

Autopsia extrémneho programovania

JÁN SUCHAL

*Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
johno@jismf.net*

Abstrakt. Agilné metódy programovania, kde sa za hlavného predstaviteľa pokladá práve extrémne programovanie, sú považované za mimoriadne vhodné najmä pre menšie projekty s často sa meniacimi požiadavkami. Vzniká tak však aj mylná predstava o tom, že praktiky, ktoré tvoria ich jadro, sú pre iné projekty nevhodné a spôsobujú ich zlyhania. Dostávajú sa takto do úzadia bez hlbšieho skúmania ich pravej podstaty. V poslednej dobe sa však ukazuje, že niektoré praktiky sú natoľko rozumné, že vhodný výber a ich použitie môže mať pozitívny dopad na takmer ľubovoľný projekt. Je preto potrebné spoznať príčiny vzniku ako aj výhody a dôsledky týchto praktík a nezavrhať ich hneď ako celok.

Tradičné miesto agilných metód

Je všeobecne známe, že agilné metódy programovania sú vhodné najmä pre projekty s často sa meniacimi požiadavkami a menšie tímy. To je síce pravda, avšak dôsledkom tohto tvrdenia vzniká však aj mylná predstava o tom, že akonáhle je projekt trochu väčší alebo je špecifikácia projektu dopredu dostatočne jasná, treba na agilné metódy rýchlo zabudnúť. Problém vidím v tom, že agilné metódy sa nesprávne chápu ako nejaký postup práce. Ak potom zlyhal postup, tak ho treba zavhnúť. Väčšina agilných metód je však definovaná dosť vágne, veľmi často len formuláciou akýchsi základných princípov. O nejakom detailne prepracovanom pracovnom procese sa tu hovoriť však určite nedá.

Extrémne programovanie (XP)

Extrémne programovanie ako agilná metóda programovania sa v súčasnosti pokladá za najrozšírenejšiu a najznámejšiu zo všetkých. V očiach mnohých je XP dokonca akýmsi predstaviteľom a základným stelesnením hlavných princípov či praktík ostatných agilných metód.

Niektoré z týchto praktík sú však natoľko kontroverzné a ich pravý význam a dôsledky natoľko nepochopené, že považujem za dôležité ich tu podrobnejšie rozobrať.

Malé verzie

Akonáhle sa do vyvíjaného produktu pridáva nová funkcionálna dôležitá z hľadiska zákazníka je potrebné hneď vydať novú verziu. Navyše to treba robiť tak často, ako sa len dá.

Zrejmovou výhodou tohto prístupu je to, že čím častejšie vývojový tím vydáva nové verzie, tým má lepšiu spätnú väzbu od zákazníka. Dokáže rýchlejšie reagovať na vznikajúce požiadavky a problémy. Čím skôr sa teda zavedie do systému nová funkcionálna, tým viac času má vývojový tím na jej prípadné opravy.

Tento nápad je vo svojej podstate výborný avšak má aj svoje úskalia. Čo ak zákazník potrebuje určité kritické funkcie a nová verzia ich ešte všetky neobsahuje? Čo ak je potrebné vykonať pred nasadením náročnú migráciu dát? Nebude častá distribúcia novej verzie v nasadzovanom prostredí príliš nákladná?

Ideálne pre tento spôsob nasadzovania sa však zdajú byť centralizované webové aplikácie. Stačí nasadiť novú verziu na centrálny server a všetci v tom okamihu používajú nový, vylepšený produkt.

Jednoduchý návrh, emergentná architektúra

V XP je systém navrhovaný čo najjednoduchšie pričom hlavným a jediným cieľom je to, aby fungoval. Akékoľvek komplikácie sú odstraňované hneď ako sa spozorujú a do systému sa zo zásady pridáva nová funkcionálna až vtedy, keď je naozaj potrebná. Tento princíp je známy pod skratkou YAGNI (*You Arent Gonna Need It*) alebo KISS (*Keep It Simple, Stupid*).

Zdá sa, že YAGNI je v priamom rozpore so zaužívaným spôsobom vývoja softvéru - špecifikácia, analýza, návrh, implementácia. Ak však mne ako programátorovi vznikne potreba použiť nejakú funkcionálnu, ktorú ešte nemám implementovanú, tak vlastne vykonávam akúsi skrytú špecifikáciu. Výhoda je v tom, že presne viem akú funkcionálnu potrebujem. V klasickom vodopádovom modeli sa môže aj ten najlepší analytik snažiť ako len vie, ale takú jasnú predstavu ako ja bude mať len zriedkavo. Ak bude príliš špecifikovať, tak len pridá programátorom zbytočnú robotu, ale ak bude príliš zjednodušovať, tak bude funkcionálna nakoniec znova chýbať.

Rozdiel oproti klasickému modelu je v tom, že jednotlivé fázy sa nevykonávajú iba raz, ale mnohokrát – v iteráciách. XP predpokladá, že výsledkom veľkého počtu takýchto iterácií začne postupne vznikať (emergovať) aj celková architektúra systému.

To, čomu sa ale snaží XP pomocou YAGNI princípu vyhnúť je zbytočne veľký návrh (*Big Design Up Front*). Predpokladá totiž, že funkcionálnu sa nikdy nepodarí odhadnúť úplne presne a navyše sa môžu zmeniť aj požiadavky zákazníka. Čo je však potrebné zdôrazniť je fakt, že XP úplne nezavrhuje návrh ako taký. Akýsi hrubý návrh

sa vytvára pri prvotnom plánovaní funkcionalít obsiahnutých v jednotlivých verziách produktu. Navyše ak ide o väčšie projekty, je dokonca nevyhnutné vykonať akúsi dekompozíciu systému na menšie celky alebo vrstvy, ktoré už majú jasnejšie definovanú funkcionalitu. [1]

Ak je teda možné nejakú funkcionalitu produktu dopredu dostatočne špecifikovať, nie je dodržovanie YAGNI princípu až také nevyhnutné, avšak XP ho odporúča prakticky všade.

Testovanie

V XP sa kladie mimoriadny dôraz na testovanie, pričom existujú dva typy testov:

- Testy jednotiek, ktoré slúžia na testovanie jednotlivých častí kódu, väčšinou na úrovni tried.
- Akceptačné testy, ktoré vychádzajú z takzvaných užívateľských scenárov, pričom testujú očakávanú funkcionalitu produktu ako celku.

Oba typy testov sú však plne automatizované, úplne pokrývajú potrebnú funkcionalitu a je možné ich kedykoľvek spustiť. Ak sú k dispozícii takéto testy, tak otestovanie kompletnej funkčnosti systému trvá väčšinou len zopár minút, na otestovanie triedy stačí maximálne niekoľko sekúnd. Dôsledky tohto prístupu sú však obrovské. Nikto z programátorov sa teda nemusí báť zmeniť akýkoľvek kód. Testy mu slúžia ako akási záchranná sieť a pokiaľ sú naozaj kvalitné, tak nielen chybu odhalia, ale ju dokážu aj veľmi presne lokalizovať. Programátori striktné využívajúci takéto automatizované testovanie dokonca tvrdia, že dôsledkom toho sa z ich programátorského života úplne vytratila fáza hľadania chýb (*debuging*).

Táto praktika sa v praxi natoľko osvedčila, že vznikla samostatná metóda programovania známa ako testami riadený vývoj (*test driven development, TDD*). Je založená na tom, že ešte pred tým ako sa napíše akákoľvek nová funkcionalita, musí sa napísať automatický test, ktorý túto funkcionalitu overí. Až potom sa môže pristúpiť k samotnej implementácii funkcionality. Akonáhle prebehnú všetky testy úspešne vykoná sa refaktoring kódu a pokračuje sa vytvorením ďalšieho testu. Tento postup je známy ako *red/green/refactor* a nazýva sa *TDD mantra*.

Táto praktika dokonca nepriamo podporuje aj spomínaný YAGNI princíp, pretože ak by chcel programátor pridať nejakú zbytočnú funkčnosť, musel by pre ňu najprv napísať test a to programátor ako tvor lenivý dobrovoľne robiť nikdy nebude.

Ukazuje sa, že kód vytváraný takýmto prístupom je aj oveľa kvalitnejší a prehľadnejší. Testy dokonca môžu slúžiť ako nepretržite aktuálna dokumentácia funkčnosti, pretože sú väčšinou tvorené ako jednoduché príklady použitia danej časti systému či produktu ako takého. Dôležité je zdôrazniť aj to, že testy samotné musia mať minimálne takú kvalitu ako kód, ktorý testujú.

Ani táto metóda sa však nedá použiť úplne všade. Automatizácia testovania používateľských rozhraní je vo všeobecnosti príliš komplikovaná, aj keď pri webových aplikáciách už sú dostupné vcelku robustné testovacie riešenia. Taktiež testovanie distribuovaných systémov týmto spôsobom je ešte len v plienkach.

Táto metóda sa však dá použiť vždy, bez ohľadu na to o aký rozsah projektu ide.

Párové programovanie

XP sa preslávilo práve svojou koncepciou párového programovania. Základom je to, že zdrojový kód tvoria dvaja ľudia naraz. Jeden monitor, jedna klávesnica, jeden počítač a dvaja ľudia. Človek, ktorý práve sedí pri klávesnici a píše kód premýšľa o tom, ako najlepšie implementovať konkrétnu metódu. Ten druhý, čo mu hľadá cez rameno, rozmýšľa globálnejšie. Rozmýšľa o tom, či by sa systém nedal nejako zjednodušiť, či sa neporuší nejaká iná funkcionálnosť systému. Ak vidí, že partnerovi niečo nejde tak ako by si predstavoval bez váhania zoberie klávesnicu a role sa vymenia.

Výsledky dosiahnuté touto na prvý pohľad značne neefektívnou praktikou sú viac ako pozoruhodné. Štúdie na univerzitách a skúsenosti z praxe ukazujú, že kvalita kódu tvorená práve v pároch je nielen oveľa vyššia, ale čo je takmer nepochopiteľné je fakt, že dvaja ľudia v páre urobia tú istú prácu asi o 20% rýchlejšie ako dvaja ľudia samostatne. [1]

Ďalšou nezanedbateľnou výhodou tohto prístupu je, že programátori, ktorí väčšinou majú rôzne úrovne znalostí či skúseností pracujúci v pároch sa tak neustále vzdelávajú. Jeden radí tomu druhému, je mu oporou. Na druhej strane sa zvyšuje sebaistota každého z nich, lebo vedia, že ten druhý dáva pozor a nedovolí mu spraviť nejakú hlúposť.

Projektoví manažéri si chvália tento prístup aj z iného dôvodu. Tvrdia, že ak pridajú do tímu nových ľudí, tak čas, ktorý je potrebný na to, aby začali byť produktívni sa výrazne skraca. Rádovo z niekoľkých mesiacov na týždeň. [1]

Na niekoľkých univerzitách boli dokonca pokusy vyučovať programovanie pomocou XP, ukázalo sa však, že táto praktika predpokladá určité zručnosti v programovaní a tak výučba programovania ako takého nie je úplne vhodná. Ak však študenti už majú nejaké skúsenosti s návrhovými vzormi a refaktoringom, môžu sa týmto štýlom veľmi efektívne zdokonaľovať napríklad v ich použití.

Ďalšie pokusy hovoria o tom, že párové programovanie je dobré len na riešenie určitých problémov. Tvrdia, že ak ide o zložitý problém s nejasným riešením je dobré programátorov párovať, naopak pri tvorbe napríklad používateľských rozhraní je to zbytočné plytvanie ľudskými zdrojmi. [2]

Ukazuje sa taktiež, že mentalita programátorov je v súčasnosti natoľko zdeformovaná sólovým štýlom programovania, že adopcia párového programovania väčšinou trvá nejaký ten čas. Netreba sa však na začiatku zľaknúť neefektívnosťou. Väčšina programátorov totiž neskôr hodnotí párové programovanie ako veľmi rozumný a hlavne omnoho zábavnejší spôsob práce.

Spoločné vlastníctvo kódu

XP vraví, že zdrojový kód môže meniť kdokoľvek a kedykoľvek chce. Zodpovednosť za produkt nesie celý tím. Je úplne zrejmé, že bez predpokladu dobrého testovania by sa toto mohlo veľmi rýchlo skončiť katastrofou. To, čomu sa, ale týmto XP snaží vyhnúť je, že danému modulu či triede bude rozumieť len jeden človek. Nemalú váhu v tom nesie práve párové programovanie. Keďže partneri sa v párovom programovaní

úplne bežne striedajú, je jasné, že každý programátor sa vo svojej práci stretne s množstvom kódu, ktorý nepísal, ale je nútený ho dobre pochopiť.

Nemôže sa teda stať, že ak odíde či ochorie nejaký programátor, tak po sebe zanechá kód, ktorému nikto nerozumie. Na druhej strane niektoré agilné metódy ako SCRUM zavádzajú presný opak. Tvrdia, že ak je za danú triedu či modul zodpovedný jeden človek, vytvorí sa medzi ním a zdrojovým kódom akýsi citový vzťah, ktorý ho núti sa o kód starať a neustále ho zlepšovať. Zástanci XP argumentujú tým, že je to veľmi neefektívne, lebo ak je potrebné pridať novú funkcionálnosť, tak musia vyhľadať vlastníka triedy a vysvetliť mu o čo vlastne ide. Stratia pri tom oveľa viac času ako keby si to sami implementovali.

Zákazník na pracovisku

V XP sa súčasťou tímu počas vývoja stáva i skutočný užívateľ alebo zákazník. Nemyslí sa tým však komunikácia cez email či telefón. Je to človek, ktorý je na plný úväzok „zamestnaný“ vo vývojovom tíme, odpovedá na otázky, pomáha vytvárať akceptačné testy a určuje priority. XP pripúšťa, že takéhoto človeka si nemôže dovoliť uvoľniť každá firma len tak, avšak argumentuje tým, že ak vyvíjaný systém nemá pre firmu ani cenu práce, ktorú by človek tam odpracoval, tak je najlepšie systém ani nezačať vyvíjať.

Kritici XP tvrdia, že firmy budú posielat' na takéto miesto neskúsených pracovníkov, aby ušetrili peniaze. Neskúsení pracovníci však nemajú jasnú predstavu o potrebnej funkcionálnosti produktu a ten teda v konečnom dôsledku nemôže byť kvalitný. Praktické skúsenosti s XP však ukazujú, že tomu tak nie je, pretože zákazníci tento problém veľmi dobre chápu a na takéto miesta posielajú skúsených a kvalitných pracovníkov.

Štandardy písania zdrojového kódu

V XP sa kladie najväčší dôraz na samotný zdrojový kód. Pri vývoji totiž nevzniká prakticky žiadna dokumentácia či špecifikácia. Všetky požiadavky sú viac menej priamo zapísané kódom do akceptačných testov, väčšina komunikácie prebieha nad zdrojovým kódom a preto je pre XP mimoriadne dôležité udržiavať ich v dobre čitateľnej, štruktúrovanej a pre všetkých zrozumiteľnej forme. Pri metódach, kde vznikajú dokumentácie to nie je také kritické, ale vo všeobecnosti sa to veľmi odporúča.

Adopcia agilných praktík

Praktiky, na ktorých je celé XP založené sú navzájom silno previazané. Kvalitný kód nemôže existovať bez dobrých testov, dobrá architektúra systému nemôže emergovať bez použitia refaktoringu, produkt nemôže byť kvalitný bez spätnej väzby od zákazníka. Zdá sa, že každá praktika v XP má svoje jedinečné a nezastupiteľné miesto.

Problém je v tom, že adoptovať všetky praktiky XP naraz sa pre vývojový tím môže zdať ako veľký skok do neznáma. Je to určite tak a výsledky takto začínajúceho tímu budú asi ďaleko od tých očakávaných. Ak by sa však vynechala alebo zanedbala nejaká kľúčová praktika, tak sa zase požadovaný výsledok nemusí dostaviť vôbec. Tento nedostatok pripúšťajú aj samotní tvorcovia XP pričom tvrdia, že pre dobre disciplinované tímy nebýva táto fáza až taká kritická.

Ak potom časom tím zistí, že nejaká praktika XP pre ich konkrétne projekty nefunguje, netreba sa báť vyskúšať ju vynechať. Ak sa to osvedčí v praxi, nie je dôvod sa k nej vracieť. Treba to však skúšať aj naopak, nie je dôvod zahrnúť automatické testovanie len preto, že sa používa v agilných metodikách, ktoré ako celok pre projekt možno nie sú vhodné. Zástancovia XP navyše dobre vedia, že reálny život je plný zmien a hovoria, že každý tím by si mal svoju metódu programovania vlastne ušit' na mieru. Toto je práve cesta k uplatneniu XP a agilných metodík v ľubovoľnom projekte.

Použitá literatúra

1. Lan Cao, Kannan Mohan, Peng Xu: How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects. In: *Proceedings of the 37th Hawaii International Conference on System Sciences*.
2. M. Muller, W.Tichy: Case Study: Extreme Programming in a University Environment.

Annotation

Autopsy of extreme programming

Agile methodologies represented mainly by extreme programming are considered as very useful techniques in development of small projects in changing environments. This fact has developed into a misunderstanding that core practices of these methodologies are not appropriate for other projects and are mistakenly considered as sources of project failures. They are being rejected without even knowing their true meaning. Nowadays is being shown that some of these practices are so rational, that they can have some positive effect almost on any project. So it is essential to identify their true meaning and consequences they are causing to be able to use them and not to reject them completely.