Wstęp do Scrum

Giovanni Asproni

Scrum jest jedną z najbardziej znanych metodologii *agile*. Została opracowana przez Kena Schwabera i Jeffa Sutherlanda około roku 1993 (data pierwszego Scrum opisanego przez Jeffa Sutherlanda w „*Agile Development: Lessons Learned from the First Scrum*" – patrz sekcja *resources* na stronie *ScrumAlliance* [11]). Scrum przynosi wiele korzyści klientom – ludziom lub organizacjom, które są założycielami projektu – oraz użytkownikom. Iteracyjna i inkrementalna natura tej metodologii, połączona z zarządzaniem priorytetami wymagań, pozwala na wczesne uzyskanie istotnych elementów, przy czym najważniejsze z nich – po pierwszej iteracji. Co więcej, Scrum pozwala klientom i użytkownikom uzyskać całkowitą kontrolę nad kierunkiem i zakresem prac projektu. Na końcu każdej iteracji mogą zdecydować o kontynuacji projektu – poprzez dodanie lub modyfikację funkcjonalności – albo o zakończeniu projektu, jeśli jest zadowalającej jakości, bądź nie jest już im potrzebny. Scrum przynosi korzyści menedżerowi projektu dostarczając narzędzia – wykres malejący (j.ang. *burn-down chart*), wykaz prac produktu (j.ang. *product backlog*), wykaz prac sprintu (j.ang. *sprint backlog*) – oraz techniki – codzienne zebrania, spotkanie planowania sprintu (j.ang. *sprint planning meeting*), spotkanie przeglądu sprintu (j.ang. *sprint review meeting*) – ukierunkowane na zwiększenie możliwości obserwacji każdego aspektu projektu. Pozwala to na większą kontrolę nad projektem i wcześniejsze wykrywanie problemów, które mogą się zdarzyć podczas trwania projektu. Wreszcie, Scrum zarówno dobrze się sprawdza w małych zespołach – siedem plus-minus dwóch programistów [15] – jak i dobrze się skaluje na większe projekty. Zresztą ta technologia była stosowana w projektach angażujących nawet do kilkuset programistów [14]. Jej implementacja czasem jednak może być trudna – Scrum, jak pozostałe metodologie *agile*, silnie bazuje na pracy zespołowej, komunikacji, zaufaniu oraz przydzielaniu odpowiedzialności i władzy. Te wszystkie sprawy wymagają poważnych zmian w przyzwyczajeniach, szczególnie dla organizacji, które nasiąkły bardziej tradycyjnymi metodami. Pełna akceptacja tych zmian wymaga czasu i ciężkiej pracy. W tym artykule – po wprowadzeniu do podstaw tej metodologii – przedyskutuję niektóre z tych problemów, włącznie z paroma sugestiami na ich rozwiązanie.

Manifest Agile

Termin *Agile Software Development* to worek, do którego można wrzucić wszystkie metodologie spełniające system norm i wartości ustanowionych przez *Manifest Agile* (patrz ramka). Najważniejszą cechą tych metodologii jest skupianie się na ludziach: na pracy zespołowej, komunikacji bezpośredniej i szybkiej wymiany informacji, w odróżnieniu od bardziej tradycyjnych, które – dla odmiany – skupiają się na procesie, na kolejności wykonywanych czynności i stosowanych narzędziach [2].

Najważniejszym celem metodologii *agile* jest dostarczenie korzyści wszystkim uczestnikom, włącznie z programistami. Korzyści dla klienta zdają się być oczywiste: oprogramowanie, które spełnia przyjęte założenia, jest zrobione na czas i mieści się w przewidzianym budżecie. Korzyść dla programistów jest znacznie mniej oczywista: motywacja i satysfakcja z wykonanej pracy. Faktem jest, że stosowanie metodologii *agile* może wiele pomóc w motywowaniu programistów [2], co jest najważniejszym czynnikiem wpływającym na ich produktywność [6], a to z kolei bezpośrednio wpływa na korzyści dla klientów. Jak zostanie pokazane w dalszej części artykułu, Scrum stosuje się do wszelkich norm i wartości ustanowionych w Manifeście Agile.

Podstawowe założenia Scrum

Proces Scrum, jak inne metodologie *agile*, jest iteracyjny – produkt jest wytwarzany w następstwie realizacji kolejnych mini-przedsięwzięć zwanych iteracjami – oraz inkrementalny – funkcjonalność produktu rośnie poprzez nowe właściwości, dodawane kolejno podczas każdej iteracji. Posiada też własną terminologię, zarówno dla ról zaangażowanych osób, jak i dla niektórych czynności procesowych. Przedstawię je wstępnie w następnych dwóch sekcjach. Szybki przegląd procesu przedstawia Rysunek 1.

Role

W procesie Scrum określa się jedynie trzy role, jakie mają otrzymywać uczestnicy projektu: właściciel produktu, szef scruma, oraz członek zespołu. Właściciel produktu jest odpowiedzialny za decyzje w sprawie cech funkcjonalnych produktu oraz zarządzanie priorytetami w ich implementacji. Reprezentuje on interesy wszystkich ludzi zaangażowanych w projekt i finalny produkt – klientów, użytkowników itd. Często tę rolę otrzymuje ktoś z zespołu marketingu, albo kluczowy użytkownik. Ta rola jest podobna do roli klienta w Manifeście Agile.

Szef scrum jest odpowiedzialny za egzekwowanie praktyk i zasad Scrum, reprezentowanie zarządców wobec projektu, „ekranowanie" zespołu i usuwanie przeszkód – upewnianie się, między innymi, że każdy członek zespołu posiada odpowiednie zasoby sprzętu i oprogramowania, stanowisko pracy itp. Często ta rola jest przydzielona kierownikowi tech-

Kontakt z autorem: *gasproni@asprotunity.com*

Manifest Agile

Manifest Agile został zaczerpnięty z [5] Manifestu *Agile Software Development*.

Odkrywamy lepsze sposoby na tworzenie oprogramowania poprzez robienie tego i pomaganie w tym innym. Poprzez tę pracę osiągnęliśmy korzyści:

- Ludzie i współpraca ponad proces i narzędzia;
- Działające oprogramowanie ponad wyczerpującą dokumentacją;
- Współpraca z klientem ponad negocjacją kontraktu;
- Reagowanie na zmiany ponad trzymaniem się planu.

Czyli, choć elementy po prawej mają swoją wartość, cenimy bardziej elementy po lewej.

nicznemu zespołu lub kierownikowi projektu, a czasem właścicielowi produktu.

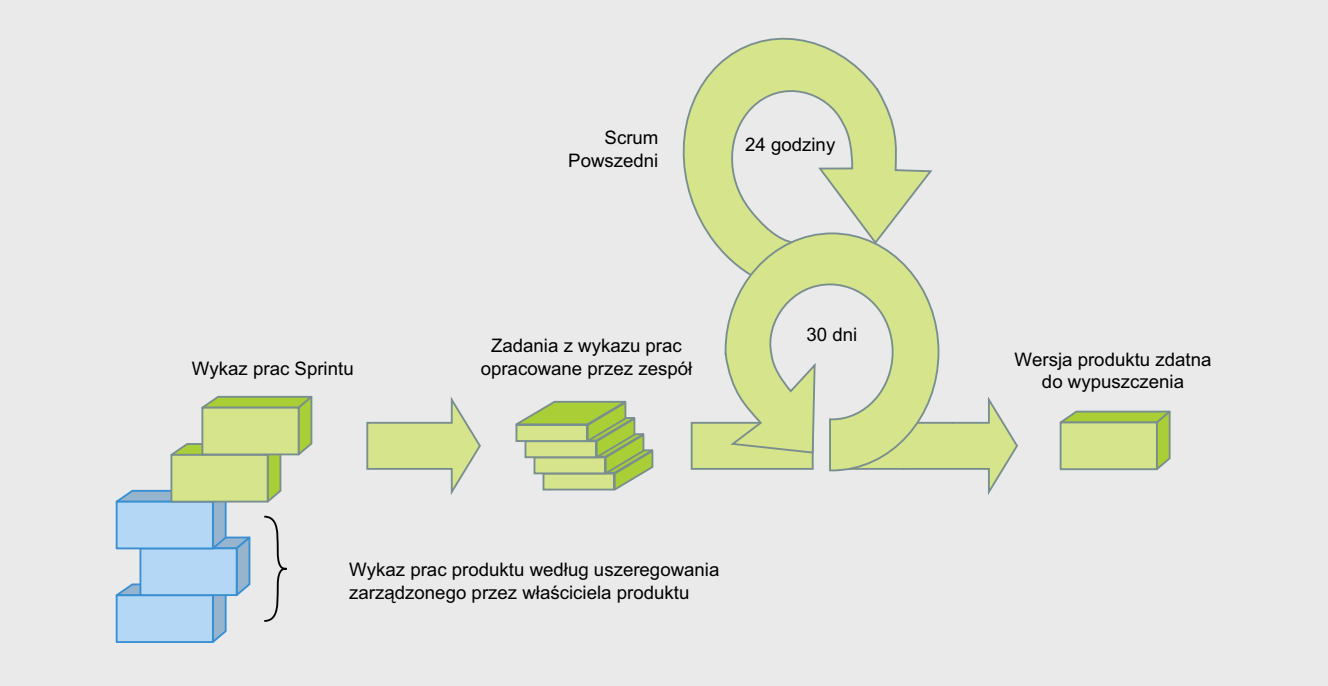
Członkiem zespołu jest każdy, kto należy do zespołu wykonującego czynności związane bezpośrednio z wytwarzaniem oprogramowania. Członkowie zespołu pełnią różnorakie funkcje – w przypadku aplikacji webowych mogą do niego należeć projektanci stron, programiści, testerzy, administratorzy baz danych, administratorzy systemu, projektanci techniczni itd. Zespół ten jest zespołem samoorganizującym się – jego członkowie sami decydują o postępach prac nad dostarczeniem produktu bez ingerencji z zewnątrz. Członkowie zespołu, w idealnym przypadku, nie mają przydzielonych tytułów (takich jak architekt, kierownik techniczny itd.); zamiast tego współpracują na równych zasadach. Każdy może być zaangażowany w dowolną z czynności – projektowanie, programowanie, tworzenie dokumentacji itd. – niezależnie od swojego zakresu ekspertyzy.

Członkostwo w zespole powinno angażować na cały czas. Co jednak nie zawsze jest możliwe, szczególnie przy takich rolach, jak administrator systemu, czy baz danych, które są zazwyczaj dzielone pomiędzy poszczególnymi zespołami wewnątrz organizacji. Moim ulubionym podejściem jest posiadanie w zespole członków „pełnoetatowych" plus – w razie potrzeby – zewnętrznych dla zespołu specjalistów „częściowo-etatowych", dostępnych jako pomoc ekspercka. Zaleca się zawsze wszystkim zaangażowanym w projekt Scrumowy, aby nie podejmowali się więcej, niż jednej roli jednocześnie. Powód jest taki, że każda z ról ma przypisany inny zestaw odpowiedzialności, przez co może nie współgrać z inną rolą. Na przykład, właściciel produktu mógłby próbować wymusić na zespole programistów pracę w nadgodzinach, żeby spełnić niemożliwy *deadline*, w którym to przypadku szef Scruma powinien poczuć się częścią zespołu i bronić go przed takim nadużyciem. Jednak takie rozdzielenie nie zawsze jest możliwe. W tym przypadku radzenie sobie z potencjalnymi konfliktami pozostawia się zdrowemu rozsądkowi zaangażowanych osób.

Czynności procesowe i narzędzia

W tym rozdziale objaśnię terminologię dotyczącą elementów procesu Scrum. Pierwszym ważnym terminem jest *sprint*, który jest tylko innym określeniem iteracji. Sugerowany czas trwania sprintu jest trzynaście dni kalendarzowych, niemniej zwykle można znaleźć zespoły pracujące z iteracjami jedno- lub dwutygodniowymi. Ważne jest, żeby raz ustalony czas trwania pierwszego sprintu był następnie taki sam dla pozostałych sprintów, co nada zespołowi odpowiedni rytm i uprości zarówno zarządzanie, jak i śledzenie czynności procesowych.

Na początku każdego sprintu organizowane jest spotkanie planowania sprintu, które jest zajęciem jednodniowym, podzielonym na dwie części. Pierwsza część to czterogodzinna sesja planowania, na której właściciel produktu tworzy tzw. wykaz prac sprintu (zestaw czynności do wykonania w bieżącej iteracji). Jest to lista czynności najwyższego priorytetu w ilości takiej, jaką zespół może wykonać w bieżącym sprincie. Czynności te wybierane są z wykazu prac produktu, czyli listy wszystkich czynności, które należy wykonać w celu wytworzenia produktu. Wykaz prac produktu może się zmieniać w czasie (i zwykle się zmienia), kiedy dochodzą nowe wymagania, bądź też modyfikuje się lub usu-



Rysunek 1. *Cykl Scrum*

wa istniejące.Po utworzeniu wykazu prac sprintu, właściciel produktu i zespół określają tzw. cel sprintu (j.ang. *sprint goal*), czyli biznesową korzyść, jaką zmiany w produkcie muszą dostarczyć, niezależnie od implementowanej funkcjonalności. Jego przeznaczeniem jest określić, na czym programiści winni się skupić oraz jakie możliwości wybierać w przypadku problemów lub niepewności. Cel sprintu powinien być mierzalny, dzięki czemu można łatwo określić, kiedy został osiągnięty. Na przykład, celem dla aplikacji webowej dla danego sprintu może być „Obsługiwać dwukrotnie więcej połączeń, niż w wersji 2.0".

W drugiej części, zespół dzieli wykaz prac sprintu na poszczególne *zadania* – jednostki pracy szacowane na cztery do sześciu godzin – do wykonania w celu dostarczenia wymaganej funkcjonalności. Lista tych zadań niekoniecznie musi być kompletna, bo wiele zadań może pojawić się gdy sprint już się zaczął. Właściciel produktu nie uczestniczy w tym spotkaniu, ale musi być dostępny, żeby odpowiadać na ewentualne pytania zespołu.

Spotkanie planowania sprintu jest ograniczone czasowo do ośmiu godzin, a pierwsza część jest ograniczona do czterech godzin. W tym miejscu nie od rzeczy jest wspomnieć, że Scrum jest bardzo ścisły pod względem ograniczeń czasowych. Jeśli upłynął czas przeznaczony dla danej czynności, to musi ona zostać zakończona – cztery godziny to dokładnie dwieście czterdzieści minut i ani minuty dłużej. Powód tego jest taki, że ludzie mają się skupić na tym, co jest ważne, i nie marnować czasu na sprawy drugorzędne. Codziennie o tej samej porze, w tym samym miejscu, oraz najlepiej jeszcze przed rozpoczęciem pracy, zespół powinien zrobić sobie *spotkanie na dzień dobry* (j.ang. *stand-up* meeting; zwany inaczej *scrumem powszednim*) trwające co najwyżej pięćdziesiąt minut – niezależnie od wielkości zespołu – na którym każdy członek zespołu odpowiada na trzy pytania: Co robiłeś wczoraj?; Co będziesz robić dzisiaj?; Co ci stoi na przeszkodzie?

Spotkanie służy jedynie synchronizowaniu (nie rozwiązywaniu problemów). Wszelkie ważne sprawy są rozpatrywane po zakończeniu spotkania. Uczestnictwo w tym spotkaniu jest otwarte dla każdego, kto jest zainteresowany projektem. Uczestnicy są podzieleni na dwie grupy. Pierwsza, czyli świnki, to ludzie należący do zespołu programistów, a reszta stanowi drugą grupę, czyli kurczaki [14]. Podczas spotkania tylko świnki mówią, podczas gdy kurczaki mogą jedynie po cichu obserwować. Zastosowanie tej reguły powoduje, że spotkanie jest krótkie, ludzie są skupieni na istotnych sprawach, a uczestnicy projektu są doinformowani na temat postępu prac i napotkanych problemów.

Dla członków zespołu uczestnictwo w tym spotkaniu jest obowiązkowe. Jeśli któryś z członków nie może w nim z jakichś powodów uczestniczyć, inny członek zespołu powinien zdać za niego sprawozdanie. W czasie trwania sprintu, na koniec każdego dnia pracy, każdy członek zespołu aktualizuje tzw. wykres malejący (j.ang. *burn-down chart*) – wykres przedstawiający ilość pracy do wykonania w bieżącym sprincie (patrz Rysunek 2.) – wraz z szacowaną ilością pracy pozostałej do wykonania w zadaniach, nad którymi pracował w ciągu dnia. W ten sposób łatwo będzie zobaczyć, czy sprint idzie pomyślnie, czy też wystąpiły jakieś opóźnienia wymagające korekty planów.

Na koniec sprintu organizuje się *spotkanie przeglądu sprintu* (j.ang. *sprint review meeting*), na którym zespół przedstawia właścicielowi produktu, co osiągnięto w ostatniej iteracji. Potem właściciel produktu określa, czy cel sprintu został osiągnięty. Czas trwania tego spotkania jest ustalony na cztery godziny.

Zasady rządzące Manifestem Agile

Priorytetową sprawą jest osiągnięcie zadowolenia klienta poprzez wczesne i nieprzerwane dostarczanie mu oprogramowania wysokiej jakości. Nie ma się co sprzeciwiać częstym zmianom wymagań, nawet na późniejszym etapie. W procesie *agile* restrykcyjność zmienia się na korzyść klienta. Należy dostarczać działające oprogramowanie często, od kilku tygodni do kilku miesięcy, z tendencją raczej do krótszych okresów czasu. Ludzie interesu i programiści muszą pracować razem codziennie w całym projekcie. Projekt winien być oparty o umotywowanych ludzi. Należy dostarczyć im środowisko pracy i wszelkie wsparcie oraz zaufać im, że wykonają swoją pracę.

Najbardziej skuteczną i wydajną metodą dostarczania i wymiany informacji w zespole programistycznym jest bezpośrednia rozmowa. Działające oprogramowanie jest głównym miernikiem postępu. Ludzie finansujący projekt, programiści oraz użytkownicy powinni móc śledzić postęp stale i bez ograniczeń. Ciągłe skupianie się na jakości technik i ulepszaniu projektu zwiększa jego zwinność (j.ang. *agility*).

Upraszczanie, jako sztuka maksymalizacji ilości nie wykonanej pracy, jest sprawą zasadniczą. Najlepsze architektury, wymagania i projekty powstają w samoorganizujących się zespołach. W regularnych odstępach czasu zespół winien rozważać, jak zwiększyć swoją wydajność i zgodnie z tym zmieniać swoje metody pracy.

Zaraz po spotkaniu przeglądu sprintu organizuje się spotkanie retrospektywne sprintu (j.ang. *sprint retrospective meeting*), trwające trzy godziny, na którym zespół i szef scruma rozmawiają o tym, które zadania i czynności zostały wykonane należycie podczas ostatniego *sprintu* i jak można usprawnić następny. Właściciel produktu nie uczestniczy w tym spotkaniu.

Z kolei, jeśli podczas sprintu okazuje się, że występują szczególnie uciążliwe problemy z realizacją postawionych zadań lub też cel sprintu okazał się już nieistotny, szef scruma lub właściciel produktu ma prawo zarządzić *nadzwyczajne zakończenie* sprintu. Innymi słowy, Sprint zostaje przerwany (j.ang. *aborted*) oraz zostaje podjęta akcja naprawienia problemu, po czym organizuje się nowy sprint, najpewniej z nowym wykazem prac (j.ang. *backlog*) i nowym celem.

Wprowadzanie do Scrum

Na początku jest projekt, właściciel produktu, szef scruma i zespół – z technicznego punktu widzenia to wygląda prosto:

- Właściciel produktu określa początkowy wykaz prac produktu, plan edycji produktu, budżet itd.;
- Zespół, wraz z szefem scrum i właścicielem produktu, decydują o czasie trwania iteracji (i tego się trzymają);
- Planowana jest pierwsza iteracja: określa się wykaz prac sprintu, cel sprintu, zadania – po czym każdy udaje się do swoich zadań.

Tego typu plany nie zawsze są łatwe do zrealizowania w praktyce, szczególnie gdy uczestniczą w tym ludzie.

Przede wszystkim, w zespole zakładanym od zera dopiero po jakimś czasie zostanie osiągnięta równowaga. Nim to nastąpi, członkowie zespołu będą się starali zaznaczyć swoją wagę wobec całego zespołu, powodując tym problemy we współpracy. Jest to normalne i właściciel produktu oraz szef zespołu powinni pohamować swoje zapędy do interwencji porządkowych, a tak-

eZ components

Enterprise PHP platform

eZ components is an enterprise ready general purpose PHP platform. As a collection of high quality independent building blocks for PHP application development eZ components will both speed up development and reduce risks.

- Designed for enterprise PHP application development
- Open source and licensed under the New BSD license
- Clear IP rights
- Thoroughly documented
- Developed, supported and maintained by eZ systems

www.ez.no

eZ publish conference 2006

Skien, June 21-23

eZ systems

że odmówić tego w razie próśb. Zespół powinien sam wypracować sobie równowagę bez wpływów z zewnątrz. W gruncie rzeczy bowiem każda zewnętrzna pomoc jedynie utrudnia zespołowi samoorganizowanie się. Innym problemem, jakże powszechnym w wielu organizacjach, jest oporność na zmiany. Jeśli się chce wprowadzić Scrum w organizacji, która stosuje już inną metodykę, lub nie stosuje żadnej, można z dużą dozą prawdopodobieństwa napotkać sprzeciw tych, którzy czują się przez jej wprowadzenie zagrożeni. W tym wypadku, dobrze jest rzucić okiem na [8], w którym autorzy prezentują wzorowy język do wprowadzania nowych idei o organizacji. Udało mi się zastosować niektóre z technik opisanych w tej książce, nim poznałem je jako wzór [1]. Na domiar powyższych problemów, każda rola ma też swoje wyzwania.

Szef Scrum powinien pohamować swe zapędy do rządzenia zespołem, nawet jeśli, co się niekiedy zdarza, członkowie zespołu tego sobie życzą – czasem żądają jego interwencji w sprawach, w których powinni sobie radzić sami. Właściciel produktu, tak jak i szef Scrum, winien trzymać na wodzy pokusę rządzenia zespołem, który może organizować się zupełnie inaczej, niż by on oczekiwał, oraz pokusę dodawania „ważniejszych spraw" do wykazu prac sprintu, gdy sprint już wystartował. Jeśli próbuje to robić, szef scruma i zespół winni się mu postawić. Jeśli nowe cechy są tak ważne, że dodanie ich jest nieodzowne, właściciel produktu zawsze może zarządzić nadzwyczajne zakończenie sprintu.

Innym wielkim wyzwaniem dla właściciela produktu jest radzenie sobie z organizacją priorytetów poszczególnych prac z wykazu. Faktem jest bowiem, że decydowanie o tym, co ma wejść do sprintu i z jakim priorytetem, to ogromna odpowiedzialność. Szczególnie dla kogoś, kto jest nowy w Scrumie, lub w ogóle pierwszy raz ma do czynienia z procesem iteracyjno-inkrementalnym, może to być stresujące. Przecież w końcu – jak głosi pewien schemat myślowy – wszystkie wymagania mają najwyższy priorytet i wszystkie muszą być spełnione w produkcie końcowym. W takim przypadku zespół i szef Scruma powinni udzielić mu pomocy w określaniu poprawnych priorytetów poprzez zadawanie pytań, tak żeby właściciel mógł spokojnie przemyśleć, które z tych bardzo ważnych wymagań mają być spełnione w pierwszej kolejności. Ta sprawa może być trudna, ale po kilku sprintach, jeśli zespół będzie w stanie dostarczyć jakąś korzyść na koniec każdego z nich, właściciel produktu będzie bardziej zadowolony i sprawa stanie się prosta. W pozycji [1], nawet jeśli nie jest ona akurat o Scrumie, można poczytać, jak mój zespół i ja radziliśmy sobie z tego typu problemami w rzeczywistym projekcie, w którym uczestniczyłem.

Generalnie zresztą dla człowieka na stanowisku menedżerskim – kierownika projektu, kierownika technicznego itd. – jedną z największych barier do pokonania w przypadku wdrażania Scruma może być odczuwalny brak kontroli nad projektem. Dla wielu ludzi przekazywanie władzy i ufanie innym jest naprawdę wyzwaniem z uwagi na obawy przed utratą możliwości wglądu, kontroli, oraz czasami osobistej siły. Okiełznanie tych obaw nigdy nie jest łatwe; zwykle wymaga ciężkiej pracy i czasu, a także nie ma w tej kwestii żadnych murowanych recept na sukces. Książka Mary Lynn Manns i Lindy Rising może dostarczyć nieco dobrych pomysłów na radzenie sobie z tym problemem [8]. Co do członków zespołu, należy też uwzględnić fakt, że nie każdy się do Scrum nadaje; ludzie często nie lubią być pozbawiani swoich tytułów i zaliczani do szeregowych członków zespołu. U najbardziej introwertycznych programistów konieczność tak ścisłej współpracy może wywołać poczucie dyskomfortu. Niektórzy programiści czują się też jedynymi właścicielami swojego kodu i nie lubią, gdy ktoś ten kod modyfikuje lub choćby ogląda. To, co się zwykle powinno osiągnąć to przekonać wątpiących tak, aby zrozumieli istotę potencjalnych problemów, jakie widzą oni w tej metodologii, pokazywać im, jak mogą być one rozwiązane i przedstawić im w Scrum te sprawy, które mogą być dla nich osobiście użyteczne [8]. W przypadku programistów, uświadomienie im perspektywy nauczenia się nowych technologii i technik zwykle jest metodą na zwalczenie tego zagrożenia. W przypadku niektórych ludzi może się zdarzyć, że nie przekonają się oni nawet do wypróbowania tej metodologii i wtedy najlepiej będzie ich usunąć z zespołu – jak dotąd, nie zetknąłem się z taką sytuacją.

Wreszcie, mogą się pojawić problemy spowodowane przez pomyłki, wynikające z braku doświadczenia we wprowadzaniu nowej metodologii w organizacji. Kilka najbardziej niebezpiecznych, na jakie się natknąłem w praktyce, to: Odgórne zarządzenie wprowadzenia metodyki; Przywiązywanie zbyt dużej wagi do procesu i zbyt małej do produktu.

Zostały one przedyskutowane w [3]. Tu zamieszczę jedynie krótkie podsumowanie wyjaśniające, dlaczego mogą one przysparzać problemów. Pierwszy z nich może się zdarzyć, jeśli kierownik projektu decyduje, że lubi Scrum i wymusza go na programistach (niekiedy na właścicielu produktu również). Problem w takim podejściu polega na tym, że wymuszanie bardzo rzadko służy programistom. Zatem próba wymuszania na nich nowej metodologii może w efekcie przynieść efekt odwrotny do zamierzonego.

Drugie z kolei jest typowe dla zespołu, który używa tego szczególnego procesu po raz pierwszy. W pewnej mierze jest to normalne: w końcu jeśli się ktoś uczy czegoś nowego, to trzeba się na tym skupić, żeby się dowiedzieć, czy robi

Odnośniki

- [1] Asproni, G., *An Experience Report on Implementing a Custom Agile Methodology on a C++/Python Project,* Overload 64, 2004, *http://www.giovanniasproni.com/articles*
- [2] Asproni, G., Motivation, *Teamwork, and Agile Development,* Agile Times Vol. 4, 2004, *http://www.giovanniasproni.com/articles*
- [3] Asproni, G., *How to Shoot Yourself in the foot. In an Agile Way,* Overload 71, 2006, *http://www.giovanniasproni.com/articles*
- [4] Beck, K., Andres, C., *Extreme Programming Explained: Embrace Change*, 2nd ed., Addison Wesley, 2004
- [5] Beck, K., et al., *The Agile Manifesto, http://www.agilemanifesto.org*
- [6] Boehm, B. W., *Software Engineering Economics*, Prentice Hall, 1981
- [7] Cockburn, A., *Agile Software Developmen*t, A. Wesley, 2002
- [8] Manns, M. L., Rising, L., *Fearless Change: patterns for introducing new ideas*, Addison Wesley, 2004
- [9] N.A., Scrum Development Group, http://groups.yahoo.com/group/scrumdevelopmen [10] N.A., XP @ Scrum, *http://www.controlchaos.com/about/xp.php*
- [10] N.A., *ScrumAlliance, http://www.scrumalliance.org/*
- [11] N.A., *Agile Project Management Group, http://finance.groups.yahoo.com/group/agileprojectmanagement/*
- [12] N.A., *AgileAlliance, http://www.agilealliance.org*
- [13] Schwaber, K., *Agile Project Management with Scrum*, Microsoft Press, 2004
- [14] Schwaber, K., Beedle, M., *Agile Software Development with Scrum*, Prentice Hall, 2002

się dobrze, czy źle. Jednak gdy zespół zaczyna spędzać więcej czasu na procesie, niż na produkcie, powinien to być znak, że chyba nie wszystko jest w porządku. Podsumowując, praca w dobrze zorganizowanym projekcie Scrumowym może być bardzo korzystnym doświadczeniem dla wszystkich uczestników: właściciel produktu dostaje lepszy produkt wcześniej, programiści mają większą satysfakcję ze swojej pracy i większe możliwości uczenia się nowych rzeczy dzięki przeplatającym się funkcjom zespołu i uczestnictwie w każdym aspekcie projektu, a kierownik projektu ma większą kontrolę nad projektem i lepszy wgląd w to, co się w nim dzieje.

Scrum i programowanie ekstremalne

Po co ten akapit? Przecież metody *agile* są różne, więc po co porównywać Scrum z programowaniem ekstremalnym (j.ang. *extreme programming*, XP) [4]? Powód prosty: XP jest prawdopodobnie najbardziej znaną metodą *agile* i pierwszą rzeczą, jaką ludzie chcą się dowiedzieć, gdy przedstawia się inną z nich, jest jak się one do siebie mają. Niestety mają się one do siebie mniej więcej jak jabłka do pomarańczy. Nawet założywszy, że obie spełniają *Manifest Agile*, dotyczą różnych spraw. Scrum dotyczy strony zarządzającej, pozostawiając inżynierskie sprawy – projektowanie, kodowanie, zarządzanie konfiguracją, testowanie itd. – samoorganizującemu się zespołowi, podczas gdy XP dotyczy właśnie inżynierskich spraw i nie dostarcza żadnych specyficznych narzędzi do zarządzania projektem. W praktyce często stosuje się je razem, a także istnieją metodologie oparte na połączonych tych dwóch metodach. Przykłady można obejrzeć w [10].

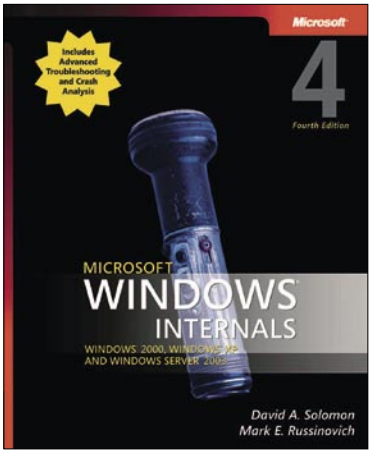
Podsumowanie

W tym artykule przedstawiłem podstawy Scrum, korzyści z jego zastosowania i niektóre z problemów – wraz z możliwymi rozwiązaniami – które mogą się pojawić podczas wdrażania. Jeśli ktoś chce nauczyć się więcej, w sekcji odnośników znajdzie mnóstwo zasobów do obejrzenia, wiele z nich jest dostępnych za darmo w Internecie. Dobrym punktem startu są strony *ScrumAlliance* [11] oraz *AgileAlliance* [13], na których można znaleźć kilka darmowych artykułów o „*Agile Development*" w ogólności i Scrum w szczególności. *ScrumAlliance* zawiera też parę odniesień do darmowych narzędzi, które mogą być użyteczne, gdyby się ktoś decydował na zastosowanie Scrum w swojej organizacji.

Inne zasoby na sieci zawierają też dwie interesujące grupy Yahoo otwarte dla wszystkich: *Scrum Development* [9] i *Agile Project Management* [12], z których można skorzystać poprzez zadawanie pytań ekspertom programowania metodami *agile* oraz uczestnictwo w różnych, często gorących dyskusjach.

Zanim ktoś spróbuje zastosować Scrum swoim projekcie, oprócz podparcia się darmowymi zasobami opisanymi wyżej, sugeruję przeczytać – co najmniej – książkę Alistaira Cockburna o *agile software development* w ogólności [7] i dwie książki o Scrumie Kena Schwabera [14].

Jeśli zdecydujesz, że naprawdę lubisz Scrum – i masz trochę zbędnych pieniędzy – możesz zostać Certyfikowanym Szefem Scrum po odbyciu dwudniowego szkolenia. Możesz znaleźć więcej szczegółów na ten temat na stronie ScrumAlliance [11]. ■



Microsoft Windows Internals, 4th Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000

Autorzy: *Mark E. Russinovich, David A. Solomon*
Wydawnictwo: *Microsoft Press 12/2004*
ISBN: *0-7356-1917-4*
Ocena: *5-*

Podczas gdy od kilku już lat trwa nieustająca kampania promocyjna środowiska programistycznego firmy Microsoft – platformy .NET, a serwisy internetowe huczą od plotek na temat wciąż odwlekanej premiery najnowszego dziecka korporacji – systemu Windows Vista, w tle tysiące programistów wciąż wspiera, rozwija, a i także tworzy oprogramowanie systemowe. Specyfika tego typu aplikacji wymaga wiedzy niemal tajemnej o wnętrznościach i mechanizmach funkcjonujących u podstaw systemów Windows XP, Windows Server 2003 i Windows 2000. Wiedzę tę niewątpliwie posiadają autorzy: M. Russinovich i D. Solomon. Niestety nie mogę napisać, iż mimo niemal tysiąca stron druku, dzięki tej książce dowiemy się wszystkiego - każdego szczegółu na temat tego, co wiedzieć powinniśmy o rodzinie systemów Windows NT. Nie istnieje jednak na rynku lepsza pozycja, równie kompleksowo omawiająca aspekty związane z wnętrznościami tego systemu. Otrzymujemy, bowiem bogaty zestaw informacji referencyjnych dotyczących: jądra i architektury systemu, zarządzania procesami, wątkami, pamięcią, konfiguracją systemu, a także opis mechanizmów działania systemu plików, wejścia – wyjścia, operacji sieciowych, bezpieczeństwa, czy nawet elementów takich jak uruchamianie, zamykanie oraz analiza błędów napotkanych w czasie działania systemu. Dostrzeżone minusy to brak wersji elektronicznej lub nawet CD z zamieszczonymi aplikacjami wykorzystanymi w książce. Zbyt krótki, ale za to bardzo ciekawy, rozdział o analizowaniu błędów występujących podczas awarii systemu oraz dość znaczna liczba dostrzeżonych błędów. Sądzę, iż każdy programista poważnie traktujący tworzenie aplikacji na platformie Windows, powinien zastanowić się nad zakupem tej niezbyt taniej pozycji.

Zrecenzował: Stefan Turalski
http://www.promise.pl