FluidCircle
Płynna doskonałość

PRAGMATISTS

SCRUM W POLSCE: PRAGMATISTS

Scrum w Polsce

Celem programu "Scrum w Polsce" jest przedstawienie **realnych doświadczeń** zespołów stosujących Scrum i pokrewne metody z rodziny *Agile*, aby kolejne firmy mogły ocenić jak Scrum może pomóc w ich sytuacji oraz by mogły płynniej rozpocząć stosowanie Scrum we własnej pracy.

O Pragmatists

Pragmatists to założona w 2009 roku przez Pawła Lipińskiego firma zajmująca się rozwojem oprogramowania. Paweł przed założeniem Pragmatists od wielu lat pracował w środowisku *Agile* jako inżynier, a następnie jako wiodący architekt i *agile coach* w Nokia-Siemens Networks. Dlatego kultura Pragmatists od początku silnie zakorzeniona jest w wartościach i metodach *Agile*. W kulturze tej wysoko ceniona jest ciągła nauka obejmująca zarówno stabilny wzrost efektywności jak i bardziej radykalne eksperymenty. Drugą wartością jest (samo)dyscyplina, dzięki której dobre praktyki mogą być stopniowo wzmacniane a złe nawyki systematycznie eliminowane. Pragmatists pielęgnuje takie właśnie podejście do pracy między innymi poprzez dobór pracowników, którzy sami z siebie cenią te same wartości.

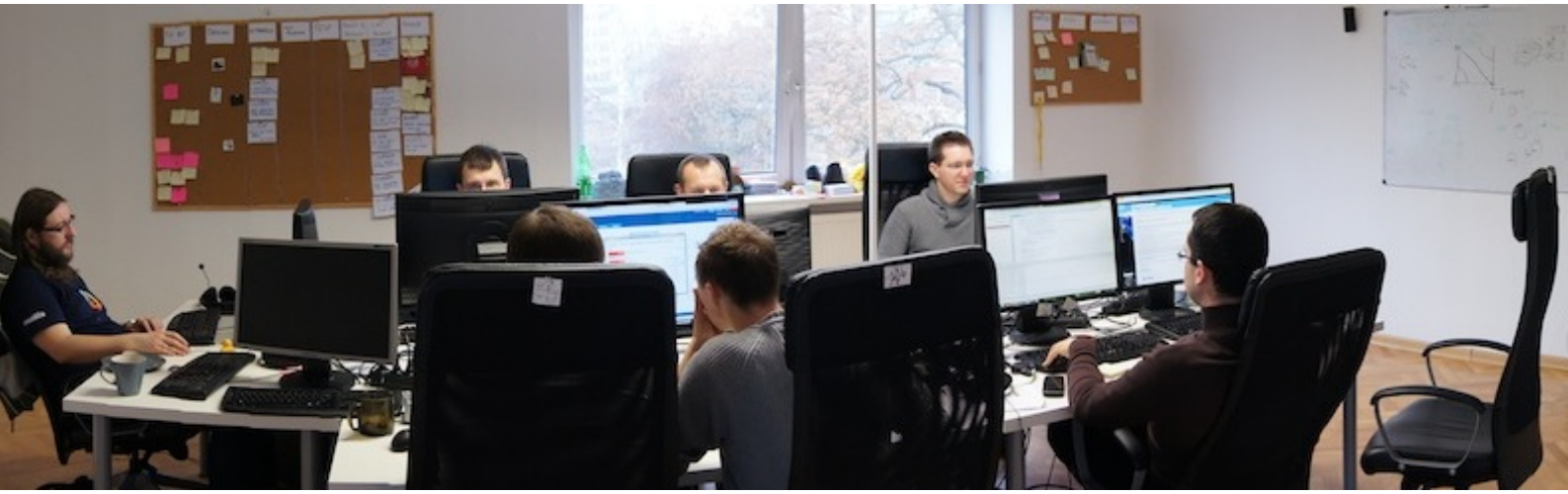
Od strony praktycznej praca Pragmatists skoncentrowana

Co to jest Scrum?

Scrum to **metoda organizacji pracy zespołów** tworzących nowe rozwiązania (informatyczne i nie tylko).

Jako **adaptacyjna** metoda zarządzania Scrum działa szczególnie dobrze tam, gdzie trudno jest z góry przewidzieć wszystkie szczegóły projektu: od prawdziwych potrzeb klienta po problemy techniczne.

Więcej informacji o Scrum w pozostałych przypadkach z serii Scrum w Polsce oraz w dziale Artykuły na stronie FluidCircle.net.



jest wokół jednego bardzo dużego projektu przeplatającego się z mniejszymi projektami dla różnych klientów. Oprócz tworzenia oprogramowania Pragmatists pomaga też innym firmom w Polsce i za granicą rozwijać się w duchu *Agile*.

Organizacja pracy

Jako swoją główną metodykę Pragmatists stosuje eXtreme Programming wzbogacony o najważniejsze elementy Scrum oraz o własne parktyki, pomagające maksymalizować wydajność zespołu i zwinność organizacyjną.

★**Interdyscyplinarny, samoorganizujący się zespół** – zespół jest częściowo rozproszony i składa się z pracowników Pragmatists zlokalizowanych w Polsce oraz pracowników klienta urzędujących w jego siedzibie w Luksemburgu.

> **Dobra praktyka**: Samoorganizacja i powiązana z nią odpowiedzialność zespołu wspierana jest przez postawę Pawła, który zachęca lub wręcz wymusza na zespole podejmowanie wszystkich decyzji, które dotyczą środowiska i metod pracy. Zamiast szefa podejmującego arbitralne decyzje zespół wspierany jest przez *agile coach*'a, który pomaga zespołowi poprawiać własną efektywność bez narzucania własnych poglądów i decyzji.

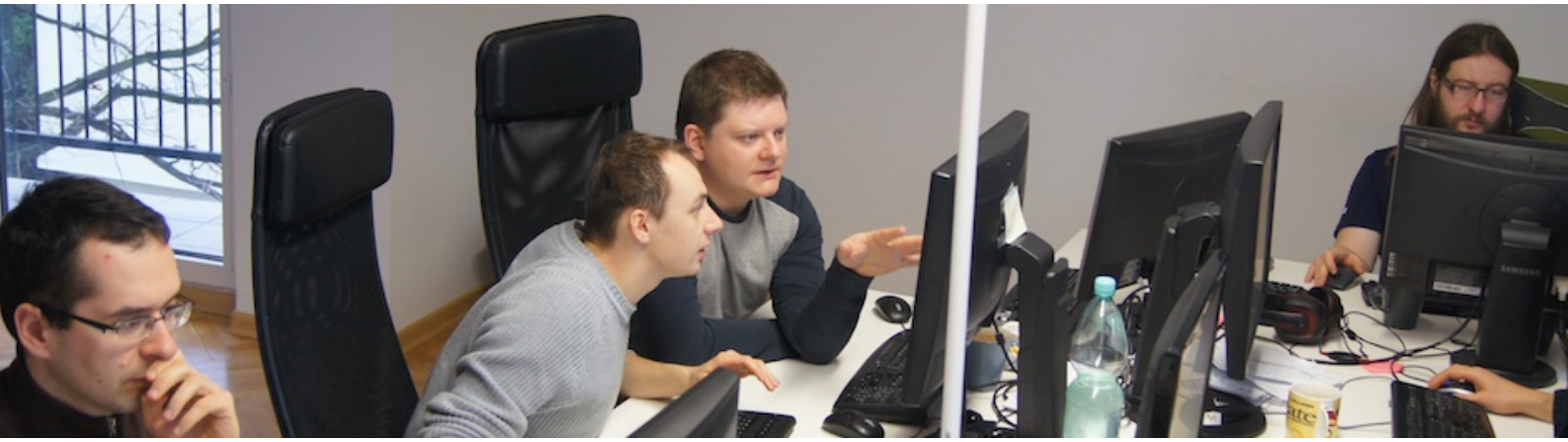
★**Praca w rytmie sprintów** umożliwiających częstą inspekcję i adaptację. W największym, rozproszonym geograficznie projekcie realizowanym przez Pragmatists trwają one dwa tygodnie, a w projektach realizowanych lokalnie – jeden tydzień.

★**Planowanie** i **przegląd sprintu** – mają miejsce odpowiednio na początku i na końcu każdego sprintu. Odbywają się z udziałem zespołu oraz Właściciela Produktu i Scrum Mastera i pozwalają na regularny przegląd rzeczywistych postępów prac i dopasowanie dalszych planów na podstawie najświeższych informacji.

★**Product Owner** – rolę tę pełni wyznaczona osoba po stronie klienta.

★**Scrum Master** – rolę tę "pełni" cały zespół, który wspólnie dba o prawidłowy przebieg procesu i usuwanie przeszkód spowalniających pracę.

★**Codzienne standupy całego zespołu** – prowadzone z wykorzystaniem specjalistycznego narzędzia wspierającego Scrum oraz oprogramowania do telekonferencji.



★**Retrospekcje** – wspólne dla całego zespołu i wewnętrzne dla jego polskiej częsci.

★**Backlog produktu** i **backlog sprintu** – stale aktualizowane (odpowiednio przez Właściciela Produktu z uwzględnieniem priorytetów biznesowych klienta i przez zespół z uwzględnieniem aktualnego stanu prac).

Pragmatists stosuje też szereg praktyk, które nie są obowiązkową częścią Scrum, lecz są często stosowane przez zwinne zespoły:

★**Fizyczna tablica Scrum** – od czasu do czasu ożywa, gdy potrzebna jest bardziej widoczna wizualizacja stanu prac, jednak nie jest stosowana stale ze względu na rozproszony charakter zespołu.

★**Historyjki użytkownika** (user stories) jako wygodna forma dla wymagań biznesowych klienta.

★**Story points** – szacowanie złożoności wymagań w jednostkach względnych, ułatwiających bardziej trafne szacowanie i umożliwiających bardziej precyzyjny pomiar efektywności zespołu. W Pragmatists relatywnie duże różnice między historyjkami utrudniają stosowanie tych punktów w celu precyzyjnego mierzenia efektywności prac, lecz przydają się jako system wczesnego ostrzegania przed dużymi rozbieżnościami między rzeczywistą pracochłonnością zadań a oczekiwaniami klienta.

Powyższa lista nie jest zamknięta, jako że proces wytwórczy Pragmatists wciąż się rozwija: np. od marca 2012, po trzech latach współpracy, zespół wraz z klientem zaczął stosować kryteria akceptacyjne w stylu **BDD** (behavior-driven development). Natychmiast uwidoczniły się braki w komunikacji, których usunięcie pozwoliło jeszcze bardziej zwiększyć wartość biznesową (trafność) nowych funkcjonalności w projekcie, który jeszcze przed tą zmianą był już uważany za duży sukces.

W chwili tworzenia niniejszego przypadku rozpoczynały się prace nad **automatyzacją testów** akceptacyjnych z myślą o dalszym zwiększeniu precyzji wyrazu i zmniejszeniu pracochłonności ich obsługi.

W ciągu trzech lat pracy wzrost produktywności Pragmatists pochodził głównie z usuwania barier w komunikacji z klientem oraz z rozwoju technologii Java. Kolejnym ważnym czynnikiem jest rozwój samego zespołu dzięki stosowaniu zwinnych praktyk i kulturze pracy zorientowanej na uczenie się i pracę zespołową.

Programowanie w Parach

Wiele wysoko wyspecjalizowanych dziedzin od lat korzysta z wzrostu efektywności i bezpieczeństwa jakie daje praca w parach: Zarówno projektanci jak i piloci w branży lotniczej i kosmicznej pracują w ten sposób. Także w medycynie bardziej skomplikowane procedury wykonywane są jednocześnie przez dwóch lekarzy. Według Pragmatists praca w parach powinna być też standardowym modelem tworzenia oprogramowania. Dlatego w Pragmatists zdecydowana większość zadań wykonywanych jest w modelu Pair Programming. Wyjątkiem są tylko sytuacje, gdy liczba aktywnych programistów jest

nieparzysta albo gdy zadaniem jest konfiguracja lub prototypowanie eksploracyjne. Jednak nawet wtedy wynik pracy obowiązkowo podlega przeglądowi przez innego członka zespołu.

Komunikacja w zespole jest dodatkowo wspierana przez fakt, że jego fizyczne rozlokowanie pozwala każdemu członkowi widzieć wszystkich innych przez cały czas. Po prostu wszyscy siedzą wokół jednego wielkiego stołu, skierowani twarzami do siebie, co zachęca do aktywnej komunikacji.

Główne korzyści ze stosowania pair programming to:

★Dużo **wyższa jakość kodu** i co za tym idzie **minimalizacja długu technicznego** dzięki podwójnej uważności i zwiększonej liczbie kreatywnych pomysłów wygenerowanych przez dynamicznie współpracującą parę.

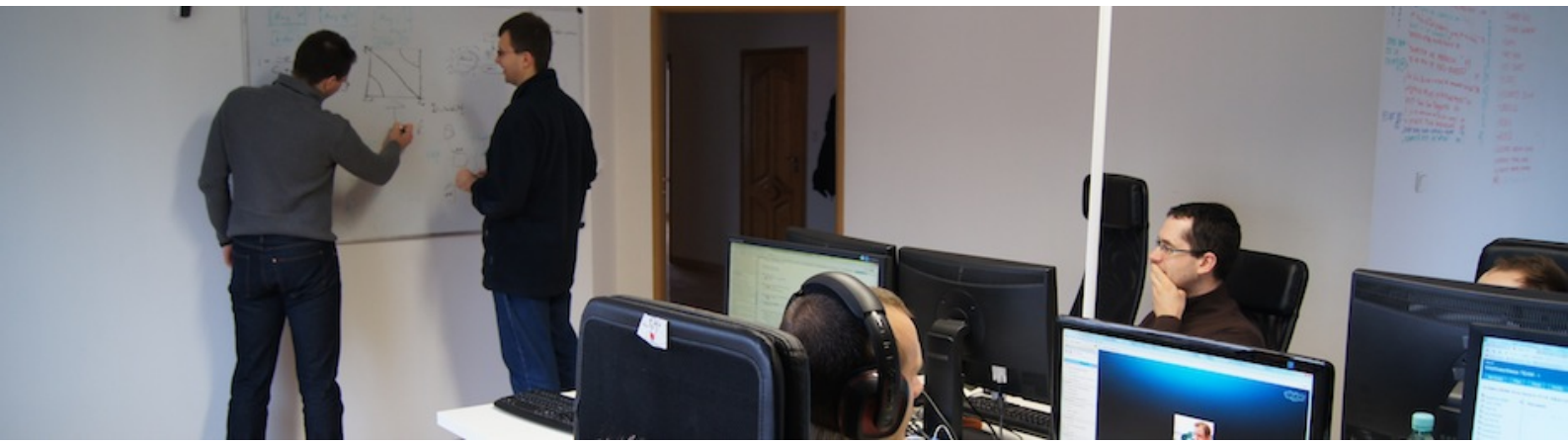
★**Propagacja wiedzy** – każdy członek zespołu orientuje się we wszystkich obszarach systemu. Nie zdarzają się sytuacje, w których krytyczny błąd musi czekać dwa tygodnie na naprawienie, bo jedyna osoba, która ma wiedzę o danym module właśnie jest na wakacjach (problem, który rzeczywiście wystąpił po stronie klienta Pragmatists).

★**Wzrost kompetencji** – wszyscy członkowie zespołu stale uczą się od siebie nawzajem.

Jednym z głównych wątpliwości managerów rozważających zastosowanie programowania w parach w swoim zespole jest obawa o wzrost pracochłonności wszystkich zadań – w doświadczeniu Pragmatists okazuje się to złudna obawa, gdyż przypisanie drugiej osoby do zadania jest z nawiązką skompensowane przez wzajemną mobilizację pracujących. Mówiąc wprost: rozproszenie uwagi i marnowanie czasu, w które łatwo wpaść w pojedynkę jest podczas pracy w parze zredukowane praktycznie do zera.

Jak skutecznie wprowadzić programowanie w parach?

Według Pragmatists główne wyzwania programowania w parach dotyczą miękkich aspektów współpracy. Pair programming wymusza bliski kontakt między członkami zespołu – w takich warunkach trudno zachować pozory chłodnego profesjonalizmu, który pozwoliłby ukryć nieuniknione tarcia. Współpracownicy muszą wykazać pewną dozę dobrej woli oraz trochę umiejętności interpersonalnych, aby zbudować atmosferę porozumienia i zaufania niezbędną do efektywnej współpracy. Każdy członek zespołu musi umieć w sposób nieagresywny lecz asertywny przedstawić swoje stanowisko oraz otwarcie przyjąć postulaty innych.



Dla wielu programistów jest to duże wyzwanie – sam fakt pokazania innym swojego trybu pracy jest dla wielu poważną przeszkodą, lecz z odpowiednią dbałością i praktykami w rodzaju grupowego Coding Dojo można poradzić sobie z tym problemem i otworzyć daną osobę na bliższą współpracę.

Standardowym trybem pracy podczas programowania w parach jest TDD ping-pong, w którym jeden z programistów tworzy test dla danej funkcjonalności, drugi pisze kod, który zaspokaja testa a pierwszy z kolei refaktoruje powstały kod. Takie przeplatanie pracy ułatwia właściwe rozłożenie obciążenia, stumuluje uwagę. Dodatkową korzyścią jest minimalizacja zagrożenia dominacją jednego z członków pary: w przeciwnym wypadku mogłoby się zdarzyć, że sprawniejszy programista wykonuje całą pracę a mniej sprawny jedynie obserwuje (albo w ogóle traci uwagę i zaczyna myśleć o czymś innym).

Kolejnym aspektem, o który warto zadbać jest do pewnego stopnia wymuszona rotacja par, bez której sprawniejsi programiści będą się dobierać ze sprawniejszymi a słabsi ze słabszymi. Dzięki temu różnica wiedzy o systemie oraz wiedzy i umiejętności technicznych zostaje szybko zredukowana. W praktyce Pragmatists sprawdza się stara zasada eXtreme Programming mówiąca, że jeśli z czymś mamy kłopot to robimy to częściej, najlepiej z kimś lepszym od siebie i konsekwentnie wychodzimy poza swoją strefę komfortu w imię wyższej skuteczności pracy.

Test Driven Development i minimalizacja długu technicznego

W doświadczeniu Pragmatists wiele zespołów stosujących Scrum nie osiąga pełnych korzyści przez brak wystarczających kompetencji technicznych poszczególnych inżynierów oraz brak odpowiednich praktyk na poziomie zespołu. Dlatego praktyki te są zawsze wysoko na liście priorytetów Pragmatists.

Jedną z tych praktyk jest Test Driven Development. TDD jest stałym elementem pracy w Pragmatists i w konsekwencji znaczna większość linii kodu, które zespół wytworzył podczas ponad trzyletniej pracy nad systemem pokryta jest zautomatyzowanymi testami.

Według Pragmatists koszt wytworzenia tych testów zwrócił się już wielokrotnie dzięki wysokiej jakości kodu i niskiemu poziomowi długu technicznego.

Oprócz tego w projektach Pragmatists konsekwentnie stosowana jest polityka **zero błędów**, czyli zasada, która każe każdy wykryty błąd naprawiać zawsze w pierwszej kolejności. Stosowanie tej zasady wymaga ciągłej uwagi i wysokiej dyscypliny, ale dzięki starannie dobranej architekturze i dobrej jakości kodu naprawa błędów najczęściej nie jest zbyt pracochłonna, a w zamian utrzymywany jest wyższy poziom zadowolenia klientów jak i komfort pracy inżynierów.

Dług techniczny odnosi się do obniżonej jakości kodu, w zamian za którą "kupione" zostało przyspieszenie zakończenia danego fragmentu systemu lub obniżenie kosztu jego wytworzenia.

Jeśli raz powstały dług techniczny nie jest spłacany, stopniowo doprowadza on do spowolnienia lub wręcz uniemożliwienia dalszego rozwoju oraz rozmnożeniu trudnych do usunięcia błędów.

Korzyści z TDD i polityki zero błędów, jakich doświadczył Pragmatists to m.in.:

★bardziej precyzyjne wyjaśnienie wymagań i możliwość wcześniejszego wykrycia ew. nieporozumień między Właścicielem Produktu, developerami a testerami - dzięki temu praktycznie nie zdarza się odrzucenie historyjki po jej dostarczeniu podczas przeglądu sprintu,

★elastyczna architektura ułatwiająca adaptację systemu do nowych potrzeb i pomysłów.

★swoboda wprowadzania znaczących przeróbek architektonicznych - takie przeróbki możliwe są wyłącznie dzięki baterii testów chroniących przed zepsuciem istniejącej funkcjonalności systemu.

★wyższa jakość kodu - mniej okazji do powstania błędów teraz i w przyszłości, a co za tym idzie możliwość realizacji polityki "zero błędów".

Podsumowanie

Proces wytwórczy Pragmatists jest wzorcowym przykładem współczesnego podejścia do rozwoju oprogramowania. Pragmatists korzysta z **pełnej implementacji Scrum** pogłębionej o techniki wzmacniające pracę zespołową oparte o **programownie w parach** oraz praktyki techniczne, takie jak **Test Driven Development**, pozwalające zachować wysoką jakość produktów.

Wywiadu będącego podstawą przypadku udzielili założyciel Pragmatists Paweł Lipiński.

Więcej przypadków oraz sumaryczne informacje o programie można znaleźć pod adresem: **http://fluidcircle.net/case**.

