NECHAJTE MA PRACOVAŤ! TÚ CHYBU ODSTRÁNIM NESKÔR...

Práca na viacerých úlohách súčasne ma vždy presvedčí o tom, že paralelizmus v ľudskom mozgu jednoducho nefunguje.

Štefan Mitrík

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
stefan.mitrik[zavináč]gmail[.]com

Abstrakt. *Pri vývoji softvéru spravidla vznikajú chyby. Čas, ktorý máme k dispozícii na vývoj softvéru sa znižuje, ale požiadavky na kvalitu zostávajú rovnaké. Programátorské tímy, ktoré nerieši chyby, ktoré vznikajú pri vývoji softvéru systematicky, znižujú jeho kvalitu a zvyšujú čas potrebný na jeho vývoj. Pre tím je podstatné vedieť, kedy a kto má riešiť chybu. Ak je programátor pri práci na svojej úlohe prerušovaný inými úlohami, jeho produktivita klesá spolu s kvalitou jeho práce. Preto si úspešný tím nevyhnutne musí jasne zadefinovať zodpovednosti a postup pri odstraňovaní chýb. V eseji sa venujem technikám odstraňovania chýb špecificky v procese Scrumu.*

Kľúčové slová: *odstraňovanie chýb, Scrum, čas, paralelizmus*

Úvod

Rýchla doba v ktorej žijeme, mení nároky na vývoj softvéru. Čas určený na vývoj sa rapídne znižuje, ale nároky na kvalitu zostávajú na rovnakej alebo dokonca vyššej úrovni. Čoraz používanejšími sa stávajú agilné metódy, ktoré zvyšujú tlak na prioritizáciu práce a priebežné odovzdávanie fungujúceho softvéru. Jednou z populárnych agilných metód je aj Scrum, ktorý rozdeľuje úsilie do časových úsekov – šprintov.

V dobe, keď cena údržby softvérových systémov niekoľkonásobne prevyšuje cenu jeho vývoja [1], sa stále venuje málo pozornosti chybám. Takmer každý, kto programoval, sa presvedčil o tom, že odhadnúť čas na naprogramovanie určitej funkcionality je náročné a ešte náročnejšie je odhadnúť čas potrebný na odstránenie chyby. Pri chybách obvykle

nevieme ich jasnú príčinu a problém, ktorý sa môže zdať na prvý pohľad jednoduchý, môže v skutočnosti súvisieť s inou ako predpokladanou časťou programu a jeho zložitosť môže byť omnoho vyššia.

Programátorské tímy sú okrem toho často pod tlakom, aby vyvíjali novú funkcionalitu a odstraňovanie chýb sa v záujme dokončovania úloh v šprinte odsúva *„na neskôr“*. Odsúvanie odstraňovania chýb však so sebou prináša celú škálu rizík a problémov počnúc časom, ktorý je potrebný na manažovanie chýb [6], až po fakt, že čím viac chýb sa v softvéri vyskytuje, tým náročnejšie je ich odstraňovanie [8].

Na druhej strane snaha o okamžité odstránenie nájdenej chyby vedie k znižovaniu koncentrácie programátora na úlohy v šprinte. Môže sa tak ľahko stať, že programátor odstráni chybu, ale kvôli časovému stresu na dokončenie úlohy v šprinte vytvorí ďalšie dve. Tento cyklus vedie k frustrácii programátora a môže viesť k zúfalým zvolaniam typu: *„Nechajte ma pracovať! Tú chybu odstránim neskôr!“*.

Ako by sme to chceli

Predstavme si stav, v ktorom by sme vedeli presne odhadnúť čas potrebný na odstránenie chyby a vedeli by sme objektívne určiť jej prioritu. Zdá sa, že takéto informácie by nám mohli stačiť na to, aby sme odstraňovanie chýb správne zaradili do plánov.

S týmto cieľom vzniklo niekoľko systémov [2,3], ktoré sa snažia o ohodnotenie času potrebného na odstránenie chyby vychádzajúc z existujúcich chýb a časov potrebných na ich odstránenie. Tieto systémy však vyžadujú rozsiahle databázy chýb a ich presnosť často nie je dostatočná.

Odhadovanie priority chyby je podobne ako odhadovanie času na jej odstránenie netriviálna úloha. Vo väčšine prípadov totiž nevieme, aké iné časti systému chyba ovplyvňuje a či je prepojená s inými chybami. Môže sa tak stať, že chyba, ktorá sa zdá byť nepodstatnou je naviazaná na iné oveľa vážnejšie chyby.

Stačí nám to?

Povedzme, že vieme relatívne presne určiť čas potrebný na odhadnutie chyby a jej prioritu, no tieto informácie nám stále nestačia. Treba ešte určiť, kto sa má odstraňovaním chyby zaoberať a kedy ho máme touto úlohou poveriť.

Odstraňovanie chýb nie je práve najpopulárnejšia úloha, preto nie je jednoduché určiť programátora zodpovedného za jej odstránenie. Ďalším problémom je šprint, do ktorého odstránenie chyby zaradiť. Priradiť ho programátorovi v jeho aktuálnom šprinte a ohroziť tak dokončenie stanovených úloh? Alebo vyčleniť v každom šprinte programátora, ktorý sa bude výlučne venovať odstraňovaniu chýb a podpore produkčnej verzie? Alebo odsunúť odstránenie chyby so zdanlivo nízkou prioritou na neskôr, prípadne tzv. „čistiaci“ šprint?

Správne zodpovedanie týchto otázok odlišuje vynikajúcich lídrov scrum tímov a ich tímy od priemerných a zlých. Okrem pohody v tíme a spokojných programátorov správne riešenie týchto problémov vedie k zvyšovaniu kvality softvéru.

Mozog vs. procesor

Už je to nejaký čas, čo procesory podporujú viac úlohové spracovanie a vytvárajú dojem, že aj na jednom jadre procesora dokážu obslúžiť súčasne niekoľko programov. Zabezpečujú to veľmi rýchlym prepínaním úloh a tak sa zdá, že programy bežia súčasne. Skúsme sa pozrieť, ako by to vyzeralo, keby čas potrebný na prepnutie medzi úlohami trval minútu. Počítač by väčšinu času trávil prepínaním úloh a používateľ by smútil za časmi, keď sa výpočty riešili sekvenčne.

Pozrime sa teraz na to, ako dlho trvá programátorovi prepnutie z jednej úlohy do inej. Pri programovaní programátor drží v hlave množstvo informácii od názvov premenných, až po funkcie knižnice, ktorú práve využíva. Pri zmene úlohy, na ktorej pracuje, sa musí zahĺbiť do novej úlohy a čas, ktorý je na to potrebný, je priamo úmerný veľkosti úloh. Podľa autora tejto úvahy môže prepnutie do inej komplexnej úlohy trvať až 6 hodín [4].

Je teda zrejmé, že aj keby sme vedeli čas potrebný na odstránenie chyby, stále nevieme, koľko produktívneho času zaberie programátorovi kvôli času potrebnému na zahĺbenie sa do problému a spätnému zahĺbeniu sa do úlohy, na ktorej pracoval predtým. Je ideálne, ak vývojári pracujú na jednej úlohe a nerozptyľujú sa riešením iných problémov.

Techniky odstraňovania chýb

Dva zoznamy úloh

Jedným zo spôsobov riešenia chýb v scrume je vytvorenie dvoch zoznamov úloh. Jeden je štandardný a vkladajú sa doň používateľove príbehy súvisiace s novou funkcionalitou, do druhého sa pridávajú objavené chyby. Vlastník produktu spolu s vedúcim tímu určia pomer v akom sa budú úlohy vyberať zo zoznamov úloh, napríklad pomerom 70:30.

Toto riešenie so sebou prináša niekoľko problémov. Prvým je prioritizácia úloh, ktorá je jednou zo základných charakteristík scrumu. Ľahko nastane stav, že napriek tomu, že máme požiadavky s vyššou prioritou v jednom zozname úloh, vyberieme úlohu z druhého zoznamu úloh napriek jej nižšej priorite. Ďalším problémom je to, že je náročné odhadovať úsilie na odstránenie chyby v bodoch pre používateľove príbehy.

Myslím si, že táto technika je vhodná pre skúsenejšie tímy, ktoré sú schopné vykonávať presnejšie odhady. Takisto by bolo vhodné vytvoriť variabilný pomer, ktorý by sa menil pred každým šprintom. Určilo by sa napríklad, že pomer môže byť v rozsahu 70:30 až 90:10, týmto by sa dal odstrániť problém prioritizácie, pretože pomer by sa menil v prospech zoznamu úloh, ktorý má úlohy s vyššou prioritou.

Dva tímy

McOwen považuje za najlepšie riešenie *„rozdelenie“* scrum tímu na dve časti [5]. Väčšia časť sa plne venuje vývoju novej funkcionality a nerozptyľuje sa ničím iným, menšia časť sa venuje výlučne odstraňovaniu chýb a podpore produkčnej verzie. Vývojári sa v úlohách striedajú po každom šprinte.

Autor uvádza, že riešenie je vhodné aj pre malé tímy zložené napríklad len z dvoch členov. Podľa môjho názoru to nie je realizovateľné pre malé tímy. Prvým problémom je množstvo chýb, ktoré treba odstraňovať. Ak sa napríklad zo štyroch členov tímu jeden plne venuje len odstraňovaniu chýb, je zrejmé, že programátori robia pri vývoji priveľa chýb a treba venovať zvýšenú pozornosť testovaniu.

Iným problémom je odstraňovanie chýb v cudzom kóde. Myslím si, že vývojár vo svojom vlastnom kóde dokáže chybu odstrániť za rádovo kratší čas ako iný vývojár. Na druhej strane treba dodať, že práve vďaka tomu sa môže eliminovať kód, ktorému rozumie len jeden vývojár.

Menej ako N

Autor článku „Managing Bugs with Scrum" [6] uvádza, že chyby s nízkou prioritou ostávajú často nevyriešené. Nájdené chyby, ktoré súvisia s príbehmi používateľa v aktuálnom šprinte, by sa mali pridať do zoznamu úloh tohto šprintu, pretože indikujú nedokončenú prácu. Naopak, chyby, ktoré nie sú priamo späté s príbehmi používateľa v aktuálnom šprinte, by mali ísť do produkčného zoznamu úloh. Tým je zabezpečené to, že programátori sa môžu sústrediť na úlohy v aktuálnom šprinte.

Riešenie problému odstraňovania chýb vidí autor v tom, že sa určí maximálny počet chýb N, ktorý môže byť v produkčnom zozname úloh. Číslo N by malo byť dostatočne nízke na to, aby tím mohol bez väčších problémov odstrániť chyby v priebehu niekoľkých dní.

Hlavné plus tohto prístupu považujem v tom, že ak sa zvolí maximálny počet chýb N ako dostatočne nízke číslo, je produkt pripravený na uvedenie do prevádzky v relatívne krátkej dobe.

Toto riešenie však nezohľadňuje prioritu chýb. Do každého šprintu sa vyberajú chyby podľa priority tak, aby ich počet v zozname úloh bol nižší ako N. Môže sa teda ľahko stať, že chyba, respektíve skupina chýb s nízkou prioritou bude sprevádzať produkt až do záverečných fáz vývoja. Tu vidím problém v tom, že ak sa neodhadne správne priorita týchto chýb a teda sú závažnejšie ako sa predpokladalo, môže sa v záverečných fázach zistiť, že tieto chyby negatívne ovplyvňujú veľkú časť zdanlivo hotového kódu.

Určitým vylepšením tejto techniky by mohlo byť zahrnutie veku chyby do procesu jej prioritizácie, teda čím staršia chyba, tým vyššia priorita.

Čistiace šprinty

Čistiace šprinty sa obvykle vykonávajú pred odovzdaním časti systému zákazníkovi alebo pred uvoľnením novej verzie aplikácie pre používateľov. Napriek tomu, že teória scrumu hovorí, že systém by mal byť pripravený na odovzdanie po každom šprinte, v praxi sa dosť často stáva, že počas predchádzajúcich šprintov vznikli časti neodladeného kódu, respektíve kódu, ktorý obsahuje chyby.

V čistiacom šprinte by mal tím odladiť existujúce časti systému a odstrániť chyby, ktoré sa nahromadili v zozname úloh. Je zrejmé, že časté čistiace šprinty indikujú nedostatočné testovanie pridávanej funkcionality, čo obvykle súvisí s nedostatkom času na dokončenie úloh. Čistiace šprinty sú tak určitým spôsobom výmenou za včasné

dokončenie funkcionality v šprintoch. Nebolo by teda vhodné zadávať vývojárom v šprintoch menej úloh, ktoré by tak mohli byť poriadne otestované a odladené?

Odpoveď na túto otázku je pochopiteľné pozitívna. Problémom je už spomínaný problém s odhadom. Väčšina tímov sa prirodzene snaží zadať do šprintu toľko úloh, koľko je podľa ich odhadu schopná zvládnuť, odhady sú však obvykle nepresné, navyše programátori robia chyby. Riešením by sa teda mohlo zdať vytvorenie dostatočného časového vankúša, vďaka ktorému by sa mohli v rámci šprintu riešiť neočakávané problémy. Tu však narážame na problém s vlastníkom produktu, ktorý obvykle vyvíja tlak na pridávanie novej funkcionality.

Okrem toho sa požiadavky na softvér aj počas vývoja menia [7], preto môže byť verzia produktu s menšími chybami po dvoch týždňoch hodnotnejšia ako odladená verzia po troch týždňoch. Zákazník tak vidí verziu produktu skôr a môže tak vyjadriť prípadnú požiadavku na zmenu.

Pri čistiacich šprintoch by malo v princípe platiť pravidlo, čím skúsenejší a zrelší tím, tým lepšie odhady a tým menej čistiacich šprintov. Odhady však nikdy nie sú dokonalé a preto sú z môjho pohľadu čistiace šprinty užitočné za predpokladu, že sa nevykonávajú príliš často.

Existuje ideálna technika?

Vidíme, že existuje celá škála techník a postupov na odstraňovanie chýb v scrume. Existuje však technika, ktorá by bola naozaj ideálna a najefektívnejšia?

Odpoveď je áno aj nie. Vo všeobecnosti sa nedá určiť technika, ktorá by sa dala považovať za najlepšiu. Súvisí to či už s rozmanitosťou tímov, špecifikami spoločností a konkrétnych projektov, ale aj prispôsobovaním a adaptovaním scrumu jednotlivými tímami. Na druhej strane je na každom tíme, aby si zvolil techniku a vybral si to, čo najlepšie vyhovuje tímu ako celku. Príkladom môže byť spoločnosť Google, v ktorej si každý scrum tím prispôsobuje a upravuje scrum tak, aby bola dosahovaná čo najvyššia produktivita a spokojnosť v tíme.

Komunikácia a jasne zadefinované pravidlá

Čo teda robiť, aby sa odstraňovanie chýb zbytočne neodsúvalo a vývojári nemali pocit, že sú rušení pri práci na svojich úlohách?

Riešením je, ako to už býva pri mnohých iných problémoch, komunikácia. Tím *musí* komunikovať, analyzovať a vyhodnocovať svoju doterajšiu činnosť. Na základe toho je možné vybrať techniku alebo kombinácie techník, ktoré sa zdajú byť najvhodnejšie. Je nevyhnutné, aby sa vybrala technika ktorá vyhovuje *každému* členovi tímu. Aj po výbere techniky treba dbať na spätnú analýzu a tím by si ju mal podľa potrieb ďalej modifikovať.

Ďalšou podmienkou úspešného riešenia chýb sú jasne zadefinované pravidlá a zodpovednosti. Ak napríklad programátor nájde chybu v kóde, mal by vedieť kam ju má nahlásiť a na základe zvolenej techniky by mal vedieť, kto a kedy ju má riešiť. Jasne zadefinované pravidlá pomáhajú predchádzať frustrácii, pretože každý člen tímu vie, čo môže očakávať a vie, ako sa má v danej situácii správať.

Je výzvou pre nás všetkých, akýmkoľvek spôsobom spojených s informačnými technológiami, aby sme okrem svojich odborných zručností nevyhnutných na tvorbu kvalitného softvéru rozvíjali aj komunikačné a sociálne zručnosti a naučili sa efektívne pracovať v tímoch. Práve tieto zručnosti odlišujú dobré a výborné tímy a pomáhajú im zefektívňovať riešenie celej škály problémov, vrátane problému odstraňovania chýb v scrume.

Použitá literatúra

1. Boehm B., Basili, V.R.: Software Defect Reduction Top 10 List, Computer, (2001), 135-137.
2. Zeng, H., Rine, D.: Estimation of Software Defects Fix Effort Using Neural Networks. In: *Proc. of COMPSAC Workshops*. (2004), 20-21.
3. Weiss, C., Premraj, R., Zimmermann, T., Zeller, A.: How long will it take to fix this bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories,* IEEE Computer Society, Washington, DC, USA, (2007), 1.
4. Spolsky, J.: Human Task Switches Considered Harmful [online] [cit. 2011-11-25] <http://www.joelonsoftware.com/articles/fog0000000022.html> (2001)
5. McOwan, T.: Handling live support in a Scrum team [online] [cit 2011-11-25] <http://www.devballs.com/2010/09/handling-live-support-in-a-scrum-team/> (2010)
6. Miller, A.: Managing Bugs with Scrum [online] [cit 2011-11-25] <http://www.ademiller.com/blogs/tech/2008/10/managing-ugs-with-scrum/> (2008)
7. Rising, L., Janoff, N.S.: The Scrum software development process for small teams, *Software, IEEE* , Vol. 17, No. 4, (2000), 26-32.
8. Fowler, M.: Continuous Integration [online] [cit 2011-11-25] <http://martinfowler.com/articles/continuousIntegration.html> (2006)

Annotation

Let me work! I'll fix the bug later...

In the software development process, bugs generally cannot be avoided. While the time available for software development often decreases rapidly, the quality requirements remain the same. Teams of programmers, who do not systematically fix bugs, lower the quality of the software and increase the time needed for development. For a team, it is important to know when and who is responsible for fixing each bug. The productivity of a programmer who is being disturbed while working on his task lowers along with the quality of his work. That is why a successful team needs clearly defined responsibilities and procedures for debugging. In my essay, I focus on debugging techniques used specifically in the Scrum process.