SCRUM - EVOLÚCIA ČI REVOLÚCIA VO VÝVOJI SOFTVÉRU?

Často nazývame tento fenomén ako softvérová kríza, no úprimne, choroba trvajúca takto dlho, už musí byť normálna. (Grady Booch)

Peter Kajan

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
xkajanp1[zavináč]is[.]stuba[.]sk

Abstrakt. *Agilný spôsob vývoja Scrum patrí medzi tie najlepšie a čím ďalej je populárnejšia a používanejšia v praxi. Produkty sú dodávané načas, čo nielen prekvapuje zákazníka, ale aj uspokojuje večne nespokojný manažment. Boli urobené mnohé štúdie a napísaných mnoho prác o tom, aké výborné výsledky dosahuje postup, pri ktorom ťahá jeden za všetkých a všetci za jedného. Je to teda najlepší spôsob vývoja softvéru? Má vôbec nejaké negatívne stránky? Do doby vzniku Scrumu bolo navrhnutých mnoho metodológii vývoja, pričom každá priniesla istý posun vpred, no žiadna nepriniesla dramatický skok vpred. Fenomén nazývaný softvérová kríza však prelomí len revolúcia vo vývoji softvéru. Preto si kladiem otázku: Je Scrum evolúcia či revolúcia v softvérovom inžinierstve?*

Kľúčové slová: *vývoj softvéru, Scrum, problémy vývoja*

Úvod

Dve vlastnosti softvéru spôsobujú manažmentu problémy s jeho plánovaním. Je to jeho zložitosť a neviditeľnosť, vďaka ktorým je plánovanie netriviálne. Zložitosť je ovplyvňovaná zložitosťou cieľového a vývojového prostredia. Napríklad pri vývoji medicínskeho softvéru je potrebné si osvojiť relevantné medicínske procesy. Avšak, takisto treba brať ohľad na dostupnosť vývojárov a iných zdrojov. Proces vývoja je taký zložitý, že

ho nie je možné kompletne a dopodrobna opísať. Táto vlastnosť známa ako neviditeľnosť softvéru robí jeho plánovanie netriviálnym. Pokusy navrhnúť metodológiu, ktorá dopodrobna opíše a presne naplánuje vývoj, zlyhali.

Napriek tomu, že tento proces je neúplne definovaný, boli navrhnuté pomerne detailné metodológie vývoja. Každá z nich priniesla istý posun vpred a vylepšenie produktivity, ale žiadna z nich nepriniesla dramatický skok vpred. Ako poznamenal Grady Booch: „Často nazývame tento fenomén ako softvérová kríza, no úprimne, choroba trvajúca takto dlho, už musí byť normálna.“[4] Vyzerá, že metodológie vývoju softvéru čakajú na revolúciu, no zatiaľ ide o prirodzenú evolúciu. Priniesla metodológia inšpirovaná rugby dramatickú zmenu alebo sme sa len zase pohli iba krôčik vpred?

Evolúcia vo vývoji

Vodopádový model, ako prvý pokus opísať proces vývoja, sa ukázal byť vhodný len na malé projekty. Podľa [4] problém je v tom, že sa predpokladá, že tento nedefinovaný proces je lineárny a vôbec nerieši ako odpovedať na náhlu zmenu v špecifikácii či prostrediach. Takisto neponúka riešenie na neočakávaný výstup niektorého z jeho podprocesov.

Vylepšenie mala priniesť Špirálová metodológia, ktorú autor [4] prirovnáva k lúpaniu cibule. Softvér sa vyvíja postupným prechádzaním cez jednotlivé vrstvy. Po ukončení každej etapy vznikne prototyp, na základe ktorého sa rozhodne, či sa projekt uberá správnym smerom, vráti sa do jednej z predchádzajúcich fáz alebo v najhoršom prípade úplne zruší.

Po Špirálovej metodológii prišla Iteratíva. Jej hlavnou myšlienkou je vývoj systému po častiach, z ktorých každá obsahuje jednotlivé podprocesy Vodopádového modelu. Rozdelenie na etapy a časté prototypovanie síce zlepšuje pružnosť, no v tomto prístupe sa stále predpokladá, že jednotlivé procesy sú definované a lineárne [4].

Problém spomínaných metodológií je, že nedostatočne riešia problém nepredvídateľnosti prostredí a podprocesov. Zmeny prostredí sú neustále a výstupy podprocesov sú nepredvídateľné, na čo je potrebné dostatočne rýchlo reagovať a treba sa tomu okamžite prispôsobiť. Je známe, že neprežijú najsilnejší, ale tí najprispôsobivejší a evolúcia vyradí tých, ktorí sa izolujú od zmeny prostredia [4].

Scrum

Tento prístup predpokladá, že analýza, návrh a vývoj sú nepredvídateľné. Zavádza kontrolné mechanizmy na zvládnutie rizík a neočakávaných situácii, výsledkom čoho je pružnosť, spoľahlivosť a schopnosť okamžitej odozvy na zmeny oboch spomínaných prostredí. Na rozdiel od iteratívneho vývoja, v ktorom je vývojársky tím schopný reagovať na zmeny iba na začiatku jednotlivých iterácii, v Scrume je reakcia okamžitá a prispôsobovanie neustále [4].

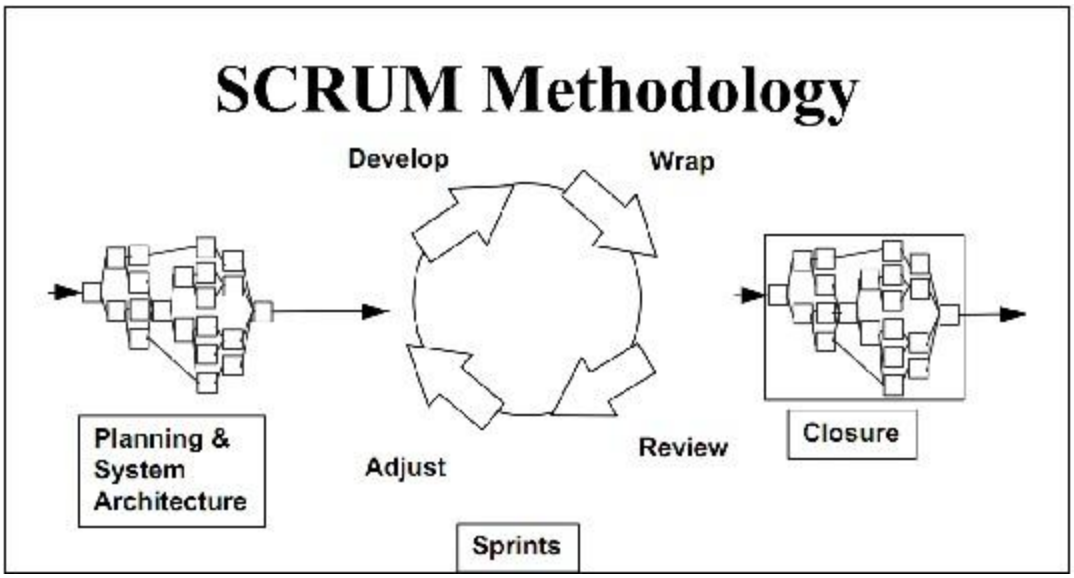
Scrum bol inšpirovaný anglickou tímovou hrou rugby. Názov je podľa rovnomennej formácie, v ktorej sa tím na povel zomkne, čo je aj jednou z hlavných myšlienok metodológie. Vývojársky tím musí byť schopný sa zopnúť a jeden druhému pomôcť vždy keď je to potrebné. Vývojári operujú na vyhradenom hracom ihrisku (v danom prostredí) a

hrajú podľa stanovených pravidiel (zoznam funkcionálnych požiadaviek, atď). Snažia sa posúvať loptu stále ďalej vpred (pridávať novú funkcionalitu). Ukončenie hry závisí od prostredia (požiadavky, konkurencia, funkcionalita, čas...) a nie od jednotlivých členov tímu. Vznik Scrumu bol podmienený potrebou adaptovať sa prostrediu podobne ako rugby. To vzniklo z futbalu tak, že hráč nedokázal zniesť prehru, zobral loptu a utekal s ňou k súperovej bráne. Porušil pravidlá, prispôsobil sa [4].

Proces vývoja pozostáva z niekoľkých fáz (pozri Obr. 1). Prvá fáza pozostáva z plánovania a návrhu. Najdôležitejší výstup plánovania je zoznam funkcionalít systému. Stanoví sa dátum odovzdania produktu, odhadnú sa náklady a stanovia mechanizmy kontroly rizík. Potom nasleduje návrh, kde sa už premýšľa nad tým, ako budú jednotlivé funkcionality implementované. Návrh sa uskutočňuje na vysokej úrovni, pričom podrobnejší sa robí priebežne [4].

Druhá fáza je samotný vývoj softvéru, ktorý sa uskutočňuje v iteráciach - šprintoch. Táto fáza nekončí pokiaľ vývojári nevytvoria produkt, ktorý spĺňa podmienky na odovzdanie [4].

Ak manažment rozhodne, že požiadavky, kvalita a čas súhlasia, prejde projekt do poslednej fázy - nasleduje uzatvorenie projektu. Pripravuje sa odovzdanie produktu, tvorí sa finálna dokumentácia, robí sa finálne testovanie a píšu sa príručky. Dôsledkom tejto fázy je odovzdanie produktu [4].



Obr. 1. Fázy vývoja metodológie Scrum [4].

Z hlavných charakteristík Scrumu podľa [4] chcem spomenúť nasledovné:

1. pružný výstup – finálny produkt sa nestanoví na začiatku, ale počas procesu vývoja.

2. pružný plán – plán sa vytvára na začiatku každého šprintu, no je možné ho počas týchto iterácii upravovať, napríklad presúvaním úloh do ďalších šprintov.

3. malé tímy – podľa mnohých prác je Scrum vhodný iba pre malé vývojárske tímy, no podľa môjho názoru, niektoré princípy by mohli inšpirovať aj tie väčšie.

4. spolupráca – podobne ako rugby je v Scrume vývoj softvéru tímová hra. Členovia by mali postaviť potreby tímu pred svoje vlastné, aj keď to znamená strávenie celého dňa dávaním rád nováčikovi.

Porovnanie metodológii

Scrum je navrhnutý pre flexibilitu, čím odpovedá na rýchle zmeny prostredia. Umožňuje zmenu výstupu kedykoľvek počas vývoja a prináša najadekvátnejší výstup [4]. Tímy sú zomknuté, členovia úzko spolupracujú, čím sa zdieľajú vedomosti. Toho dôsledkom je perfektné prostredie na zaúčanie nováčikov, či prehlbovanie vedomostí skúsenejších.

Vývoj týmto spôsobom je intenzívny, pri vývoji vládne tímový duch. V porovnaní s iteratívnym spôsobom vývoja je schopný okamžite reagovať na zmenu v prostredí, zvyšuje pružnosť tímu, neobmedzuje jeho kreativitu. Porovnanie Scrumu s ostatnými prístupmi približuje tabuľka 1 [4].

Tab. 1. Porovnanie metodológii [4]

	Vodapádový	Špirálový	Iteratívny	Scrum
Definované procesy	potrebné	potrebné	potrebné	Iba plánovanie a uzatvorenie
Finálny produkt	Stanovený počas plánovania	Stanovený počas plánovania	Stanovený počas projektu	Stanovený počas projektu
Náklady na projekt	Stanovené počas plánovania	Stanovené počas plánovania	Stanovené počas projektu	Stanovené počas projektu
Schopnosť odozvy na prostredie	Iba počas plánovania	Hlavne počas plánovania	Na konci každej iterácie	Kedykoľvek
Flexibilita a kreativita tímu	Limitovaná (cookbook aproach)	Limitovaná (cookbook aproach)	Limitovaná (cookbook aproach)	Nelimitovaná počas iterácii
Získanie vedomostí	Školenie	Školenie	Školenie	Tímová práca počas projektu
Pravdepodobnosť úspechu	Nízka	Nízka až stredná	Stredná	Vysoká

Každodenné problémy

Práca [2], poukazuje na niekoľko problémov spozorovaných v praxi, pri monitorovaní každodenných procesov. V spomínanej práci, ako prvý nedostatok uviedli nekonzistentnosť terminológie v rámci tímov, ktorá bola spôsobená práve pružnosťou a dynamickými zmenami prostredia. Tento problém by bolo možné jednoducho riešiť zaradením procesu stanovenia terminológie do procesu vývoja, napríklad na začiatok každého šprintu alebo častejšie, podľa stupňa dynamickosti prostredia.

Ďalším spozorovaným problémom v spomínanej práci je nedisciplinovanosť tímu. Pod tím sa myslia značné rozdiely medzi predpokladaným časom a skutočným časom splnenia úlohy a zlé resp. chýbajúce vytváranie nových úloh. Autori navrhujú zaviesť metriku alebo nástroj na zavedenie istého poriadku do empirického procesu vývoja. Podľa môjho názoru toto je problém jednotlivcov v tíme a ich postoja. Ak berú prácu a vývoj seriózne, disciplína je na postačujúcej úrovni. No z praxe viem, že tím môže z veľkej časti len predstierať vývoj v Scrume, aby uspokojil manažment. Tím sa stavia k vývoju nezodpovedne, práve z čoho môžu prameniť vyššie spomenuté problémy. V tomto prípade má navrhovaný nástroj podľa môjho názoru svoje opodstatnenie [2].

Záhadný kód či všeobecná vina

V Scrume väčšinou jednu funkcionalitu systému vyvíja jeden človek. Pozitívne na tom je, že táto osobe je zodpovedná za kód tejto funkcie. Nevzniká tu syndróm všeobecnej viny, ktorý často zľahčuje situácie. Vývojár môže byť jednoduchšie odmenený alebo aj postihovaný. Táto kvázi výhoda však oponuje hlavnej myšlienke tímového ducha Scrumu.

Nežiadúcim dôsledkom zodpovednosti jednotlivca je, že môže nastať prípad, kedy iba jeden vývojár rozumie časti kódu, technológiam či funkcionalite. Čo potom keď tento člen tímu dlhodobejšie ochorie alebo jednoducho odíde z firmy?

Na druhej strane pri extrémnom programovaní (hlavne pri programovaní v pároch) znalosti kódu jednotlivých členov tímu sa podstatne prelínajú, kód je to spoločné dielo. Síce tu vzniká spomínaný syndróm spoločnej viny, no je to podľa mňa menší problém, ako keď kódu nerozumie nikto. Tento nedostatok Scrumu sa môže riešiť jeho kombináciou s Extrémnym programovaním. Mnohé práce tvrdia, že tieto prístupy sa dopĺňajú a toto zlúčenie prístupov odporúčajú.

Po dvadsiatich rokoch

V Scrume sa vytvára málo štandardnej dokumentácie, väčšinou iba tá nevyhnutná. Ako zdroj vedomostí v tejto metodológii je odporúčaná častá komunikácia, veľká miera spolupráce vývojárov a zdokumentovaný zdrojový kód. Podľa [3] treba zlepšiť proces získavania znalostí a nespoliehať sa len na vedomosti tímu, či dokumentáciu. Ďalej si môžeme klásť otázky: Ako je to s udržiavaním projektov? Je možné udržiavať takýto projekt aj dlhé obdobie?

Štúdia [3] skúmala projekty trvajúce cez 20 rokov. Tvrdí, že aj počas takto dlhého obdobia sa vedomosti udržali, a to pomocou:

1. žijúcich dokumentov (living documents),
2. dobrej komunikácie (well-connected communication) a
3. prototypov (exemplar software systems).

Žijúce dokumenty

Sú to ľudia, ktorí pracujú pre firmu viac ako 20 rokov, takzvaní pionieri (pioneer employees) [3]. Tu si treba uvedomiť, že zamestnávatelia sa síce implicitne snažia udržať si zamestnancov čím dlhšiu domu, no udržať si zamestnancov (samozrejme nie všetkých) viac ako 20 rokov nie je jednoduché. Podľa môjho názoru spoliehať sa, že sme si schopní udržať si zamestnancov takto dlhé obdobie, je priveľké riziko.

Dobrá komunikácia

Dobrá komunikácia medzi vývojármi je štandardná pre agilné metódy, no treba sa uistiť či naozaj funguje. Ak tím pozostáva z uzavretých ľudí neschopných komunikovať či dobre vychádzať s ostatnými, čo je v IT pomerne bežné, dobrá komunikácia zaručene fungovať nebude.

Prototypy

Posledný zdroj vedomostí sú prototypy zahrňujúce okrem spustiteľnej aplikácie a aj zdokumentovaný zdrojový kód. Podľa autorov [3] spustiteľná aplikácia značne napomáha pochopeniu procesov. Ďalej dôsledkom získavania vedomostí z predtým implementovaného softvéru spôsobuje, že sa tento softvér nezahadzuje, ale znovu sa použije. Kód tohto softvéru musí byť čitateľný a komponent musí byť napísaný tak, aby bol znovupoužiteľný. Z praxe viem, že kód býva neprehľadný a zle zdokumentovaný a vývojárov napísať naozaj znovupoužiteľný softvér je málo.

Sme zodpovední k prostrediu?

Scrum sa pýši tým, že mu nerobí problém prispôsobiť sa zmenám prostredia. Manažment a zákazník vo veľkej miere ovplyvňujú proces vývoja počas celej doby jeho trvanie. No sú vývojári dostatočne zodpovední k prostrediu? Táto metóda, podobne ako ostatné sa zameriavajú na to, ako vyvinúť softvér a nekladú už veľký dôraz na to, ako bude pôsobiť v rámci systému interagujúceho s rôznymi ľuďmi. Autor článku [1] poukazuje na fakt, že vývojári, hlavne v agilných metódach, sa snažia hlavne uspokojiť objednávateľa, no neberú priveľký ohľad na ostatných.

Napríklad pre zákazníka vyvinieme webovú aplikáciu presne podľa jeho požiadaviek. Avšak dyslektik, ktorému robí problém prečítať bezpätkové písmo na čisto bielom pozadí, je účastník (stakeholder), s ktorým sa pri vývoji nepočítalo. Často do vývoja zasahuje iba úzke spektrum ľudí, a keď sa snažíme vyhovieť iba ich požiadavkám, často sa na ostatných zabudne. Autor spomínanej eseje kritizuje fakt, že táto špecializácia je v praxi veľmi bežná [1].

Je potrebné, aby metódy vývoja brali ohľad aj na dôsledky softvéru v rámci prostredia. Rád by som spomenul reálny prípad toho, kedy softvér môže spôsobiť vážne škody. Firma blízka tej, v ktorej pracujem, vyvíjala softvér, ktorý mal okrem iného vypočítavať dávky röntgenového žiarenia vyžarovaného na pacienta. Pri pretypovaní celého čísla na reálne, prišlo v určitých prípadoch k malej zmene hodnoty čísla určujúceho veľkosť dávky. V tomto prípade tá zmena však nebola bezvýznamná a spôsobila, že odvtedy pacient má zvýšené riziko vzniku nádoru.

Často sa vývojári riadia pravidlom: „Myslite v rámci škatule! (Don´t think outside the box!)“, čo súvisí napríklad aj s enkapsuláciou v objektovo-orientovanej paradigme. No kým nebude do procesu vývoja zaradených viac používateľov ako len zákazník a kým sa nebude viac dbať na zodpovednosť softvéru voči prostrediu, je potrebné aby sme mysleli aj mimo škatule [1].

Záver

Zložitosť a neviditeľnosť softvéru spôsobujú jeho nepredvídateľnosť, čo implikuje značné problémy s plánovaním vývoja. Na to aby bolo plánovanie úspešnejšie vznikali metodológie vývoja, ktoré sa postupom času evolvovali a prispôsobovali sa trendom. V dnešnej dobe sú rozšírené mnohé agilné metódy. Dnes medzi najpoužívanejšie v praxi patrí Scrum.

Podľa mňa najdôležitejšia idea tejto metodológie je súdržnosť tímu a jeho snaha dosiahnuť spoločný cieľ. Tímový duch v práci zlepší vzájomné vzťahy a vytvorí výbornú atmosféru. V tíme ľahšie vzniknú priateľstvá. V takomto prostredí vývoj softvéru stane viac než prácou a možno sa jedného dňa prichytíme, že sa tešíme do roboty.

Ďalej si myslím, že Scrum by sa mal kombinovať s extrémnym programovaním, ktoré podporuje myšlienku súdržného tímu. Vďaka programovaniu v pároch nováčikovia rýchlo zapadnú a kód sa stane spoločné dielo celého tímu. Súčasťou extrémneho programovania je aj refactoring, ktorý okrem iného pomôže udržať kód čitateľný a ten môže slúžiť ako zdroj informácii aj o 20 rokov.

V neposlednom rade fakt, že nielen Scrum, ale celkovo metodológie vývoja, často spôsobia vznik softvéru nezodpovedného k prostrediu. Podľa mňa je to vážny problém a jeho riešenie je netriviálne, ale potrebné.

Podľa môjho názoru je Scrum jedna z najlepších dnešných metodológii vývoja, najmä ak sa vhodne skombinuje s extrémnym programovaním. Jedným dychom však dodávam, že si treba byť vedomý jeho nedostatkov a snažiť sa, aby sa neprejavili v projektoch, na ktorých pôsobíme. Výsledky tímov vyvíjajúcich týmto prístupom dosahujú výborné výsledky, čo potvrdzuje, že tento prístup spravil veľký krok v evolúcií metodológii vývoja. Podľa môjho názoru nie sme ďaleko od revolúcie.

Použitá literatúra

1. Gotterbarn, D. 2004. UML and agile methods: in support of irresponsible development. *SIGCSE Bull*. 36, 2 (Jun. 2004), 11-13.

2. Ktata, O. and Lévesque, G. 2010. Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?. In *Proceedings of the Third C* Conference on Computer Science and Software Engineering* (Montréal, Quebec, Canada, May 19 - 20, 2010). C3S2E '10. ACM, New York, NY, 101-107. DOI= http://doi.acm.org/10.1145/1822327.1822341
3. Ratanotayanon, S., Kotak, J., Sim, S.E.: After the Scrum: Twenty Years of Working without Documentation. *CiteCeerX*.
4. Rising, L. and Janoff, N. S. 2000. The Scrum Software Development Process for Small Teams. *IEEE Softw.* 17, 4 (Jul. 2000), 26-32. DOI= http://dx.doi.org/10.1109/52.854065

Annotation

Scrum – evolution or revolution in software development

Agile software development method called Scrum is one of best and as time goes by it's getting more and more popular and used in commercial sphere. Products are done in time which surprises customer and satisfies never satisfied management. There have been made many studies about how good the results are, if procedure where everybody goes for one is used. So is this the best way of the software development? Does this method have also some negatives? There have been developed many software development methodologies until the origin of the Scrum. All of them made little step forward in software development but none of them made bigger jump. Phenomenon called software crisis could be break only by revolution in the development. That's why I'm asking myself question: Is Scrum evolution or revolution in software engineering?