Zabezpečenie kvality v softvérovom projekte, príprava testov, plán testovania a overovania vlastností produktu

Čo má byť skôr? Test alebo kód?

PETER ŠUPINA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
supina02@student.fiit.stuba.sk

Abstrakt. Tradičné metódy vývoja softvéru zaraďujú proces testovania súčiastok až po dokončení ich implementácie. To môže viesť k neskoršiemu odhaleniu chýb zanesených v zdrojovom texte programu a k zvýšeniu nákladov na odstránenie týchto chýb. Vývoj riadený testami (Test-driven development) je založený na vytváraní (zväčša automatizovaných) testov súčiastky ešte pred tým, ako je zapísaný samotný produktívny kód danej súčiastky. Táto esej rozoberá výhody a nevýhody vytvárania testov súčiastok pred ich implementáciou a zameriava sa najmä na využiteľnosť tejto metódy v projektoch menšieho rozsahu. V ďalšej časti sa nachádza porovnanie agilných metód vývoja softvéru so staršími modelmi životného cyklu softvéru a uvádza sa tu aj popis techník, ktorými je možné zabezpečiť kvalitu pri použití agilných metód vývoja.

Úvod

Softvérové inžinierstvo ako vedná disciplína prechádza už od svojho vzniku veľkým rozvojom. Na začiatku bola, a v pomerne veľkom rozsahu aj dnes je, tvorba softvéru pomerne chaotická činnosť a často je označovaná frázou „napíš kód a oprav ho“ [1]. Softvér je vyvíjaný bez nejakého väčšieho plánovania a je riadený momentálnymi krátkodobými rozhodnutiami. Podobným spôsobom som pristupoval k tvorbe softvéru aj ja (a predpokladám, že aj väčšina mojich súčasných kolegov) predtým, ako som sa začal hlbšie venovať oblasti softvérového inžinierstva. Takýto prístup v skutočnosti dobre fungoval na malých projektoch, avšak ako softvérový systém rástol, čoraz ťažšie bolo pridávať novú funkcionalitu. V takomto systéme sa potom samozrejme vyskytovalo veľa chýb a bolo veľmi ťažké ich opraviť.

Manažment v softvérovom inžinierstve, december 2006, s. 1-7.

Reakciou na túto skutočnosť bola snaha voviesť do vývoja softvéru určitú metodológiu. Tieto metodológie boli spočiatku zamerané najmä na disciplinovaný proces pri vývoji softvéru s cieľom vyvíjať softvér predvídateľnejším a efektívnejším spôsobom. Bolo to dosiahnuté vyvinutím podrobne popísaných procesov, ktoré boli silne podporované plánovaním, známym z ostatných inžinierskych disciplín. Vznikli tak *metodológie riadené plánom* (inžinierske metodológie) [1].

Metodológie riadené plánom sa používajú už dlhý čas, avšak nie sú často označované ako populárne. Sú často kritizované najmä pre ich prílišnú byrokratickosť, pretože pre dodržiavanie týchto metodológií je potrebné vykonávať veľké množstvo dodatočných činností a samotný vývoj sa týmto spôsobom spomaľuje. Z vlastných skúseností viem, že neskúseným vývojárom (a tiež pri nedostatočne presných požiadavkách zákazníka – častý prípad) sa môže ľahko stať, že nesprávne odhadnú množstvo úsilia, ktoré je potrebné venovať plánovaniu. Vtedy spravidla nastávajú dva prípady:

1. *Nedostatočné plánovanie*, kedy sa plánovaniu venuje málo zdrojov, čo nakoniec môže viesť späť k chaotickému vývoju;
2. *Prehnané plánovanie*, kedy plánovanie trvá príliš dlho, čo neskôr môže viesť k nedostatku času pri samotnom vývoji softvéru, a teda k zníženej kvalite výsledného produktu alebo k nedodržaniu termínu odovzdania systému (najmä vtedy, ak je termín na ukončenie projektu stanovený pred vypracovaním plánu – štandardný postup vo vyučovacom procese).

Túto vlastnosť sa v posledných rokoch snažia odstrániť *agilné metodológie*. Tieto sa snažia nájsť kompromis medzi vývojom bez podporného procesu a vývojom s príliš mohutným procesom – chcú ponechať iba také množstvo procesu, ktoré bude primerané k výhodám vyplývajúcim z jeho použitia. Najviditeľnejším rozdielom oproti inžinierskym metodológiám je menšie množstvo požadovanej dokumentácie; sú orientované viac na zdrojový kód programu a ten sa väčšinou stáva kľúčovou časťou dokumentácie. Ďalším rozdielom je, že sú skôr adaptívne ako prediktívne a sú zamerané skôr na ľudí ako na proces (odvíjajú sa najmä od skúseností vývojárskeho tímu a prispôsobujú sa týmto skúsenostiam).

Vzhľadom na menšie množstvo dokumentácie pri agilných metódach (a teda menšiu viditeľnosť procesu), veľa manažérov nie je veľmi nadšených, keď sa v podniku vraví o možnom nasadzovaní agilných metód. Veľké obavy majú najmä v súvislosti s nedostatkom informácií o zabezpečovaní kvality pri ich použití. Preto som sa rozhodol preskúmať aj túto oblasť. Ďalším dôvodom boli moje skúsenosti s plánovaním a chcel som zistiť, či je možné zabezpečiť kvalitu aj bez podrobného počiatočného plánovania.

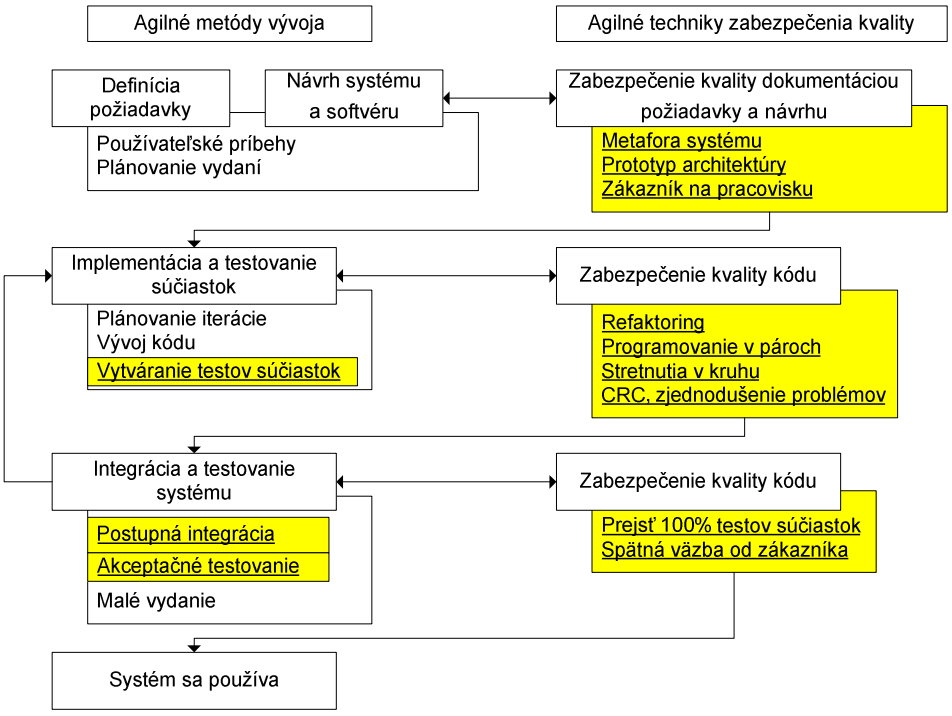
Zabezpečenie kvality pri použití agilných metód

Agilné metódy môžu viesť rýchlejšiemu vývoju softvéru, ale my potrebujeme vedieť aj to, či dokážu zvládnuť naše požiadavky na kvalitu. Ming Huo spolu s ďalšími autormi porovnáva vodopádový model (najstarší model procesu pre vývoj softvéru) a agilné

metódy (rozšírené až od konca 90-tych rokov 20. storočia) a sústreďuje sa na techniky, ktoré sa pri týchto metódach používajú na zabezpečenie kvality (Quality Assurance, QA) [2]. Vodopádový model ako zástupcu tradičných metód si vybrali preto, že je najstarší, najvyzretejší a hlavne je lineárnejší ako ostatné, čo im uľahčilo prezentovanie QA procesov. Skúmali tiež ako sú v týchto metódach zastúpené statické a dynamické typy QA techník a zistili, že kým vodopádový model používa aj statické aj dynamické techniky, agilné metódy sú zamerané najmä na dynamické techniky.

Vodopádový model pozostáva z piatich etáp: špecifikácia požiadaviek, systémový a softvérový návrh, implementácia a testovanie modulov, integrácia a testovanie systému, prevádzka a údržba. Každá z týchto etáp má dobre definované výstupy a keďže výstupy jeden etapy sú vstupmi do nasledujúcej etapy, nasledujúca etapa nemôže začať, kým neskončí predchádzajúca etapa a jej výsledky nie sú verifikované a validované podporným procesom danej etapy.

Agilné metódy vývoja majú s vodopádovým modelom spoločnú postupnosť vývoja, avšak implementácia a integrácia tu prebiehajú v krátkych cykloch (jeden až štyri týždne), čo neplatí pre klasický vodopádový model. Na **Obr. 1** ľavá časť zobrazuje hlavnú postupnosť agilných procesov a pravá časť zahŕňa agilné techniky, ktoré majú QA schopnosti.



Obr. 1: Agilné metódy a zabezpečenie kvality [2]

V agilných metódach existujú techniky, ktoré majú zároveň vývojovú funkciu a sú aj schopné zabezpečovať kvalitu. Tieto techniky sú na **Obr. 1** podčiarknuté. Popíšem iba niektoré z nich.

Zákazník na pracovisku (On-site customer) je spoločnou technikou vo väčšine agilných metód. Zákazníci pomáhajú vývojárom prispôsobiť a prípadne opraviť požiadavky na produkt. Vo vodopádovom modeli sú zákazníci zapojený iba v etape špecifikácie požiadaviek, prípadne v etape návrhu systému. To znamená, že pri agilných metódach je zákazník oveľa viac zapojený do procesu vývoja softvéru. Podľa mňa je to výhodné, pretože nakoniec to bude práve zákazník, kto bude používať systém a takto má možnosť systém dostatočne ovplyvniť. A je teda pravdepodobnejšie, že bude spokojný s výsledným produktom.

Refaktoring je technika pre zmenu štruktúry existujúceho kódu, pričom sa mení vnútorná štruktúra bez zmeny vonkajšieho správania. Prebieha v malých krokoch, transformáciách, ktoré zachovávajú správanie a keďže sú malé, aj pravdepodobnosť znefunkčnenia kódu je malá. Postupnosťou takýchto malých krokov je však možné docieliť zlepšenie prehľadnosti kódu a vďaka tomu znížiť pravdepodobnosť zanesenia chýb počas vývoja.

Postupná integrácia (Continuous integration) je populárna technika a znamená, že tím neintegruje kód raz alebo dva razy. Miesto toho tím potrebuje udržiavať systém plne integrovaný stále, takže integrácia prebieha aj niekoľkokrát za deň. Pomáha zavčasu odhaľovať chyby, najmä ak ide o chyby v rozhraniach medzi modulmi. Myslím, že aj keď to v každom kroku zaberie o niečo viac času, oveľa viac času sa ušetrí tým, že sa možné chyby neprenesú do ďalšej fázy vývoja, čo by viedlo k zvýšeniu nákladov a k spomaleniu v dôsledku ťažkého hľadania týchto chýb.

Akceptačné testovanie (*Acceptance testing*) sa vykonáva po úspešnom vykonaní všetkých testov súčiastok. Rozdiel medzi akceptačným testovaním v agilných metódach a tradičným akceptačným testovaním je ten, že tu sa vykonáva oveľa skôr a častejšie. Je jasné, že týmto spôsobom sa rýchlejšie zistí odklonenie od pôvodných požiadaviek a tiež sa ľahšie monitoruje, ktoré požiadavky sú už splnené (čo súvisí so sledovaním stavu projektu).

Autori [2] nakoniec dospievajú k nasledujúcemu výsledku:

1. agilné metódy obsahujú techniky s QA schopnosťami. Niektoré z nich sú priamo vo fáze vývoja, iné môžu byť oddelené ako podporné techniky;
2. frekvencia, s akou sa tieto agilné QA techniky používajú, je vyššia ako pri vodopádovom modeli;
3. agilné QA techniky sú dostupné už vo veľmi skorej fáze vývojového procesu.

Ukázali tým, že aj pri agilných metódach je kvalita zabezpečovaná, a teda, podľa mňa, obavy o kvalitu týmto spôsobom vytvoreného softvéru sú často prehnané.

Martin Fowler odporúča [1] vyskúšať si agilné metódy najprv na malých projektoch. Vysvetľuje tiež kedy je výhodné nasadiť ich:

1. ak požiadavky na produkt nie sú vopred dané, alebo je pravdepodobné, že sa budú často meniť;

2. ak máme k dispozícii spoľahlivých a motivovaných vývojárov;
3. ak sa zákazníci chcú viacej zapájať do procesu tvorby softvéru.

Vývoj riadený testami

Vývoj riadený testami (Test-driven development, TDD) je dôležitou technikou agilných metód. Pri používaní tejto techniky sú testy (pokiaľ možno automatizované) prírastkovo zapisované ešte predtým ako je implementovaný vlastný výkonný kód programu [3].

TDD bol ojedinele používaný už desaťročia. Prvá zmienka o použití TDD pochádza zo skorých 60-tych rokov, kedy bol využití NASA v „Project Mercury". TDD sa nedávno zviditeľnil ako dôležitý postup v metodike Extrémne Programovanie [3].

Pri použití TDD, vývojári, ešte pred implementáciou vlastného výkonného kódu programu, zapisujú automatizované testy súčiastok pre tú novú funkcionalitu programu, ktorú práve idú implementovať. Vývojári po zapísaní testov vytvárajú výkonný kód programu tak, aby úspešne prešiel týmito testami. Tento proces sa v zásade prirodzene prispôsobuje danej situácii. Vývojár napíše zopár testov, implementuje kód, napíše zopár testov, implementuje kód, atď. Vývojár je schopný túto prácu vykonávať, pretože postupne navrhuje a implementuje malé prírastky a rozširuje funkcionalitu systému v zvládnuteľnom tempe. Nová funkcionalita sa považuje za správne implementovanú až potom, keď testy novej súčiastky a všetky ostatné testy súčiastok (zapísané pre potreby predošlého kódu) skončia úspechom [3].

Po prijatí funkčných požiadaviek, softvérový inžinieri využívajúci TDD vyvíjajú výkonný kód, pričom používajú rýchle iterácie nasledujúcich činností [3]:

- zapísanie malého počtu automatizovaných testov súčiastok;
- vykonanie týchto nových testov súčiastok, aby sa uistili, že zlyhajú (pretože ešte nie je zapísaný žiadny výkonný kód);
- implementovanie kódu, ktorý by mal umožniť úspešné vykonanie týchto nových testov súčiastok;
- opakované vykonanie týchto nových testov súčiastok, aby sa uistili, že teraz s novým kódom prejdú;
- periodicky (napr. raz za deň za predpokladu, že rozsah kódu je dostatočne malý) opakované vykonávanie všetkých testov súčiastok, aby sa uistili, že nový kód nespôsobuje zlyhanie testov, ktoré predtým boli úspešné. Toto je samozrejme spätné testovanie.

Williams a spol. vo svojej štúdii [3] zisťujú možné výhody použitia TDD na týchto vlastnostiach:

- efektívnosť;
- prínosy testov;

– redukovanie vkladania chýb.

Svoj výskum robili na projektoch spoločnosti IBM. Porovnanie tímov a produktov je uvedené v **Tab. 1**.

	Pôvodný 7. iterácia	**Nový 1. vydanie**
Veľkosť tímu (vývojári)	5	9
Skúsenosti tímu (jazyk a doména)	Skúsení	Niektorí neskúsení
Umiestnenie	Usporiadaný	Rozložený
Rozsah kódu (KLOC) Nový; Pôvodný; Celkovo	**6,6**; 44,9; 56,5	**64,6**; 9,0; 73,6
Jazyk	Java/C++	Java
Testovanie súčiastky	Účelové	TDD
Technické vedenie	Zdieľaný zdroj	Nadšený školiteľ

Tab. 1: Porovnanie tímov a produktov

Ľudia neskúsení s TDD nakoniec dosiahli o 40% menšie množstvo chýb vo fáze verifikácie, ktorá nasledovala po testovaní modulov v porovnaní s tradičným spôsobom testovania modulov.

Myslím, že je vhodné porozmýšľať o nasadení testov modulov pred implementáciou výkonného kódu programu už len preto, že je možné redukovať množstvo zavedených chýb v skorej fáze.

Záver

V eseji som načrtol význam agilných metód v súčasnej dobe a ukázal som, že aj pre ne existujú techniky na zabezpečenie kvality. Zároveň som sa bližšie pozrel na TDD ako jeden možný spôsob na zlepšenie kvality kódu.

Do budúcnosti je možné lepšie preskúmať možnosť využitia TDD aj pri tradičných postupoch vývoja softvéru a zistiť, aké by to mohlo mať dôsledky.

Použitá literatúra

1. Fowler, Martin. 2005. *The New Methodology.* http://martinfowler.com/articles/newMethodology.html

2. Huo, M., Verner J., Zhu, L., Babar, M. A.: Software Quality and Agile Methods. In: *Proc. of the 28th Annual Intl. Computer Software and Applications Conference*, 2004, pp. 520-525.
3. Williams, L., Maximilien, E. M., Vouk, M.: Test-Driven Development as a Defect-Reduction Practice. In: *Proc. of the 14th Intl. Symposium on Software Reliability Engineering*, 2003, p. 34.

Annotation

Quality assurance in software project, test preparing, product test and evaluation plan

Traditional software development methodologies schedule the unit testing phase after the software implementation phase. That may lead to latter discovering of errors in the program source code and to rise in the error correction costs. Test-driven development is based on creation of (mostly automated) test cases even before writing of the unit's production code. This essay compares advantages and disadvantages of test-driven development and keeps focus on using this methodology in projects of small size. I also compare agile methods in software development with older models of software lifecycle and their quality assurance capability.