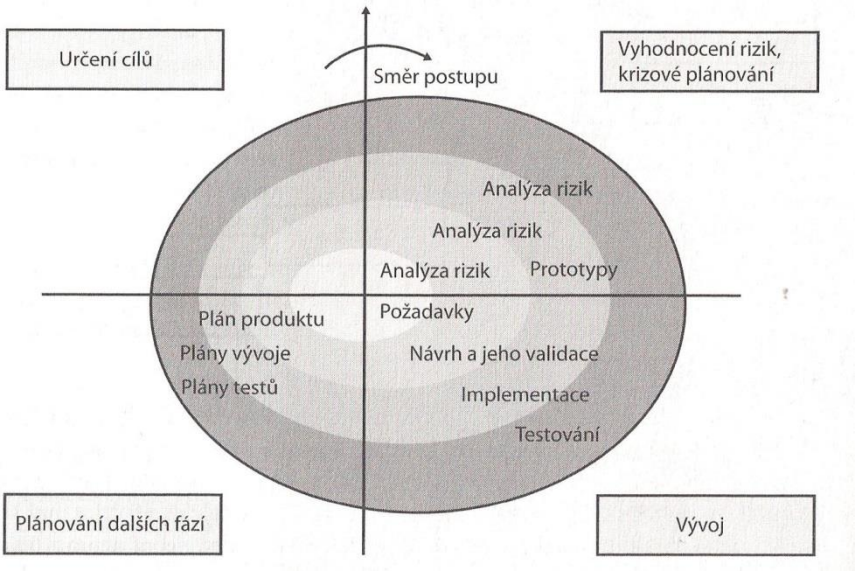
Zdroj: Vlastní

2.3.3 Spirálový model

Spirálový model byl navržen v roce 1986 Barry Boehmem jako vylepšení nevýhod vodopádového modelu. Základní myšlenkou, na které je spirálový model postaven, je uvolnění striktních pravidel vývoje vodopádového modelu. Projekt je často vytvářen tak, že na začátku je naplánován prototyp, nebo značně zjednodušená verze produktu. Ten je v následujících iteracích vyhodnocován. U některých produktů se využívá i zpětná vazba od zákazníka, kdy je produkt vypuštěn formou veřejných beta verzí.

Hlavní výhodou je iterativní proces obsahující 4 hlavní fáze:

- **Určení cílů:** dochází k určení cílů pro současnou fázi projektu.
- **Vyhodnocení rizik:** rozpoznání největších rizik a hledání cest k jejich omezení.
- **Vývoj:** probíhá specifikace požadavků, návrh, vývoj a testování.
- **Plánování:** vyhodnocení projektu a počátek plánování přechodu na další cyklus spirály.



Obr. 7: Spirálový model

Zdroj: Page (2009, s. 64)

2.3.4 Inkrementální model

Inkrementální (přírůstkový) model dělí systém na samostatně realizované části (přírůstky). Každý přírůstek prochází jednotlivými fázemi vývoje a postupně je začleňován do celého softwarového produktu.

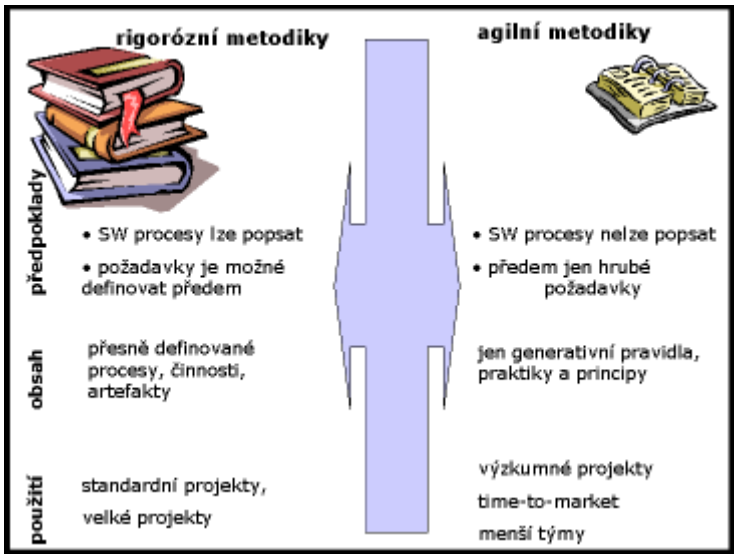
2.3.5 Evoluční model

V evolučním (iterativním) modelu jsou vyvíjeny jednotlivé verze produktu (iterace) bez úvodní důkladné specifikace, ale pouze s pomocí hrubých požadavků, které se dále zpřesňují v dalších iteracích. Mezi největší výhody patří možnost včasné zpětné vazby od zákazníka a velká flexibilita pro realizování změn díky rozdělení vývoje do jednotlivých iterací. Mezi nevýhody patří špatně odhadnutelná délka a rozsah řešení.

2.4 Porovnání rigorózních a agilních metodik

V dnešní době existují dvě hlavní skupiny metodik. Rigorózní metodiky vycházejí převážně z vodopádového modelu životního cyklu softwaru. Díky špatným možnostem reakce na případné změny jsou považovány za *těžké* metodiky. Oproti tomu agilní metodiky, využívající iterativních a inkrementálního modelů se zapojením zákazníka do celého procesu vývoje, jsou schopny mnohem rychleji a flexibilně reagovat na měnící se prostředí a podmínky jsou považovány za lehké metodiky

Obě metodiky vycházejí z rozdílného pohledu na tvorbu softwaru a své využití naleznou v různých typech projektů. Hlavní rozdíly jsou zobrazeny na Obr. 7: Srovnání rigorózních a agilních metodik.



Obr. 8: Srovnání rigorózních a agilních metodik

Zdroj: Buchalcevová (2009, s. 70)

Rigorózní metodiky jsou vhodné pro velké projekty realizované ve velkých týmem s podrobně připravenými specifikacemi a postupy. Oproti tomu agilní metodiky se nejlépe uplatní v prostředí menších týmů.

Rozdílné jsou i cíle celého projektu, kdy se rigorózní metodiky snaží držet daných požadavků specifikovaných na začátku projektu a zaměřují se na kvalitu procesů s předpokladem, že kvalitní procesy povedou kvalitnímu výsledku. Agilní metodiky na rozdíl od rigorózních staví na předpokladu, že zákazník sám nejlépe ví, co chce. Dále počítají s rychle se měnícím prostředím, a proto nekladou důraz na podrobnou tvorbu specifikací a obsáhlé dokumentace. Jejich hlavním zaměřením je aktuální hodnota pro zákazníka.

V posledních letech můžeme pozorovat velký nárůst firem využívajících agilní metodiky, kdy s ohledem na každoroční dotazníkové šetření „*State of Agile",* které zpracovává společnost Version One Ltd., můžeme říci, že naprostá většina organizací v nějaké míře využívá agilní praktiky, kdy více než polovina všech týmů je agilních. Z výzkumu také vyplývá, že nejvíce využívanou metodikou je Scrum, nebo metodiky od něj odvozené (VersionOne, c2013, c2014, c2015).

3 Možnosti Softwarové podpory v rámci stavebnictví

Využívání softwaru pro zefektivnění firemních procesů je v dnešní době nedílnou a mnohdy i nezbytnou součástí naprosté většiny velkých a středních podniků a nemalé části malých podniků. Tento vzrůstající trend je logickým vyústěním technologického pokroku. Softwarové nástroje lze rozdělit na dvě hlavní skupiny. První skupinou jsou komplexní podnikové informační systémy pro řízení podniku (ERP - Enterprise Resource Planning) jako SAP, K2 nebo KARAT software. Druhou skupinou jsou poté „samostatné“ softwarové nástroje, podporující konkrétní oborově zaměřené činnosti.

3.1 Informační systémy ve stavebnictví

Stavebnictví patří z hlediska informačních systému mezi velmi specifické obory. Oproti jiným odvětvím, jako například strojírenství, kde jsou IS nedílnou součástí výrobních podniků, je ve stavebnictví hned několik faktorů, které značně ztěžují využívání komplexních IS. Stavební výroba je podstatně složitější v ohledu plánování vstupů a pracovních procesů. Stavební výroba se neodehrává vždy na stejném místě a za stejných podmínek, a proto kromě specifických podmínek pro každou stavbu dochází i k častým změnám samotných nákladů během realizace stavby, které jsou ovlivněny mnoha okolnostmi, jako je umístění stavby, geopolitické podmínky, počasí, cena zdrojů nebo výběr subdodavatelů atd. Z tohoto důvodu je běžnou praxí improvizace a schopnost rychlé reakce na měnící se podmínky před nastavením pevných pravidel.

Některé firmy využívají své ERP systémy pouze pro některé z agend, jako účetnictví, řízení vztahů se zákazníky (CRM - Customer Relationship Management), či pro správu dokumentů (DMS - Document Management System). Tyto moduly sice pomáhají se zpracováním ekonomických dat a celkovým zefektivněním komunikace, ale neřeší všechna specifika dodávek. A proto je často využíváno komplexních IS vyvinutých speciálně pro stavebnictví jako ERP systém INFOpower od společnosti RTS, a.s. nebo kombinaci obecných ERP systémů s vlastními moduly a softwarů určených pro plánování a realizaci

staveb, které lze často propojit s používaným informačním systémem. (Novotný, 2012), (Pech, 2013)

3.2 Samostatné softwarové nástroje ve stavebnictví

Ve stavebnictví je využívána i široká škála specifických programů pro zefektivnění práce. Díky mnoha specifickým činnostem spojených se stavebnictvím je zde zastoupeno široké spektrum programů pohybujících se od CAD (Computer-aided design) programů typu AutoCad, či ArchiCad přes specifické programy využívané pro výpočty a statické analýzy různých typů konstrukcí až po rozpočtové programy sloužící k přípravě rozpočtů pro výběrová řízení a následnému sledování stavby ve fázi realizace po její dokončení. Mezi nejvíce používané rozpočtové programy v ČR lze zařadit KROS od společnosti ÚRS PRAHA, a.s., RTS stavitel+ od společnosti RTS, a.s. a ASPE od společnosti IBR Consulting s.r.o.

3.3 Nové trendy ve stavebnictví

Informační technologie zažívají nepřetržitý vývoj již několik desetiletí a současnost není výjimkou. V rámci stavebnictví se objevuje nezanedbatelný trend minulých let, a to orientace na webová řešení. S tím je spojen i větší důraz na rozvíjení BI (Business intelligence) jako rozšíření stávajících ERP systémů. Manažerské informační systémy značně usnadňují a zlepšují získávání podstatných informací a uceleného přehledu o aktuálním dění ve firmě, čímž poskytují manažerům silný nástroj pro efektivní řízení a plánování. Podrobněji tuto problematiku popisuje Hajn (2014).

4 Vývoj a testování aplikace

Tato kapitola je zaměřena na samotný proces vývoje a testování stavebního softwaru Aspe 10, na kterém jsem se jako člen vývojového týmu v rámci roční řízené praxe podílel. Mým hlavním úkolem bylo testování vyvíjeného řešení, reportování nalezených chyb a zajištění jejich opravy. V rámci vývojového týmu jsem se dále podílel na vytváření návrhu některých funkcionalit a tvorbě nápovědy k programu.

Aspe je komerčním softwarem vyvíjeným firmou IBR, Consulting s.r.o., která vznikla v roce 2007 jako dceřiná firma společnosti VALBEK-EU, a.s. Její hlavní činností je vývoj vlastních softwarových řešení ve stavebnictví. Hlavní část vývoje je soustředěna na vývoj desktopové aplikace Aspe. Dále je vyvíjen webový klient k desktopové aplikaci Aspe online, Manažerský informační portál (MIP) k přehlednému sledování plnění a financování zakázek v čase, který je nově vyčleněn jako samostatné středisko BI. Dalším produktem je Esticon, který představuje webové řešení pro oceňování staveb v různých fázích vývoje. Všechny tyto činnosti jsou zastřešeny pod divizí SW. Mimo tuto divizi jsou ve firmě zastoupeny i další divize působící ve stavebnictví, které tak vývojářům poskytují praktický základ pro návrh a vývoj.

4.1 Požadavky na aplikaci

Od vydání Aspe 9 na trh uběhlo již 5 let. Přestože docházelo k pravidelným updatům funkcionalit, celkový vzhled a UI působily v dnešní době značně nemoderně a nevyhovovaly tak představám a obchodním cílům vedení. Obchodní model Aspe je postaven na dvou pilířích, jedním z nich je vývoj nových funkcionalit a jejich implementace. Druhý pilíř tvoří placená technická podpora k produktům zastřešující kompletní uživatelský servis k produktům. Obchodní cíle firmy počítají s rozšířením programu Aspe mezi nové zákazníky a tím zvýšení podílu v rámci trhu. S tím je spojený i nárůst podílu technické podpory k programu na celkových tržbách střediska, které již teď patří k nejdůležitějšímu zdroji příjmu.

Z výše zmíněného důvodu bylo rozhodnuto, že stávající verze Aspe 9 již nevyhovuje představám a cílům podniku, a proto vznikl požadavek na vývoj zcela nové verze

programu, která nahradí stávající Aspe 9. Nová verze bude přebírat veškeré funkcionality z předchozí verze s tím, že v rámci verze 10 budou navíc zahrnuty nové požadavky zákazníků, které byly zadány v pozdějších fázích vývoje, a nebudou tak již přidány do Aspe 9. Nový produkt by měl uživatelům přinést větší komfort díky modernizaci uživatelského rozhraní, které za poslední roky dostálo jen minimálních změn. Dále bude přepracována práce se soubory, a to zejména import a export souborů MS Excel. Na začátku vývoje byly stanoveny pouze obecné cíle, jaké by měl výsledný produkt splňovat. Konkrétní specifikace k jejich dosažení byly vytvářeny v průběhu vývoje.

4.2 Vývojový tým

Na vývoji nové verze Aspe se nějakou měrou podíleli, či podílí všichni zaměstnanci střediska Aspe. V rámci střediska jsou zaměstnanci rozděleni do dvou skupin, a to na pracovníky vývoje SW, pod které spadají programátoři, analytici, a pracovníky technické podpory, kteří mají na starost společně s obchodním oddělením komunikaci se zákazníky. Z hlediska vývoje softwaru zastávají členové týmu různé role:

- **Analýza:** Hlavní činností je rozpracování návrhů nových funkčností a analýza chyb zapsaných v bug reportu. Následně vytváří tzv. „Úpravy“ *(týmem používaný termín pro zadání pro programátory),* které jsou přiřazeny programátorům. Po dokončení úpravy programátorem provádí kontrolu a v případě kladného výsledku předá úpravu k zevrubnému otestování a schválení.

- **Programátoři:** Implementace samotných úprav a otestování jejich funkčnosti.

- **Testeři:** Do této kategorie spadají ostatní pracovníci střediska Aspe. Z většiny se jedná o pracovníky technické podpory, kteří se různou měrou podílejí na testování programu, vytváření školící dokumentace k programu nebo tvorbě nápovědy. Samotné testery lze rozdělit na dvě skupiny z hlediska stylu testování. U většiny z nich se jedná pouze o doplňkovou činnost k hlavní náplni práce, kterou je zajišťování technické podpory zákazníkům využívajících program Aspe, a spočívá v simulaci reálné práce s programem a zapisování nestandartního chování. Tato činnost ovšem není nijak pravidelná. Druhou skupinou jsou pracovníci, kteří již systematicky testují a schvalují jednotlivé úpravy od analýzy a aktivně vyhledávají

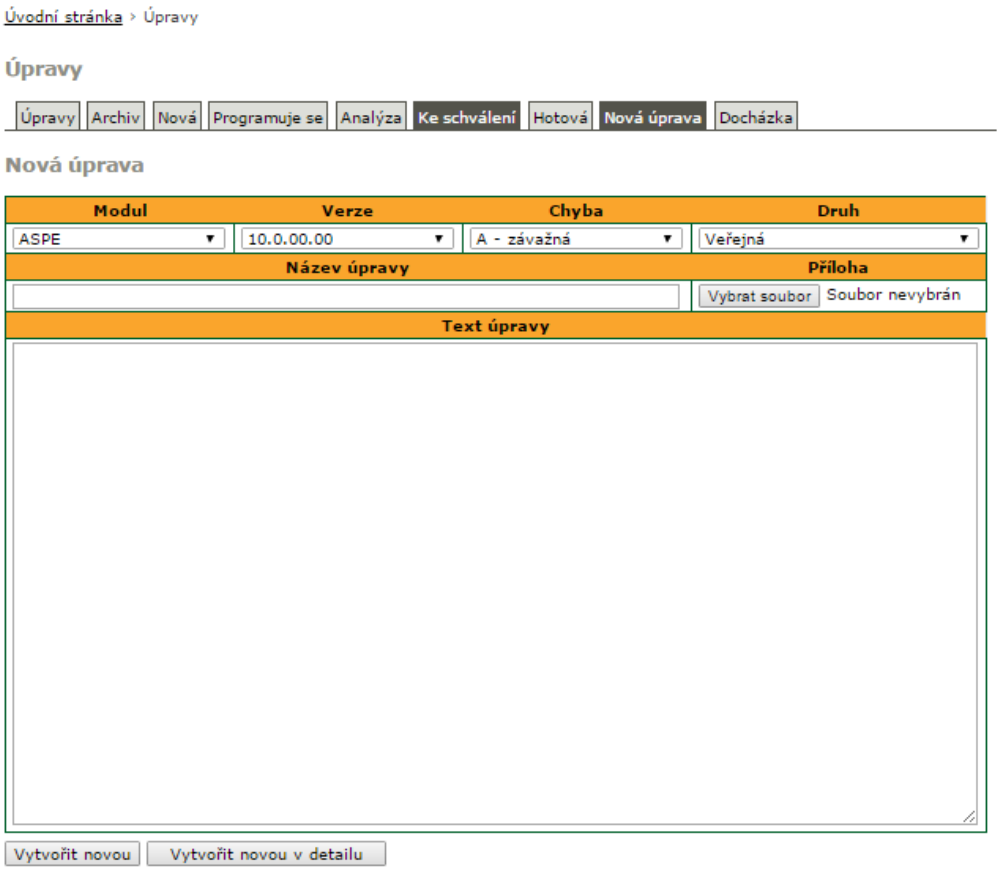
nové chyby, které poté ve spolupráci s analýzou zadávají jako úpravy. Ty jsou následně předány programátorům.

V některých případech splývá rozdělení na analytiky a testery, kdy i analýza provádí testy a někteří testeři vytváří nové úpravy. Ovšem každý návrh od testerů by měl být překontrolován analýzou, aby nedocházelo k nesystémovým úpravám, a tím plýtvaní zdroji. Stejně tak i každá opravená chyba, či nová úprava by měla být otestována testery, aby byla zajištěna verifikace příslušné úpravy.

4.3 Systém na evidenci úprav

Ke správě úprav je využíváno webové řešení na zakázku od firmy DYNWEB, jejíž služby společnost využívá i ve formě samotných webových stránek, či vlastního intranetu. Samotný systém je po přihlášení přístupný přímo z webových stránek. Součástí celého systému je i evidence připomínek od zákazníků a docházkový systém zaznamenávající čas strávený programátory nad konkrétní úpravou.

Systém je rozdělen na několik záložek představujících konkrétní fázi vývoje dané úpravy. Jednotlivé záložky jsou dále vybaveny filtry pro rychlejší orientaci v systému. V záložce **Nová úprava** (Obr. 9: Vytvoření nové úpravy) probíhá vytváření nových úprav. Do pole Text úpravy je vložen bug report (kap. 1.2.5) s nalezenou chybou, případně popis nové funkcionality k naprogramování. V horní části formuláře je poté vybrán modul, kterého se daná úprava týká, aktuální verze, vybrána závažnost chyby, název úpravy a případně přiložena příloha.



Obr. 9: Vytvoření nové úpravy

Zdroj: Vlastní

Po vytvoření úpravy je úprava přesunuta pod záložku **Analýza**, kde je případně doplněna a dále přiřazena konkrétnímu programátorovi. Na záložce **Programuje se** jsou evidovány rozpracované úpravy s ukazatelem stavu. Po dokončení úpravy programátor předá úpravu zpět analýze, která úpravu buď přesune do záložky **Ke schválení** k zevrubnému otestování a schválení, nebo vrátí zpět programátorovi. Testeři si následně vybírají ze záložky **Ke schválení** jednotlivé úpravy, které testují. V případě kladného výsledku úpravu schválí, a tím ji přesunou do záložky Hotová, nebo úpravu neschválí a vrátí ji tak konkrétnímu programátorovi do záložky **Programuje se** k dopracování. Ke každé testované úpravě je přiloženo vyjádření, které obsahuje jméno testera a verzi programu, na které bylo řešení testováno. V případě potřeby jsou zde popsány konkrétní kroky během testování a nalezené nesrovnalosti z hlediska zadání. V případě nalezení chyb, které nejsou v zadání

úpravy, je po konzultaci s analýzou rozhodnuto, zda bude vytvořena nová úprava, nebo doplněna stávající.

Tento systém úprav je postaven na vysoké míře komunikace napříč celým vývojovým týmem, protože mnohdy je potřeba, aby tester měl úplné informace o rozsahu dané úpravy, a nevracel tak zbytečně úpravu k přepracování programátorovi. Problém by mohl být částečně vyřešen, pokud by se neschválené úpravy vracely analýze místo konkrétnímu programátorovi, ale otázkou zůstává, zda by nedošlo k výraznému zpomalení vývoje.

4.4 Vývoj aplikace

V rámci vývoje bylo využíváno agilního přístupu, kdy byl kladen důraz na pravidelné porady a osobní komunikaci v rámci týmu v průběhu dne. Během vývoje byl program prezentován a konzultován s některými zákazníky, od kterých byla získávána zpětná vazba pro případné změny. V rámci vývoje bylo využíváno převážně evolučního modelu, kdy UI a dílčí funckionality byly vypracovávány na základě návrhu vývojového týmu, či požadavků zákazníků, a následně implementovány v rámci nepravidelných iterací do nových vývojových verzí programu. Tyto verze byly následně testovány a připomínkovány.

4.4.1 Představení programu Aspe

Aspe je desktopový rozpočtový software pro stavebnictví s dlouhou tradicí. Na českém trhu se pohybuje již od roku 1990. Z počátku byl určen hlavně pro tvorbu nabídkových rozpočtů. V současné době nabízí možnost kompletního sledování zakázky od tvorby nabídkových rozpočtů přes období realizace zakázky, kde dochází k čerpání rozpočtu a následné fakturaci, až po její dokončení. Poskytuje tak jednotnou základnu pro komunikaci mezi projektantem, investorem a dodavatelem stavby.

V současné době je mezi zákazníky na trhu Aspe 9 ve verzi 9.9.0.30[5], které bude nahrazeno nově vydaným Aspe 10 ve verzi 10.0.0.0. Od rozhodnutí o vývoji Aspe 10 běžel

[5] Číslování verzí je rozděleno na 4 části, kdy poslední část je věnována verzím Aspe 9, kdežto třetí část je používána v rámci vývojových verzí v rámci vývoje Aspe 10.

souběžný vývoj obou verzí Aspe, jak distribuovaného Aspe 9, tak nově vyvíjeného Aspe 10, které ale bylo vedeno v rámci vývoje jako 9.9.x.0[6].

4.4.2 Průběh vývoje Aspe 10

Na návrhu nové podoby Aspe se podíleli vybraní členové vývojového týmu, kteří zastávají různé role v rámci firmy. Zpravidla vznikal jako výsledek porad v rámci členů vedení, obchodního oddělení, analýzy a technické podpory, kteří s programem denně pracují a mají kontakt se zákazníky. Některé funkcionality byly ale navrženy přímo jako požadavek konkrétního zaměstnance. Následně byly požadavky analyzovány a zadány do vývoje. Během vývoje došlo k několika zásadním i menším změnám v návrhu, které měly různé příčiny, a měly tak za následek prodloužení předpokládané doby vývoje. Mezi hlavní důvody lze započítat požadavky na nové funkcionality pocházející jak od zákazníků, tak od vývojového týmu, technické problémy spojené s přepisem kódu starších verzí Aspe a v neposlední řadě rozhodnutím, učiněném během vývoje, o rozdělení původního modulu REALIZACE na tři samostatné moduly (ZBV[7], ČERPÁNÍ[8], FAKTURY[9]).

V současné době již vývoj probíhá druhým rokem, přibližně po roce vývoje byla přístupná první „testovatelná“ verze, což byla verze ve které již bylo možné provádět první dynamické testování pomocí simulace reálné práce s programem. Před touto verzí byly možnosti dynamického testování značně omezeny, testování se soustředilo pouze na jednotlivé funkcionality v rámci jednotlivých knihoven. V období mého nástupu se program nacházel ve vývojové verzi 9.9.20.0, kde se první 2 moduly (ASPE a STAVBA) nacházely ve stavu, kdy již hlavní funkcionality byly hotovy. Moduly spadající do realizace stavby (ZBV, ČERPÁNÍ a FAKTURY) byly spustitelné, ale stále postrádaly některé klíčové funkcionality. Dále program obsahoval další funkční moduly podporující práci s programem, jako Číselníky (obsahující databáze údajů, vazeb a vzorců používaných v Aspe), Cenové soustavy (modul pro správu ceníků stavební produkce), či modul Administrátor pro správu uživatelů a tvorbu nových firemních databází. Aspe obsahuje i další moduly. Kromě modulu NABÍDKY, který slouží k sehrání nabídek a jejich

[6] V rámci vývoje bylo vytvořeno 46 verzí, před vydáním finální distribuční verze 10.0.0.0
[7] Modul ZBV – Změny během výstavby slouží k práci se změnami rozpočtu během realizace zakázky.
[8] Modul ČERPÁNÍ – slouží k tvorbě zjišťovacích protokolů, pomocí nichž je čerpán rozpočet stavby.
[9] Modul FAKTURY – slouží k ruční tvorbě faktur, či tvorbě faktur připravených z podkladů ze zjišťovacích protokolů.

vyhodnocení v rámci soutěže, již nejsou tak zásadní pro práci s programem a v této verzi byly buď zcela nepřístupné, nebo v takové podobě, že bylo zbytečné provádět jakékoliv testy.

Modul ASPE představuje úvodní modul s přehledem firemní struktury a jednotlivých staveb zařazených do závodů, či provozních jednotek, které mají jednotlivé zakázky na starost. Druhým modulem je STAVBA, která slouží k tvorbě rozpočtů stavby, což představuje rozdělení stavby na jednotlivé stavební objekty obsahující stavební díly, pod které spadají jednotlivé položky rozpočtu. V tomto období program obsahoval velké množství chyb, které způsobovaly nespočet pádů.

Přibližně s verzí 9.9.30.0 již bylo možné s programem v omezené míře pracovat i v modulech ZBV, REALIZACE a FAKTURY. S verzí 9.9.41.0 byl dopracován i modul NABÍDKY a vývoj se soustředil na dokončení dílčích funkcionalit, ladění grafického rozhraní a nastavení licencí a práv pro uživatele. U zbývajících modulů SR (Souhrnný rozpočet), MONITORING, HMG (Harmonogram) a VP (Výrobní plán) bylo rozhodnuto, že se opraví pouze základní chyby a funkčnost zůstane na stejné úrovni jako v předcházející verzi Aspe 9.

V současné době je již vytvořena distribuční verze 10.0.0.0, na které jsou laděny poslední detaily. Tato verze je určena pro všechny zákazníky, kterým bude rozeslána v rámci uzavřených smluv o podpoře. Mezitím však již probíhá akceptační testování u jednoho ze zákazníků s vlastními požadavky, které by mělo být ukončeno přibližně měsíc po rozeslání distribuční verze.

Tím však proces vývoje nové verze nekončí. Na samotném programu je potřeba odstranit ještě několik známých „chyb“ a zapracovat všechny připomínky z akceptačních testů, které budou obsaženy ve verzi 10.0.1.0.

4.5 Testování aplikace

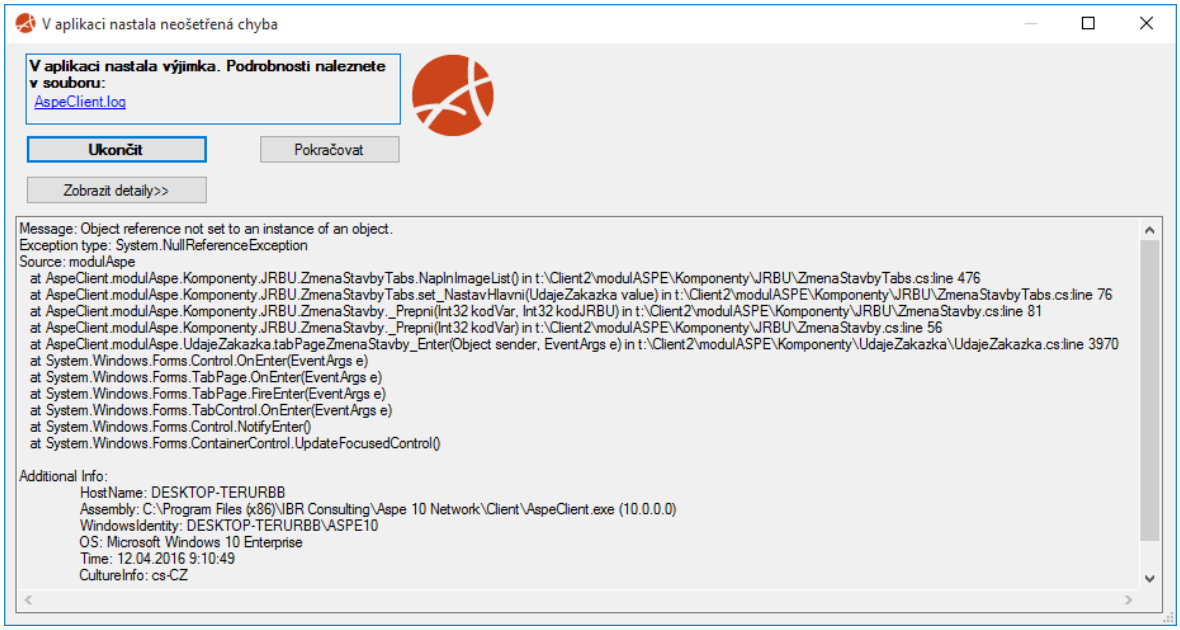
Testování je nedílnou součástí vývoje softwaru, v případě rozpočtových programů to ovšem platí dvojnásob. Je nezbytné věnovat zvláštní pozornost všem vzorcům, zobrazeným výpočtům a jejich následné interpretaci programem.

Při testování programu bylo v naprosté většině případů využíváno manuálního dynamického testování, a to hlavně z důvodu, že ve vývojovém týmu nejsou vyhrazeny pozice pro testery, kteří by měli na starost správu a vytváření automatických testů. Samotné testování je z velké části prováděno zaměstnanci technické podpory, kteří nemají potřebné znalosti pro tvorbu automatických testů a testování není jejich hlavní náplní práce.

4.5.1 Průběh testování funkcionalit

Přístupy k testování aplikace se měnily v závislosti na fázi, ve které se projekt nacházel. V době mého nástupu již byly vytvářeny spustitelné verze softwaru, ale stále se nacházel ve fázi vývoje, ve které nebyla hotova velká část funkcionalit. Z důvodu velké provázanosti jednotlivých akcí a cest v programu nebylo možné testovat veškerou funkcionalitu, a proto bylo testování omezeno na dílčí funkčnosti a grafické změny, které souvisely se změnou UI a používané terminologie v rámci formulářů. V praxi to znamenalo buď otestovat novou, či opravenou funkcionalitu na základě zadané úpravy, nebo aktivní vyhledávání nových chyb a jejich reportování. Pokud se jednalo o grafické nesrovnalosti, byly vytvořeny dokumenty s okomentovanými screenshoty, které byly zpravidla vztaženy k určitému celku – nejčastěji modulu. Následně byla vytvořena souhrnná úprava grafických prvků pro daný modul.

Chyby, které způsobovaly pády, či jinou nežádanou činnost programu, byly zapisovány a přeposílány formou bug reportu k analýze. Zde byl proveden rozbor a hledána příčina dané chyby a následně rozhodnuto o naléhavosti opravy dané chyby. Mnohdy bylo zjištěno, že chyba má souvislost s jinou částí programu a její opravení bude možné až po dokončení ostatních funkcionalit. Pro usnadnění opravy chyb byla využívána vývojová verze programu, která do programem zobrazených chybových hlášek přidávala i čísla řádků v programu, viz Obr. 10. V pozdějších fázích vývoje, kdy se intenzivnímu testování věnovalo více testerů, vznikla potřeba přidání dalších údajů identifikujících konkrétního testera a technické údaje o sestavě, na které byl program testován. Tento krok značně zjednodušil opravu chyb. Programátor, který chybu opravoval, mohl v případě nesrovnalostí hned konzultovat chybu přímo s daným testerem.



Obr. 10: Chybová hláška při pádu programu

Zdroj: Vlastní

Ve fázi vývoje, ve které již byla většina základních funkcionalit dokončena a program byl schopen komplexního testování, byla hlavní pozornost soustředěna na testování jednotlivých úprav a simulace kompletních procesů a akcí, které mohou nastat při používání programu. Dále byly laděny poslední detaily v rámci GUI, které byly během předešlého testování, či vývoje opomenuty. Před vydáním distribuční verze proběhlo opět komplexní regresní testování, aby se ověřila funkčnost již otestovaných funkcionalit, a byla tak zajištěna validace finálního produktu.

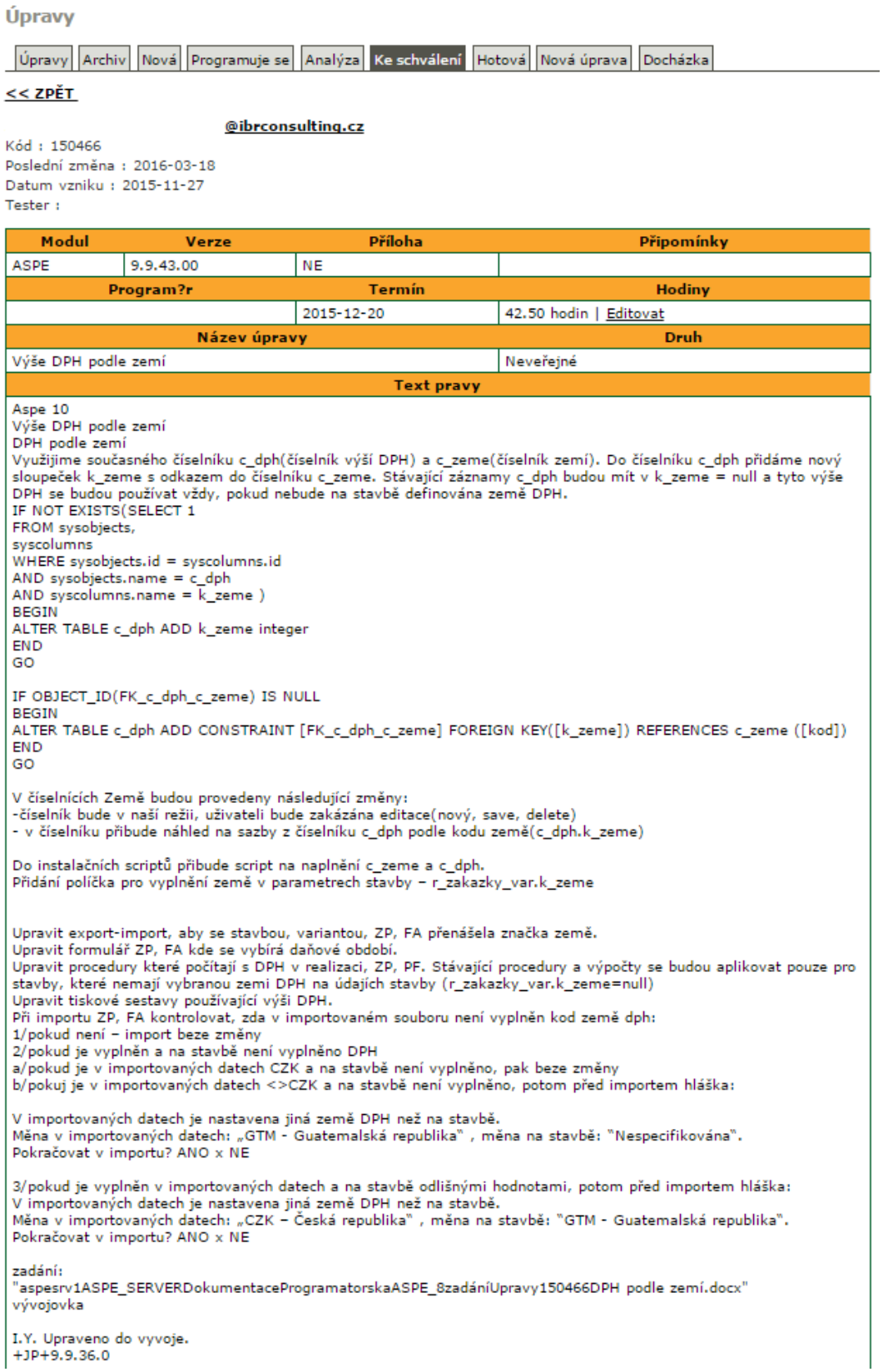
4.5.2 Zátěžové testy - testování odezvy a uživatelských práv

Samostatnou kapitolou v rámci testování byly testy spojené se síťovou verzí programu. Jelikož program Aspe je určen převážně pro velké firmy, které využívají převážně síťových řešení, je nezbytná správná funkčnost z hlediska oprávnění pro jednotlivé uživatele a zajištění rychlosti a funkčnosti při práci více uživatelů na stejných datech. Pro tato testování byly vyhrazeny speciální dny, ve kterých byl nasazen maximální počet testerů. Každý z testerů měl přiřazen vlastní účet s přednastavenými právy. Jednotliví testeři simulovali situace, které při používání programu mohou nastat. Kromě reportování

samotných chyb byly měřeny i neadekvátně dlouhé odezvy programu, či simulovány akce spojené s prací v síti. Testováno bylo například souběžné přihlášení více uživatelů, práva na akce v jednotlivých útvarech nebo souběžná práce na stejné stavbě.

4.5.3 Postup testování konkrétní úpravy

Jako modelový příklad konkrétního případu jsem vybral úpravu č. 150466, která upravovala funkčnost výběru země pro nastavení výše sazeb DPH na stavbě. Úprava rozšiřovala funkcionalitu pro výběr země pro nastavení sazeb DPH a s tím spojený výběr daňového období a následný export a import stavby. Z důvodů změn funkcionality bylo rozhodnuto, že pro předejití případné budoucí nekonzistenci dat bude z programu vyjmuta funkcionalita přidávání nových zemí pro nastavení sazeb DPH pro jednotlivá časová období a uživatelům zakázána editace stávajících sazeb DPH v číselníku *Země*. Přidání nových zemí již nebude možné provádět přes UI programu, ale pouze pomocí skriptu přímo nad databází konkrétní firmy, a zůstane tak pod správou vývojářů. Pohled na kompletní zadání úpravy poskytuje Obr. 11: Zadání úpravy 150466 na další straně.



Úpravy

Úpravy | Archiv | Nová | Programuje se | Analýza | Ke schválení | Hotová | Nová úprava | Docházka

<< ZPĚT

@ibrconsulting.cz

Kód : 150466
Poslední změna : 2016-03-18
Datum vzniku : 2015-11-27
Tester :

Modul	Verze	Příloha	Připomínky
ASPE	9.9.43.00	NE	
Program?r		**Termín**	**Hodiny**
		2015-12-20	42.50 hodin | Editovat
Název úpravy			**Druh**
Výše DPH podle zemí			Neveřejné

Text pravy

Aspe 10
Výše DPH podle zemí
DPH podle zemí
Využijime současného číselníku c_dph(číselník výší DPH) a c_zeme(číselník zemí). Do číselníku c_dph přidáme nový sloupeček k_zeme s odkazem do číselníku c_zeme. Stávající záznamy c_dph budou mít v k_zeme = null a tyto výše DPH se budou používat vždy, pokud nebude na stavbě definována země DPH.

```
IF NOT EXISTS(SELECT 1
FROM sysobjects,
syscolumns
WHERE sysobjects.id = syscolumns.id
AND sysobjects.name = c_dph
AND syscolumns.name = k_zeme )
BEGIN
ALTER TABLE c_dph ADD k_zeme integer
END
GO

IF OBJECT_ID(FK_c_dph_c_zeme) IS NULL
BEGIN
ALTER TABLE c_dph ADD CONSTRAINT [FK_c_dph_c_zeme] FOREIGN KEY([k_zeme]) REFERENCES c_zeme ([kod])
END
GO
```

V číselnících Země budou provedeny následující změny:
-číselník bude v naší režii, uživateli bude zakázána editace(nový, save, delete)
- v číselníku přibude náhled na sazby z číselníku c_dph podle kodu země(c_dph.k_zeme)

Do instalačních scriptů přibude script na naplnění c_zeme a c_dph.
Přidání políčka pro vyplnění země v parametrech stavby – r_zakazky_var.k_zeme

Upravit export-import, aby se stavbou, variantou, ZP, FA přenášela značka země.
Upravit formulář ZP, FA kde se vybírá daňové období.
Upravit procedury které počítají s DPH v realizaci, ZP, PF. Stávající procedury a výpočty se budou aplikovat pouze pro stavby, které nemají vybranou zemi DPH na údajích stavby (r_zakazky_var.k_zeme=null)
Upravit tiskové sestavy používající výši DPH.
Při importu ZP, FA kontrolovat, zda v importovaném souboru není vyplněn kod země dph:
1/pokud není – import beze změny
2/pokud je vyplněn a na stavbě není vyplněno DPH
a/pokud je v importovaných datech CZK a na stavbě není vyplněno, pak beze změny
b/pokuj je v importovaných datech <>CZK a na stavbě není vyplněno, potom před importem hláška:

V importovaných datech je nastavena jiná země DPH než na stavbě.
Měna v importovaných datech: „GTM - Guatemalská republika" , měna na stavbě: "Nespecifikována".
Pokračovat v importu? ANO x NE

3/pokud je vyplněn v importovaných datech a na stavbě odlišnými hodnotami, potom před importem hláška:
V importovaných datech je nastavena jiná země DPH než na stavbě.
Měna v importovaných datech: „CZK – Česká republika" , měna na stavbě: "GTM - Guatemalská republika".
Pokračovat v importu? ANO x NE

zadání:
"aspesrv1ASPE_SERVERDokumentaceProgramatorskaASPE_8zadáníUpravy150466DPH podle zemí.docx"
vývojovka

I.Y. Upraveno do vyvoje.
+JP+9.9.36.0

Obr. 11: Zadání úpravy 150466

Zdroj: Vlastní

Z důvodů změn funkcionality bylo nejdříve nutné vytvořit nový záznam v číselnících c_zeme a c_dph[10]. Pomocí níže zobrazeného skriptu (Obr. 12) byly číselníky naplněny daty.

Nejprve skript zkontroluje, zda existuje v databázi požadovaná země, a pokud neexistuje, tak založí novou zemi do tabulky c_zeme. Následně zkontroluje, zda v tabulce c_dph neexistuje vkládaný záznam, a případně jej vytvoří.

```
if not exists(select * from c_zeme where znacka = 'GER' and nazev = 'Germany')
BEGIN
insert into c_zeme (znacka, nazev) values ('GER','Germany')
END
GO

if not exists(select * from c_dph where rok= and mesic
      sazba1=0 and sazba2=7 and sazba3=19 and
      k_zeme=(select kod from c_zeme where znacka ='GER'))
BEGIN
insert into c_dph (rok, mesic, sazba1, sazba2, sazba3, k_zeme)
      select '2007','01',0,7,19 kod from c_zeme where znacka = 'GER'
END
GO
```

Obr. 12: Skript pro naplnění databáze

Zdroj: Vlastní

Následně mohly být otestovány jednotlivé body úpravy. Prvním krokem bylo ověření, zda jednotlivé exporty zapisují do příslušného XML[11] souboru informace o použité zemi DPH. Následně byly provedeny importy popsané v bodech 1-3 a ověřeno tak správné chování programu. V posledním kroku byly otestovány nové procedury využívající výši DPH na stavbě, na které je vybrána Země DPH, a následně správnost výstupu u tiskových sestav využívajících výši DPH. Nalezené nesrovnalosti byly zapsány do vyjádření ke konkrétní úpravě, která byla poté vrácena zpět k dopracování. Po opravě byla úprava následně schválena.

[10] c_zeme, c_dph: jedná se o pojmenování tabulek v SQL databázi programu
[11] XML – Extensible Markup Language – jedná se o rozšiřitelný značkovací jazyk využívaný k výměně dat mezi aplikacemi

4.6 Zhodnocení procesu vývoje a testování

Vývoj nové verze byl prodloužen vůči původnímu plánu o více než 10 měsíců. Za hlavní příčinu lze pokládat rozhodnutí o změně návrhu, která se týkala modulu REALIZACE a jeho rozdělení. Na prodloužení termínu měly ovšem vliv i problémy v rámci vývoje, kdy občas docházelo k vydání verze, ve které se objevily nové chyby, jež se dříve nevyskytovaly, a to v různě závažné míře. Následkem byla nutnost neustále regresně testovat již otestované funkcionality, což bylo značně časově náročné a ne zcela efektivní. Dalším problémem, s kterým se testování potýkalo, byla i absence dokumentace k některým částem programu. S tím byla spojena i vyšší náročnost testování, či neotestování některých cest v rámci programu, a tím nenalezení případné chyby nebo její objevení až v pozdějších fázích vývoje.

4.6.1 Ekonomické zhodnocení

Z ekonomického hlediska zatím nelze zcela určit přínos nového řešení. Současným zákazníkům, kteří platí technickou podporu, bude Aspe 10 zasláno zdarma jako součást smluvních podmínek. Ceny pro nově zakoupený program Aspe se pohybují v závislosti na zvoleném řešení (síťové nebo lokální) a zakoupených modulech. Cena lokální verze se pohybuje v rozmezí 30 000 – 75 000 Kč v závislosti na zvolených modulech. Síťové řešení začíná na 350 000 Kč a je závislé na zvolených modulech a počtu jednotlivých uživatelů. Cena roční technické podpory[12] poté představuje 18 % z celkové základní částky za program Další příjmy plynou z placených služeb pro zákazníky, které zahrnují technický servis, analytické a programátorské práce a další služby spojené s prací v Aspe.[13] Ceny pro tyto služby se pohybují v rozmezí 1 200 – 1 400 Kč/hod. Mimo tyto služby jsou pořádána i školení pro práci s programem, kde se cena za školitele pohybuje ve výše zmíněném rozmezí s tím, že je započítáno dalších 400 Kč za každého účastníka.[14] (Interní materiály firmy IBR Consulting s.r.o.).

[12] Zákazníci se zaplacenou technickou podporou mají nárok na pravidelné aktualizace a využívání tel. linky s technickou podporou. Dále mají v rámci ceníku zpoplatněných služeb technické podpory přibližně třetinové ceny oproti zákazníkům, kteří nemají zaplacenou technickou podporu.
[13] Jedná se převážně o převody rozpočtů do formátu XC4, či sehrání nabídek do Aspe.
[14] Jedná se o ceny bez DPH pro zákazníky s placenou technickou podporou.

Jelikož vývoj Aspe 10 probíhal souběžně s vývojem nových funkcionalit do Aspe 9, které byly implementovány do obou verzí, a jednotliví členové vývojového týmu pracovali zároveň na obou verzích, nelze celkovou cenu vývoje Aspe 10 zcela přesně peněžně vyjádřit. Vycházíme-li z průměrných platů v IT sektoru v rámci ČR za rok 2015 zveřejněným personální agenturou Hays Czech Republic a odhadnutém podílu členů vývojového týmu na vývoji Aspe 10, můžeme dojít k odhadovaným mzdovým nákladům[15] na vývoj. Celková doba vývoje byla 23 měsíců, pokud rozdělíme náklady z hlediska pozic ve vývojovém týmu, můžeme dojít k číslům:

Programátoři: Průměrná hrubá mzda se pohybuje od 40 000 Kč do 75 000 Kč v závislosti na pozici, zaměření a zkušenostech, což představuje superhrubou mzdu od 53 600 Kč do 100 500 Kč měsíčně. Na projektu se podílelo celkem 6 programátorů. Polovinu z nich tvořili programátoři s průměrnou mzdou 40 000 Kč s tím, že 2 z nich se podíleli na projektu ze 100 % a zbytek z 60 %. Dále 1 databázový specialista z 80 %, s průměrnou hrubou mzdou 50 000 Kč, která představuje superhrubou mzdu 67 000 Kč, a 1 senior programátor s průměrnou mzdou 75 000 Kč z 50 %. Celkové náklady na programátorskou činnost v rámci vývoje jsou přehledně zobrazeny v *Tab. 1: Mzdové náklady na programátory*.

Tab. 1: Mzdové náklady na programátory

Pozice	Mzdové náklady	Počet programátorů	Podíl na vývoji Aspe 10	Celkové náklady
C# Programátor	53600	2	1	2 465 600,00 Kč
C# Programátor	53600	2	0,6	1 479 360,00 Kč
Databázový specialista	67000	1	0,8	1 232 800,00 Kč
Senior vývojář	100500	4	0,5	4 623 000,00 Kč
	Cena celkem v Kč	**9**		**9 800 760,00 Kč**

Zdroj: Vlastní

Analytici: Na projektu se podíleli 2 analytici, z nichž jeden z nich zastává i roli vedoucího vývoje celého vývoje. Čas strávený na vývoje Aspe 10, v rámci jejich práce, představoval 60 % u analytika a 30 % u vedoucího vývoje. Průměrné platy v těchto pozicích jsou 50 000

[15] V ekonomickém zhodnocení jsou uvažovány pouze mzdové náklady. Náklady spojené s využíváním ICT, školením zaměstnanců apod. nejsou do zhodnocení zahrnuty, protože přímo nesouvisí s vývojem Aspe 10, ale jedná se o náklady spojené s běžným provozem.

Kč pro analytiky a 80 000 Kč pro vedoucí pracovníky vývojového oddělení (superhrubá mzda 67 000 Kč a 107 200 Kč). Celkové náklady na mzdy v rámci analýzy lze vyčíslit částkou **1 664 280 Kč**.

Testeři: Jak již bylo popsáno v kapitole 4.2, do této skupiny spadají ostatní pracovníci střediska. Někteří z pracovníků, kteří se podíleli na návrhu Aspe 10, byli do vývoje zapojeni již od počátku vývoje, ale většina pracovníků se postupně zapojila do vývoje 10 měsíců před vydáním. Jelikož se každý z pracovníků podílel na testování a návrhu jinou měrou a v konečných fázích vývoje vlastní testování představovalo jejich reálnou práci, kterou by odváděli i při používání Aspe 9. Zároveň, i díky rozmanitosti zastávaných pozic v rámci firemní struktury, se i jejich mzdové ohodnocení výrazně liší. Pokud zprůměrujeme platové ohodnocení všech členů týmu částkou 25 000 Kč měsíčně, což představuje mzdový náklad pro firmu 33 500 Kč a přepočteme čas strávený testováním všech „testerů“ na plné měsíční úvazky, dostaneme se přibližně na počet 4 pracovníků, kteří se 10 měsíců na plný úvazek podíleli na testování programu (z nejmenovaného zdroje). Celkový náklad na testování představuje přibližně **1 340 000 Kč**. Tato částka je ovšem pouze odhadem, který je díky výše zmíněným skutečnostem značně nepřesný. (Hays Czech Republic, 2016).

Celkové mzdové náklady na samotný vývoj aplikace se za 23 měsíců vyšplhaly přibližně k částce **12 800 000 Kč**. Z hlediska obchodního modelu, ve kterém většina příjmů přichází od stálých zákazníků ve formě poskytování technické podpory a současná verze jim bude zaslána zdarma v rámci smluvních podmínek, nelze zcela vyčíslit návratnost rozhodnutí o vývoji nové verze Aspe 10, oproti pokračování vývoje Aspe 9. V rámci rozšíření tržního podílu Aspe zatím nelze určit, na kolik vývoj splnil plánovaný obchodní cíl.[16]

[16] Obchodní model je koncipován tak, že středisko musí naplnit určitý objem tržeb za vlastní výkony v rámci roku. Předpokládaný ekonomický přínos spojený s novými zákazníky nelze vyčíslit z důvodů velké diverzity jednotlivých řešení. Ceny za produkty se pohybují od 20 000 Kč, ale mohou se vyšplhat do cen v řádech milionů korun.

5 Návrhy k zlepšení procesu vývoje

Jak již bylo zmíněno v předešlé kapitole, proces vývoje a s tím spojené testování aplikace má jisté nedostatky. Ty budou blíže přiblíženy v této kapitole a společně s nimi zde budou prezentovány návrhy na možná řešení těchto problémů. Některá drobná zlepšení již byla implementována, u jiných se jedná prozatím o pouhý návrh.

5.1 Dokumentace

V rámci testování některých funkcionalit spojených hlavně s registry, na kterých se nacházejí specifické funkcionality pro konkrétní firmy, jsem se potýkal s problémem chybějící dokumentace k obsahu daných registrů. Bylo velmi obtížné otestovat všechny přidané funkcionality, ke kterým neexistovala dokumentace, jež by jasně popsala obsah daných registrů. Z tohoto důvodu by bylo žádoucí, aby byla vypracována dokumentace k jednotlivým registrům, případně i k dalším funkcionalitám v rámci programu. Značně by to pomohlo urychlit proces zapracování nových zaměstnanců technické podpory, ale i zjednodušit samotné testování.

5.2 Životní cyklus úpravy

V rámci testování úprav někdy docházelo k případům, kdy daná úprava nebyla zcela přesně popsána, nebo byla v rámci programování po ústní domluvě s analýzou změněna, což vedlo k situacím, kdy tester vyhodnotil úpravu jako neschválenou, a ta byla opětovně vrácena programátorovi k přepracování. Tomuto problému by šlo předejít, kdyby byly upraveny cesty v rámci systému na evidenci úprav a daná úprava by byla nejdříve předána analýze, která by ověřila připomínky od testerů.

5.3 Zavedení automatických regresních testů

Jedním problémem, s kterým se tým během testování potýkal, byla nutnost provádět regresní testy pro ověření, zda nové implementace neměly vliv na stávající funkčnost. Během vývoje bylo takto odhaleno mnoho chyb, které se dříve neprojevovaly a vyskytly se

až v nové verzi. Z důvodu složitosti programu byly tyto testy vykonávané manuálně značně časově náročné, a proto by bylo dobré pro budoucí vývoj zvážit využití automatizace.

5.4 Zhodnocení nových návrhů

Zaznamenané problémy v rámci vývoje byly dle mého názoru způsobeny převážně vývojem nové verze, kdy bylo zasahováno do základních funkcí programu. Následkem byla nutnost testovat veškeré funkcionality programu. Do vývoje Aspe 10 bylo také zapojeno větší množství pracovníků, než když probíhá vývoj dílčích verzí na již fungujícím základu.

Přínos navrhovaných změn vidím v rámci vývoje větších celků, kdy je do vývoje zapojeno větší množství pracovníků. V případě vývoje malých funkcionalit, které jsou nasazovány na již fungující jádro programu, není využití automatizace v rámci regresního testování nezbytné, ale rozhodně by bylo dobré zvážit možnosti automatizace a její návratnosti v budoucnu při práci na nových modulech, či při přepracovávání stávajících modulů, které nebyly dokončeny v rámci vývoje verze 10.0.0.0.

Závěr

Hlavní cíl bakalářské práce, vyvinutí spolehlivé a řádně otestované verze Aspe 10, která nahradí současnou verzi Aspe 9, byl z velké většiny splněn. V současné době je program připraven k distribuci mezi zákazníky. Díky nově implementovaným funkcionalitám využívaných investory je pro mnoho dodavatelů nezbytné přejít na novou verzi programu. Současně s vydáním nové verze bude značně omezena technická podpora k předchozím verzím. Spolehlivost programu bude v příštích měsících ověřena v rámci nasazení do provozu mezi zákazníky a případné připomínky budou zapracovány do verze 10.0.1.0, která bude vydána po skončení akceptačního testování u jednoho ze zákazníků.

Celkový ekonomický přínos nového řešení nelze zatím vyčíslit, ale lze předpokládat, že díky nové verzi se značně zvedne zájem o placená školení v programu. Zároveň pokud se naplní obchodní cíle firmy, a nová verze přiláká nové zákazníky, lze předpokládat i růst tržeb díky novým instalacím a zvýšení počtu zákazníků využívajících technické podpory.

K dalším stanoveným úkolům práce se řadí vymezení role testování v rámci vývoje softwaru a popis používaných postupů v rámci vývoje a testování softwaru, kde pro tento účel byly v teoretické části představeny základní pojmy a praktiky. Všechny tyto úkoly byly splněny a lze říci, že v případě kladného přijetí u zákazníků budou všechny cíle splněny.

Stavební rozpočtový software Aspe 10 poskytuje uživatelům lepší komfort zajištěný přehlednějším UI. Dále obsahuje nové funkcionality, které značně zjednodušují práci se stavebními rozpočty. V neposlední řadě také poskytuje vývojovému týmu nové možnosti pro vývoj dalších funkcionalit. Jak již bylo popsáno výše, vývoj programu probíhá neustále, a zákazníkům tak budou průběžně předkládána další a další vylepšení.

Seznam použité literatury

Citace

BECK, K., BEEDLE, M., BENNEKUM, A. V., COCKBURN, A., CUNNINGHAM, W., FOWLER, M., GRENNING, J., HIGHSMITH, J., HUNT, A., JEFFRIES, R., KERN, J., MARICK, B., MARTIN, R. C., MELLOR, S., SCHWABER, K., SUTHERLAND, J. AND THOMAS, D. *Manifesto for Agile Software Development* [online]. 2001 [cit. 2016-01-13]. Dostupné z: http://agilemanifesto.org/

BUCHALCEVOVÁ, Alena. 2009. *Metodiky budování informačních systémů*. Vyd. 1. Praha: Oeconomica. ISBN 978-80-245-1540-3.

Businessdictionary.com. What is software tester? definition and meaning. In: *Businessdictionary.com*[online]. [cit. 2015-12-08]. Dostupné z: http://www.businessdictionary.com/definition/software-tester.html

Cigniti predicts software testing trends for 2016. (2016, Jan 18). *Business Wire*. New York. Dostupné také z: http://search.proquest.com/docview/1757675423?accountid=17116

GRAHAM, Dorothy, Erik VAN VEENENDAAL, Isabel EVANS a Rex BLACK. *Foundations of software testing: ISTQB certification*. Rev. ed. Australia: Course Technology Cengage Learning, 2008. ISBN 978-184-4809-899

HAJN, Petr. 2014. Business intelligence jako rozšíření ERP systémů stavebních firem. *IT Systems* [online]. 2014(4) [cit. 2016-03-26]. Dostupné z: http://www.systemonline.cz/business-intelligence/business-intelligence-jako-rozsireni-erp-systemu-stavebnich-firem.htm

HAYS CZECH REPUBLIC. 2016. Hays platový průzkumu 2016. In: *Nejnovější pracovní nabídky v Praze, Brně a České Republice* [online]. [cit. 2016-05-03]. Dostupné z:http://hays.cz/cs/groups/hays_common/@cz/@content/documents/digitalasset/hays_1602080.pdf

Interní materiály firmy IBR Consulting s.r.o.

KRÁLOVÁ, Iveta. 2012. *Metodika testování podle mezinárodních praktik a standardů*. Praha. Diplomová práce. Vysoká škola ekonomická v Praze. Vedoucí práce Buchalcevová, Alena. Dostupné také z: https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=40728

MYERS, Glenford J., Tom. BADGETT, Todd M. THOMAS a Corey SANDLER. 2004. *The art of software testing*. 2nd ed. Hoboken, N.J.: John Wiley. ISBN 04-714-6912-2.

NOVOTNÝ, Miroslav. 2012. Informační systémy ve stavebnictví: Krize mění přístup stavebních firem k IT.*IT Systems* [online]. **2012**(10) [cit. 2016-03-25]. Dostupné z: http://www.systemonline.cz/it-reseni-pro-stavebnictvi/informacni-systemy-ve-stavebnictvi.htm

PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. 2009. *Jak testuje software Microsoft*. Vyd. 1. Brno: Computer Press. ISBN 978-80-251-2869-5.

PATTON, Ron. 2002. *Testování softwaru*. Vyd. 1. Praha: Computer Press. Programování. ISBN 80-722-6636-5.

PECH, Pavel. 2013. ERP systémy ve stavebnictví. *IT Systems* [online]. **2013**(4) [cit. 2016-03-26]. Dostupné z: http://www.systemonline.cz/erp/erp-systemy-ve-stavebnictvi.htm

RATIONAL SOFTWARE CORPORATION. Rational Unified Process: Best Practices for Software Development Teams [online]. 2011 [cit. 2016-03-03]. Dostupné z: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Sample bug report. *Softwaretestinghelp* [online]. 2015, 2015-11-03 [cit. 2015-12-08]. Dostupné z: http://www.softwaretestinghelp.com/sample-bug-report/

SAUCE LABS. *Testing trends in 2016: A survey of software professionals* [online]. In: . s. 16 [cit. 2016-04-11]. Dostupné z: https://saucelabs.com/resources/white-papers/sauce-labs-state-of-testing-report-2016.pdf

SCHWABER, Ken a Jeff SUTHERLAND. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*[online]. 2013 [cit. 2016-01-23]. Dostupné z: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

Software Testing - Methods. c2016. *Tutorialspoint* [online]. [cit. 2016-04-08]. Dostupné z:http://www.tutorialspoint.com/software_testing/software_testing_methods.htm

SPILLNER, Andreas, Tilo LINZ a Hans SCHAEFER. 2014. *Software testing foundations: a study guide for the certified tester exam : foundation level, ISTQB compliant*. 4rd ed. Sebastopol, CA: Distributed by O'Reilly Media. ISBN 978-1-937538-42-2.

SPONSOR, Software Engineering Standards Committee of the IEEE Computer Society. 1994. *IEEE guide for software verification and validation plans* [online]. New York, NY: Institute of Electrical and Electronics Engineers [cit. 2016-05-02]. ISBN 07-381-0410-8. Dostupné z:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=838043

VERSIONONE. c2013. 7th Annual State of Agile Development Survey. In: *Agile Project Management Software for Agile Development | VersionOne* [online]. [cit. 2016-05-02]. Dostupné z: https://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf

VERSIONONE. c2014. 8th Annual State of Agile Development Survey. In: *Agile Project Management Software for Agile Development | VersionOne* [online]. [cit. 2016-05-02]. Dostupné z: https://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

VERSIONONE. c2015. 9th Annual State of Agile Development Survey. In: *Agile Project Management Software for Agile Development | VersionOne* [online]. [cit. 2016-05-02]. Dostupné z: https://www.versionone.com/pdf/state-of-agile-development-survey-ninth.pdf

ZIMMERMANN, Thomas, Rahul PREMRAJ, Nicolas BETTENBURG, Sascha JUST, Adrian SCHROTER a Cathrin WEISS. What Makes a Good Bug Report? In: *Software Engineering, IEEE Transactions on (Volume:36 , Issue: 5)* [online]. IEEE, 2010, 2011-07-20, s. 618-643 [cit. 2015-12-08]. DOI: http://dx.doi.org/10.1109/TSE.2010.63. ISSN 00985589. Dostupné z: http://search.proquest.com/docview/759299788/abstract?accountid=17116

Bibliografie

Aspe - stavební software. *Aspe* [online]. [cit. 2016-01-12]. Dostupné z: http://www.aspe.cz/cs/

How to write a good bug report? Tips and Tricks. *Softwaretestinghelp.com* [online]. 2015, 2015-11-03 [cit. 2015-12-08]. Dostupné z: http://www.softwaretestinghelp.com/how-to-write-good-bug-report/

ISO/IEC 12207:2008. *Systems and software engineering: Software life cycle processes*. Second edition. United States of America: ISO/IEC-IEEE, 2008.

SVOZILOVÁ, Alena, Ken JOHNSTON a Bj ROLLISON. 2011. *Projektový management*. 2., aktualiz. a dopl. vyd. Praha: Grada, 380 s. Expert (Grada). ISBN 978-80-247-3611-2.