1 Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS

Semestr	LS 2015/2016
Autoři	David Král (xkrad23), Martin Hradil (xhram29)
Téma	Extrémní programování – nová verze
Datum odevzdání	15. 5. 2016

2 Abstrakt

Srovnání nové verze extrémního programování s její starou verzí. Popsání nové verze a vysvětlení z jakých částí se skládá. Tyto části jsou poté popsány a vysvětleny.

3 Klíčová slova

XP1, XP2, extrémní programování, Beck

4 Obsah

1 Úvod

Extrémní programování ve verzi 2 upravuje a přeformulovává některé svoje principy a praktiky. Díky nim mohou programátoři nyní ještě lépe a efektivněji uspokojit potřeby zákazníka.

Cílem této práce je popsat a přiblížit staré i změněné hodnoty, principy a praktiky, a to podle knihy od K. Becka, Extreme Programming Explained: Embrace Change, 2nd Edition

Toho chceme dosáhnout krátkým popisem a vysvětlením jednotlivých částí. V případě, že se popisovaná část změnila vůči první verzi, bude i zdůrazněno a ukázáno, jakým stylem.

2 Co je extrémní programování

Patří mezi agilní metodiky vývoje softwaru.

Extrémní programování je softwarová metodika vytvořená v roce 1996 Kentem Beckem. Je to lehká metodika patřící do agilních metodik. Po použití v reálném prostředí byla mnoha lidmi uznána za úspěšnou, hlavně díky své orientaci na uspokojení zákazníka místo na splnění plánů. Extrémní programování podává výsledky, jak jsou potřeba a kdy jsou potřeba, s ohledem na časté změny požadavků zákazníka. Hlavním stavebním kamenem metodiky je dobrá týmová práce, díky které lze úspěšně vytvářet kód s co nejméně chybami.

V roce 2004 byla vydána druhá verze extrémního programování, ve které byly přidána hodnota respekt, některé praktiky byly přidány, jiné odstraněny, a všechny rozděleny do dvou skupin podle toho, jak by po sobě měly následovat.

3 Hodnoty

Extrémní programování verze 1 je založeno na čtyřech základních vlastnostech, kterým se říká „Hodnoty“.

- Komunikace
- Jednoduchost
- Zpětná vazba
- Odvaha

Ve druhé verzi této metodiky došlo k rozšíření těchto základních čtyř hodnot o jednu další.

- Respekt

3.1 Komunikace

Velká většina problémů, které při řízení projektu nastávají, je způsobena právě komunikací. Řešitel konkrétního problému nemusí například znát všechny požadavky kvůli nejasnému zadání. Pokud se ale pak nesnaží si tyto požadavky zjistit od zadavatele, velmi pravděpodobně dojde k chybnému vypracování.

Kvůli snaze předejít těmto problémům doporučuje extrémní programování včasnou a častou komunikaci, které tyto problémy podchytí.

3.2 Jednoduchost

Pod jednoduchostí si v tomto případě máme představit požadavek, který zajišťuje, že nebudeme vytvářet něco, co není potřeba a není požadováno, jen proto, že by to mohlo být

požadováno v budoucnosti. Díky tomuto požadavku se zadání omezí vždy jen na to, co je právě potřeba a nebude programátor zdržován věcmi, které ani nemusí mít smysl.

3.3 Zpětná vazba

Zpětná vazba je vždy důležitá. Potřebujeme často získat rychlou zpětnou vazbu, která nám pomůže například opravit chyby či získat informace, které hledáme. Proto je dobré mít odpovědi vždy po ruce na nejčastější otázky, které se při vytváření projektu a jeho plánování mohou objevit.

- Jak drahé bude vytvořit tento požadavek?
- Je tento kód v pořádku?
- A pod.

Pokud se tedy držíme rad extrémního programování, dokážeme si odpovědět na předchozí otázky. Díky tomu, že programátoři průběžně vytváří jednotkové testy, které dokáží otestovat, zda je kód v pořádku a pracuje jak má. Dalším typickým znakem zpětné vazby, je například zpětná vazba od zákazníka, který pomocí svých akceptačních testů dokáže ověřit v jakém stavu se produkt nachází.

3.4 Odvaha

Programátor musí mít odvahu jelikož občas je potřeba například změnit některou část rozhraní, která může způsobit obrovské množství chyb, které bude potřeba opravit.

Důležité je aby také dokázal čelit negativní zpětné vazbě. Není vždy jednoduché přijmou kritiku na něco, na čem člověk pracoval několik dnů.

Extrémní programování doporučuje využití například párového programování, které dokáže odvahu povzbudit.

3.5 Respekt

Je potřeba mít vzájemný respekt ve firmě ale také se zákazníkem oboustranně. Pokud je respekt jak má být, tak to značně usnadňuje komunikaci, zpětnou vazbu ale také zvyšuje odvahu jednotlivých členů tvořit změny a jít do větších výzev.

Dalo by se říct, že respekt je potřeba ke všem ostatním hodnotám aby fungovalo vše jak má.

4 Principy

Protože hodnoty extrémního programování jsou velice abstraktní, slouží principy jako most mezi hodnotami a jednotlivými praktikami. Mají za úkol pomoci porozumět důvodům za praktikami schovanými.

Například Test-First Programming je praktika, Communication a Feedback jsou hodnoty, a Mutual Benefit je princip - testy pomáhají při vývoji a zároveň říkají jiným programátorům, jestli svými změnami nevytvořili nějakou chybu.

4.1 Humanity (lidskost)

Programátoři nejsou stroje. Jsou to lidé, kteří mají nějaké potřeby, při jejichž splnění podávají lepší výsledky.

- Basic safety (základní bezpečnost): potrava, fyzická bezpečnost sebe a příbuzných, strach ze ztráty zaměstnání.
- Accomplishment (uznání): příležitost a schopnost přispět společnosti.
- Belonging (sounáležitost): příslušnost k nějaké skupině, sdílení jejích hodnost a přispívání ke společným cílům.
- Growth (růst): příležitost se učit a rozvíjet.
- Intimacy (porozumnění): schopnost vzájemného porozumění druhými.

Jsou i jiné potřeby, jako třeba odpočinek, cvičení, nebo socializace. Ty ale nemusí být splněny v pracovním prostředí. Složitost organizování týmu spočívá v nutnosti balancovat potřeby jednotlivců s potřebami týmu. I když komunikace je důležitá, týmu příliš neprospívá vyprávění příběhů z dovolené.

4.2 Economics (ekonomie)

Je třeba dělat hodnotnou práci, za kterou někdo platí. Ujistit se, že výstupy práce pomohou zákazníkovi ke splnění jeho cílů.

4.3 Mutual Benefit (společný prospěch)

Softwarový vývoj je týmová aktivita. Pracovní aktivity by měly přispět ku prospěchu všech. Mezi takové aktivity je možné zařadi například:

- Testy: pomohou mě teď i ostatním v budoucnu
- Refaktoring: odstranění složitosti pomůže mě i ostatním
- Pojmenovávání: Názvy vystihující podstatu věcí v kódu ulehčí orientaci mě i ostatním

4.4 Self-Similarity (podobnost)

Je dobré použít již jednou vymyšlené a prověřené řešení znovu. Když se v takovém řešení objeví chyba, je jednodušší ji najít i v dalších případech, kde bylo použito. Není ale nutné za každou cenu kopírovat. Pokud by bylo potřeba řešení příliš upravovat, je lepší vymyslet nové.

4.5 Improvement (zlepšování)

Člověk není dokonalý. Cílem je udělat to dnes nejlépe jak to jde a při tom porozumět, jak by to šlo udělat ještě lépe zítra. Nejlepší je nějak začít a poté postupně zdokonalovat.

4.6 Diversity (různost)

Různí lidé přinášení různé nápady. Různé nápady mohou přinášet konflikty ohledně dalšího postupu. Je důležité brát konflikt ne jako "moje řešení je lepší a to tvoje je k ničemu", ale jako "jsou dvě možnosti jako to vyřešit - jak to vymyslíme"

4.7 Reflection (přemítání)

Zamyslet se nad tím, jak to dělám a proč to dělám. Analyzovat rozhodnutí a poučit se z neúspěchů. Konverzace s ostatními o neúspěších nebo nemožnosti vymyslet řešení může pomoci v postupu.

4.8 Flow (tok)

Je lepší dodávat menší výsledky častěji, než větší pomaleji. Při nečekaném problému, který prodlouží například nasazení o týden, se co nejrychleji vrátit k předchozí pravidelnosti.

4.9 Opportunity (příležitost)

Je dobré vidět problémy jako příležitosti ke změně. Eliminace problému by neměla spočívat pouze v jeho vyřešení, ale k vylepšení a učení.

Je problém udělat roční plán? Uděláme čtvrtletní časem uvidíme. Člověk dělá mnoho chyb? Zkusíme párové programování.

4.10 Redundancy (přebytečnost)

V případě problému je dobré mít záložní řešení. Rychlé vyřešení tohoto problému, i když ne úplně nejlepším záložním řešením, nezpozdí vývoj tolik jako vymýšlení naprosto jiného řešení od začátku.

Ve své podstatě je fáze testování redundantní, protože testování, i když jiné, probíhá již během vývoje. Pokud je kód dostatečně popsán testy a funguje správně v několika nasazeních za sebou, lze testování dané části přeskočit.

4.11 Failure (selhání)

Pokud si nejsem jistý jak dál, je možné zkusit všechny možnosti. Třeba z nich vznikne řešení, které bude lepší než to první. Platí to i při více lidech, kteří se nemohou dohodnout na řešení. Po implementaci všech se možná ukáže, které je nejlepší.

4.12 Quality (kvalita)

Snížení kvality často nevede k rychlejšímu postupu a zvýšení nevede k pomalejšímu. Menší kvalita teď se totiž může projevit zpomalením v budoucnu, kdy je potřeba s nekvalitním kódem pracovat. Na druhou stranu by nejistota, zda můj kód je dost kvalitní, neměla zpomalit vývoj. Raději to teď udělat nejlépe jak to jde, a později to předělat, pokud bude třeba.

4.13 Baby Steps (malé kroky)

Při velkých změnách je dobré dělat malé kroky. Říci si "co je nejmenší rozpoznatelný kus práce, který vede správným směrem?" navede k práci, kterou je možno dokončit a odškrtnout si. Práce na velké změně najednou přináší tvorbu kódu, který stále není hotový a psychicky je taková práce více vyčerpávající.

4.14 Accepted Responsibility (přijímaná zodpovědnost)

Zodpovědnost nelze přidělit, lze ji pouze akceptovat. Pokud se člověku někdo snaží dát zodpovědnost, je jenom na něm, zda ji přijme. Pokud se někdo přihlásí k určité práci, tentýž člověk také odhadne její trvání a náklady. Osoba zodpovědná za implementaci user story je taktéž zodpovědná za její design, implementaci a testování.

Se zodpovědností přichází i autorita. Pokud odborník na procesy může přikázat , jak dělat určitou práci, ale sám ji nedělá nebo za ní není zodpovědný, je autorita a zodpovědnost špatně určena.

5 Praktiky

Existují primární a corollary. Primární praktiky jsou povinné pro úspěšné zavedení a fungování extrémního programování. Nicméně v praxi se často setkáváme s tím, že nejsou zaváděné úplně ale pouze částečně. Bývá to nejčastěji tím, že těm, kteří tuto metodiku zavádějí, zkrátka nevyhovují. Neuvědomují si však, že ač jsou pro ně z nějakého důvodu nevyhovující, tak v celkovém obraze se doplňují všechny navzájem a zajišťují fungování extrémního programování jako celku.

Dále zde jsou corollary a ty samy o sobě nejsou povinné zahrnout, každopádně byly vytvořeny k lepší funkci XP.

5.1 Primární

Následující uvedené praktiky patří do praktik z verze 1:

1. The Planning Game (Plánovací hra)
2. Small Releases (Vydávání malých verzí)
3. Metaphor (Metafora)
4. Simple Design (Jednoduchý návrh)
5. Testing (Testování)
6. Refactoring (Refaktorizace kódu)
7. Pair Programming (Párové programování)
8. Collective Ownership (Společné vlastnictví kódu)
9. Continuous Integration (Průběžná integrace)
10. 40-Hour Week (Udržitelné tempo)
11. On-Site Customer (Zákazník na pracovišti)
12. Coding Standards (Standardy kódu)

S příchodem nové verze extrémního programování se část těchto praktik přejmenovala nebo byla úplně vyřazena. Na místo vyřazených byly dosazeny praktiky nové a celkem jich je v nové verzi 13.

1. Sit Together (Sedět pohromadě)
2. Whole Team (Celý tým)
3. Informative Workspace (Informativní pracoviště)
4. Energized Work (Energická práce)
5. Pair Programming (Párové programování)
6. Stories (Úkoly)
7. Weekly Cycle (Týdenní cykly)
8. Quarterly Cycle (Čtvrtletní cyklus)
9. Slack (Odlehčení)
10. Ten-Minute Build (Desetiminutové sestavení)
11. Continuous Integration (Průběžná integrace)
12. Test-First Programming
13. Incremental Design

Aby bylo naprosto jasné, které praktiky zůstaly, které byly přejmenovány a jaké byly přidány, pokusím se shrnout předchozí údaje do názorné tabulky.

XP1 název	XP2 název
-	Sit Together
-	Whole team
-	Informative Workspace

40-Hour Week	Energized Work
Pair Programming	Pair Programming
The Planning Game	Stories
The Planning Game	Weekly Cycle
Small Releases	Quarterly Cycle
-	Slack
-	Ten-minute build
Continuous Integration	Continuous integration
Testing	Test-first Programming
Simple Design	Incremental Design
Metaphor	-
Collective Ownership	Corollary -> Shared code
On-Site Customer	Corollary -> Real customer involvement
Coding Standards	-

5.1.1 Sit together

Celý tým zahrnující vývojáře, managery, zákazníka by měl být pohromadě nejlépe v jedné kanceláři. Navzájem vědět o problémech a spolupracovat na jejich řešení.

Kladné body

- Zjednodušuje komunikaci a zvyšuje šanci na úspěšné dokončení projektu
- Zvyšuje neformální komunikaci a tím se tým sjednocuje

Záporné body

- Velké množství lidí v jedné místnosti zvyšuje hladinu hluku a tím je tým rušen

5.1.2 Whole team

Veškeří kvalifikovaní lidé, které tým potřebuje pro svou funkci by měli být zahrnuti v týmu. Toto zahrnuje jak speciální technické a business znalosti ale také lidi, kteří jsou odpovědní za projektové řízení.

Kladné body

- Takto může tým pokračovat bez zdržení s hledáním potřebných lidí na konkrétní úkoly

Záporné body

- Tento přístup vyžaduje aby jeho členové byli zaměstnání na plný úvazek. Nicméně specialisté jsou často potřeba na více projektech a pokud se jedná o nějakou specializaci co není běžná, tak není lehké tyto lidi získat.

5.1.3 Informative Workplace

Všechny důležité informace, které se týkají projektu a jeho fáze, by měly být dostupné přímo na pracovišti. Například na tabuli, nástěnce či pomocí papírků na zdi.

Kladné body

- Každý má okamžitý přehled o probíhající etapě projektu a jak si projekt vede

Záporné body

- Ve velkém projektu možná náročnost na místo a nepřehlednost

5.1.4 Energized work

Cílem je pracovat co nejvíce hodin, kdy člověk dokáže zůstat vysoce produktivní. Strávit programováním 16 hodin a být následující den k nepoužití vede k celkově menší produktivitě. Při přesčasové práci, občas za pomoci kávy nebo dalších povzbuzovadel, je riziko, že programátor kvůli únavě a nesoustředění naopak začne posouvat postup zpět. Toho si při tom ani nemusí být vědom.

Únava má podobné následky jako nemoc. Je důležité dbát na své zdraví, protože pouze zdravý člověk dokáže dát 100% do své práce a spolupráce. Být v práci nemocný přináší rizika pro další členy týmu, které může v práci zpomalovat nebo je nakazit.

Součástí dne by měl být čas určený pro intenzivní práci. Vypnutí telefonu a emailu na dvě hodiny a věnování se čistě programování může přinést pozitivní výsledky, které v budoucnu mohou zařídit menší počet hodin v práci nebo větší efektivitu.

Kladné body

- Lepší fyzické i psychické zdraví
- Menší počet chyb

Záporné body

- Těžko měřitelné

5.1.5 Pair programming

Kód je psán dvojicí programátorů, kteří sdílejí jeden počítač. Jeden píše kód a druhý ho sleduje, hledá chyby a vylepšení. Dvojice si mění pozice na základe toho, kdo vidí jaké vylepšení či ví jak pokračovat. Zároveň se často mění jednotlivé dvojice (např. 2x denně). Díky vzájemné kontrole je možné podchytit značné množství chyb, které může být způsobeno únavou.

Kladné body

- Každý ze členů týmu ví jak funguje celý systém
- Okamžitá kontrola čtyř očí

Záporné body

- Někteří vývojáři tento styl nepodporují
- Problémy kvůli různému stylu programování

5.1.6 Stories

Veškeré požadavky na systém jsou napsané formou úkolů. Připomínky těchto úkolů jsou uvedeny na kartičkách, které jsou dostupné každému členovi týmu. Detailnější informace o úkolu jsou předávány ústně a kartičky slouží pouze pro připomenutí. Karet musí být pouze malý počet kvůli udržení přehlednosti.

Kladné body

- Každý člen ví co je právě v plánu

Záporné body

- Pro některé typy funkcionalit mohou být tyto úkoly příliš málo konkrétní
- Požadavky, které nejsou funkcemi se těžko zachycují pomocí úkolů

5.1.7 Weekly cycle

Jedná se o iterace trvající 1 nebo maximálně 2 týdny. Mohou se skládat i z více úkolů. Základní postup v iteraci musí být dostupný zákazníkovi k náhledu a kontrole. Během etap jsou úkoly zadané fixně a neměly by se měnit.

Kladné body

- Díky neměnnému plánu si může zaměstnanec rozvrhnout práci

Záporné body

- Riziko zaseknutí se na problému a následné přelévání úkolů do další iterace

5.1.8 Quarterly cycle

Vydání nové verze je naplánováno alespoň 4x do roka. Nová verze je nasazována do provozu na používání opravdovými uživateli.

Kladné body

- Možnost relativně rychle zjistit, zda to co bylo naprogramováno mělo nějaký přínos.

Záporné body

- Takto časté vydávání je značně složité a ne vždy možné.

5.1.9 Slack

Do každé iterace je dobré zakomponovat také úkoly, které nemají tak velkou prioritu a tím pádem je možné je vynechat a upustit od nich pokud se iterace dostane do časového skluzu.

Kladné body

- Pokud je etapa ve skluzu, je možné její úspěšnost zachránit odložením úkolů.

Záporné body

- Zaměstnanci by se mohli naučit, že pokud nestíhají, tak se úkol proste jednoduše vypustí a pak by plánování spousty úkolů mohlo být zbytečné

5.1.10 Ten-minute build

Při sestavování systémů a spouštění testů by celá tato operace neměla přesáhnout 10 minut. Díky tomu je možné pro programátora získat velmi rychlou zpětnou vazbu od systému, které program sestavuje, v jakém stavu se nachází a jestli neobsahuje chyby.

Kladné body

- Rychlá zpětná vazba

Záporné body

- Pokud se vytváří multiplatformní systém, tak může být nereálné otestování aplikace na veškerých podporovaných systémech

5.1.11 Continuous integration

Do produkčního softwaru jsou přidávány změny nejméně jednou denně a po jejich přidání jsou spuštěny automatické testy, které poskytnou okamžitou zpětnou vazbu. Takto vygenerovaný typ buildu reprezentuje současný stav projektu. Ve většině případů by tudíž měl být v naprostém pořádku. Pokud by dlouhodobě nebyl, je zřejmé, že se v řízení projektu něco dělá špatně.

Kladné body

- Dobrý přehled o stavu projektu

Záporné body

- V případě rozsáhlých refaktorací je velmi namáhavé udržet naprosto perfektně funkční build

5.1.12 Test-first programming

Před začátkem celého programování systému, jsou napsané automatické testy, které se posléze programátor snaží naprogramovat. Úspěšnost programátora při plnění cílů se poté nejvíce rozhodují na základě toho, zda fungují tyto předpřipravené testy. To znamená, že pokud testy prochází, tak se úkol považuje za splněný.

Kladné body

- Jasně stanoveny podmínky pro funkcionalitu před jejím naprogramováním
- Okamžitá zpětná vazba

Záporné body

- Velmi náročné na údržbu v případě refaktoračních změn

5.1.13 Incremental design

Veškerý design systému není vytvořen na začátku projektu ale vyvíjí společně s projektem. Tím je zajištěno, že se přizpůsobuje nejnovějším požadavkům. Design je vytvářen na co nejjednodušším systému funkčnosti aby byl lehce upravovatelný a přizpůsobitelný.

Kladné body

- Design sedí naprosto potřebám systému

Záporné body

- Pokud je špatně tento přístup použitý, tak to vede k velkému množství přepracovávání

5.2 Důsledkové (Corollary) praktiky

Důsledkové praktiky dále zlepšují vývoj, ale k jejich úspěšnému použití je třeba zvládnout odpovídající primární praktiky.

Například pokud není počet chyb v kódu téměř nula (za pomocí párového programování, průběžné integrace a test-first programování), tak Daily Deployment může naopak uškodit.

Některé praktiky byly převzaty a přepracovány z XP 1, některé jsou úplně nové. Srovnání je v následující tabulce.

XP 1 název	XP 2 Corollary
On-site Customer	Real Customer Involvement (zapojení zákazníka)
-	Incremental Deployment (přírustkové nasazování)
-	Team Continuity (týmová stálost)
-	Shrinking Teams (změnšování týmů)
-	Root Cause Analysis (analýza hlavní příčiny)
Collective Code Ownership	Shared Code (sdílený kód)
Simple Design	Code and Tests (kód a testy)
-	Single Code Base (jedna vývojová větev)
-	Daily Deployment (denní nasazení)
-	Negotiated Scope Contract (vyjednávání rozsahu)
-	Pay-per-use (platba za použití)

5.2.1

5.2.2 Real Customer Involvement

Lidé, kteří budou používat vyvíjený software, se mohou účastnit čtvrtletních a týdenních plánování, kde mohou dostat část zdrojů na vývoj čehokoli chtějí. Pokud je zákazník v situaci, že je ve svém odvětví postaven před problém, který budou muset řešit všichni jeho konkurenti, včasná reakce a změna plánů může přinést významnou konkurenční výhodu.

Cílem účasti zákazníka v procesu vývoje je minimalizace plýtvání zdroji pomocí přímého kontaktu osob s potřebami s těmi, kteří mohou tyto potřeby splnit.

Kladné body

- Rychlá reakce na změny potřeb zákazníka
- Minimalizace plýtvání zdroji

Záporné body

- Více specifický výsledný software
- Nutný pořádek v organizaci

5.2.3 Incremental Deployment

Nahrazení starého systému sebou může přinést neočekávané problémy, kvůli kterým nebude možné nový systém používat a bude ho nutné upravit. To prodlužuje a prodražuje celý projekt

Vhodnější je po částech nahrazovat starý program novým, pokud lze tyto části navzájem vyměnit nebo provozovat současně .

Kladné body

- Rychlá reakce na nekompatibilitu
- Kratší čas na konečné nasazení

Záporné body

- Větší náklady na školení a případnou úpravu dat
- Delší čas na vývoj

5.2.4 Team Continuity

Ve velkých organizacích je snaha o oddělení lidí od znalostí. Programátoři jsou "lidské zdroje", které po projektu putují zpět do využitelných zdrojů. Malé firmy tento problém nemají, protože existuje pouze jeden "tým". Přitom týmová spolupráce je základem úspěšného vývoje. Čím více se lidé znají, tím lépe se jim spolupracuje a navzájem vědí, kdo je v čem dobrý, jak se dá na koho spolehnout, a jaké od koho očekávat výsledky.

Týmy lidí, které existují dlouhodobě, budou mít lepší výsledky, než když jsou programátoři přiřazováni pouze na jednotlivé projekty. Rotace malého počtu lidí mezi týmy zajistí potřebnou změnu kolektivu a distribuci znalostí napříč týmy.

Kladné body

- Rychlejší začátek vývoje
- Lepší soudržnost týmu a lehčí překonávání problémů

Záporné body

- Koncentrace znalostí v jednom týmu
- Složitější plánování lidí

5.2.5 Shrinking Teams

Rozložení práce rovnoměrně mezi členy týmu způsobuje, že po čase, kdy se lidé zlepší, nejsou všichni vytíženi 100% času. Kdyby byla práce přidělována členům týmu postupně na 100%-ní vytížení s tím, že na posledního zbyde méně než 100%, po nějakém čase se zlepší efektivita členů a na posledního práce už nezbude. Tohoto člověka je poté možné přeřadit na jiný projekt

Kladné body

- Efektivnější využití lidí

Záporné body

- Možná sociální problémy v rámci týmu kvůli množství práce

5.2.6 Root Cause Analysis

Při nálezu chyby je důležité odstranit nejen samotnou chybu, ale i její příčinu, a zjistit, proč chyba vznikla a jak zabezpečit, aby se už neopakovala. K tomu lze využít systému "Five Whys"

- Proč jsme chybu nezjistili dříve? Protože jsme nevěděli, že částka může být někdy negativní.
- Proč jsme to nevěděli? Protože to ví jenom pan X a ten není součástí týmu.
- Proč není součástí týmu? Protože stále podporuje starý systém.
- Proč to nikdo jiný neví? Protože to pro management není priorita.
- Proč to není priorita? Protože management neví, že nalezení chyby dříve by stálo méně.

Zodpovědět si pětkrát "proč" může pomoci s tím, že stejný typ chyby se již nebude opakovat.

Kladné body

- Minimalizace chyb v budoucnu .

Záporné body

- Více času na řešení jednotlivých chyb.

5.2.7 Shared Code

Kdokoli v týmu může vylepšit jakýkoli kód. Pokud je něco špatně v části programu náležící někomu jinému, mohu tuto část ihned opravit, pokud se týká toho, na čem dělám já.

Při použitý této praktiky je velké riziko, že do kódu bude zanesena další chyba a kód bude nesrozumitelný. Je proto velice důležité, aby členové týmu měli na mysli, že záleží na výsledku týmu a ne na tom, co každý udělá zvlášť.

Kladné body

- Rychlé opravy chyb

Záporné body

- Velké riziko dalších chyb
- Nutná vysoká kvalita kódu

5.2.8 Code and Tests

Kód a testy jsou jediné dokumenty o programu, které jsou vždy aktuální. Další dokumenty by měly být generovány z kódu a testů, a důležitá historie projektu by měla být v paměti lidí, kteří na něm pracují.

Zákazník platí za to, co systém umí teď a co tým dovede systém naučit zítra. Jakýkoli dokument podporující tyto dvě věci je přínosný, zbytek je pouze plýtvání. Části dokumentace jsou najednou k ničemu, když jsou do vývoje uvedeny změny a čas strávený tvorbou dokumentace mohl být využit jinak.

Kladné body

- Menší náklady na dokumentaci

Záporné body

- Nutnost verzování
- Větší kód kvůli komentářům.

5.2.9 Single Code Base

Vývoj by měl probíhat pouze v jedné větvi. Dlouhodobý vývoj ve více větvích může vést k chybám a prodražení. Oprava chyby v jedné větvi může způsobit chybu v jiné, jejíž oprava způsobí chybu v jiné... lepší je všechny větve sjednotit a odlišnosti řešit pomocí parametrů a konfiguračních souborů.

Kladné body

- Menší počet chyb

Záporné body

- Sjednocení existujících větví může být složité a nákladné

5.2.10 Daily Deployment

Doba mezi vývojem a nasazením do produkce by mela být co nejmenší. Díky tomu lze odhalit chyby dříve.

Kladné body

- Rychlejší zpětná vazba
- Rychlejší nasazení

Záporné body

- Nutnost velmi malého počtu chyb
- Nutnost vysoké automatizace nasazování.
- Nutnost důvěry zákazníka

5.2.11 Negotiated Scope Contract

Ve vývoji softwaru jsou obvykle čas a peníze neměnné, ale lze pohybovat s požadavky a rozsahem. Nahrazením dlouhé smlouvy několika kratšími lze snížit riziko, že bude třeba pracovat na již nepotřebných požadavcích. Mezi kontrakty se lze se zákazníkem dohodnout na dalším postupu nebo vyhodnotit požadavky a připomínky uživatelů.

Kladné body

- Nevyvíjení toho, na čem již nezáleží.

Záporné body

- Nejistota, co se vlastně bude vyvíjet.

5.2.12 Pay-per-use (platba za použití)

Peníze jsou ta nejlepší zpětná vazba. Uživatelé platí za to, co se jim používá lépe. Jestliže po změně klesl počet použití, asi bude vhodné s tím něco udělat. Platba za použití nutí vytvářet software, který se bude dobře používat, na rozdíl od platby za release.

Kladné body

- Zákazník dává užíváním najevo, jak se mu produkt líbí.
- Rychlá identifikace uživatelského problému.

Záporné body

- Méně jisté příjmy

6 Závěr

Naše práce se primárně zaměřovala na popsání nové verze extrémního programování. Tam kde to bylo možné, jsme vytvořili také názorné srovnání s její předchozí verzí. Popis předchozí verze nicméně nebyl zahrnutý v této práci, jelikož se nejedná o novou verzi.

Problémy jsme spatřovali nejčastěji ve zjišťování informací k danému tématu, jelikož na většině zdrojů, kam jsme se dívali, se tato nová verze neodlišuje slovně od staré.

7 Citovaná literatura

1. **Beck, Kent a Andres, Cynthia.** *Extreme Programming Explained: Embrace Change, 2nd Edition.* Boston, MA : Addison-Wesley, 2004. 0321278658.

2. **Williams, Laurie.** Case Study Retrospective: Kent Beck's XP Versions 1 and 2. *Center for Systems and Software Engineering.* [Online] 7. 3 2005. [Citace: 1. 5 2016.] http://sunset.usc.edu/events/2005/arr/proceedings/presentations/OneDayWorkshops/agile/williams.pdf.

3. **Wake, William.** Overview of XP Explained, Second Edition. *Exploring Extreme Programming.* [Online] XP123, 2010. [Citace: 2016. 5 1.] http://xp123.com/xplor/xp0502/index.shtml.

4. **Prechelt, Lutz.** Agile Methods: eXtreme Programming (XP). *Institut für Informatik.* [Online] 18. 1 2016. [Citace: 1. 5 2016.] http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT2-2015/42_agile_xp.pdf.

5. **Wells, Don.** Extreme Programming Rules. *Extreme Programming: A gentle introduction.* [Online] 8. 10 2013. [Citace: 1. 5 2016.] http://www.extremeprogramming.org/rules.html.