



Zwinne Rozwijanie Oprogramowania

zwinne praktyki, procesy i wartości

PAWEŁ LIPIŃSKI

Pragmatists Sp. z o.o.

email: contact@pragmatists.pl

tel: +48 691-512-774



Wstęp

Artykuł ten ma za zadanie wprowadzić Czytelnika w tematykę zwinnego rozwijania oprogramowania. Skierowany jest głównie do osób nie mających praktycznego doświadczenia z zastosowaniem zwinnych metodyk, choć myślę, że może być wartościowy również dla tych, którzy mieli z nimi kontakt lub nawet aktualnie pracują w takim trybie. Staralem się skupić w nim głównie na ideach związanych ze zwinnością w projektach oraz na *zwinnym sposobie myślenia*, który niezbędny jest do efektywnego wdrożenia zwinnych metodyk, a mniej na praktykach i procesach, którym poświęconych jest wiele dostępnych książek i artykułów.

Programowanie to nie to samo co tworzenie projektów programistycznych. Do programowania wystarczy znajomość języka, bibliotek, technologii, algorytmów. Tego można nauczyć się na studiach, z książek, czytając API i metodą prób i błędów. Choć można nauczyć się tak naprawdę dużo, trudno zrealizować udany projekt bazując tylko na czysto technicznej wiedzy i dobrych chęciach. Tym bardziej, że często nie pracuje się z doświadczonymi programistami. Siedzi się więc w pracy w weekendy i po godzinach, by dostarczyć oprogramowanie na czas. Ciągłe zmagania się z problemami z niezrozumieniem wymagań, co wychodzi dopiero w testach. Z niską jakością kodu, który jest przepisywany na wcale nie koniecznie lepszy, za każdym razem, gdy ktoś nowy dołącza do zespołu. Tak naprawdę mało kto zna odpowiedź na najbardziej podstawowe w naszym zawodzie pytanie: **jak tworzyć wysokiej jakości oprogramowanie w powtarzalny sposób?**

Wystarczy posłuchać znajomych, którzy w swoich miejscach pracy są użytkownikami tworzonego tak oprogramowania: projekty nie dostarczane na czas, z ciągłymi problemami jakościowymi, z ciągnącymi się miesiącami wdrożeniami, z sytuacjami gdy poprawka jednego błędu powoduje wprowadzenie paru nowych, z ciągłymi napięciami między zawiedzionym odbiorcą i zestresowanym dostawcą. Taka jest często rzeczywistość naszej profesji. Nasuwa się więc pytanie – czy nie ma lepszego sposobu rozwijania oprogramowania? Czy NIE DA SIĘ realizować projektów tak, by klienci byli zadowoleni z rezultatów? Czy zawsze muszą być problemy jakością aplikacji?



Oczywiście aplikacje DA się tworzyć tak, by użytkownik był z nich zadowolony i da się również pisać je w takiej jakości, żeby nie trzeba było borykać się z dziesiątkami błędów po wdrożeniu. Ja dochodziłem do tego stopniowo, ucząc się początkowo wyłącznie na własnych błędach i starając się realizować kolejne projekty coraz lepiej. W którymś momencie spotkałem się ze zwinnymi metodykami i od tego czasu starałem się implementować jak najwięcej z proponowanych tam praktyk. Czym dokładniej zwracałem uwagę na ich stosowanie i głębiej starałem się zrozumieć stojące za nimi idee, tym wyraźniej lepsze były efekty.

Przy tworzeniu firmy Pragmatists starałem się znaleźć osoby podzielające moje poglądy na tworzenie oprogramowania i prowadzenie projektów. Opierając się o zawarte w tym artykule idee i wartości staraliśmy się wspólnie wdrażać związane z nimi praktyki. Artykuł ten zawiera więc moje komentarze do *Agile Manifesto* bazujące na paru latach (drogi do) zwinnej pracy przed firmą Pragmatists oraz ostatniego pół roku bardzo intensywnych pod tym względem doświadczeń w ramach tego zespołu.



Manifest Zwinnego Rozwijania Oprogramowania

Na czym polega zwinne rozwijanie oprogramowania? Skupmy się na Manifeście Zwinnego Rozwijania Oprogramowania, który został opracowany w 2001 roku przez ludzi którzy zadawali sobie pytania podobne do tych, które zadałem we wstępie i wypracowali działające i powtarzalne metody prowadzenia udanych projektów. Jest on sformułowany w postaci czterech zdań wartościujących i dwunastu precyzujących je zasad. Warto zwrócić uwagę na budowę tych zdań. Każde ma prawą i lewą stronę – po prawej są elementy znane z większości projektów: narzędzia, procesy, umowy, dokumentacje, plany. Lewa strona zawiera wartości ważniejsze z punktu widzenia zwinnych projektów. Manifest zwraca uwagę, że elementy po prawej nie są bez znaczenia – mogą w znaczący sposób wpłynąć na przebieg i rezultat projektu – lecz te po lewej są znacznie ważniejsze i mają dla projektów krytyczne znaczenie. Przejdźmy zatem przez te zdania, kolejno komentując w ich znaczenie.

Osoby i interakcje NAD procesy i narzędzia

W zwinnych metodykach bierze się pod uwagę nie tylko praktyki, procesy, języki, czy architekturę, ale główny nacisk kładzie się na zespołowość tworzenia oprogramowania. Mówimy, że programowanie to sport zespołowy – żadna pojedyncza osoba nie jest w stanie udźwignąć samodzielnie większego projektu. I nie chodzi tu o samodzielne jego tworzenie, lecz o samodzielne ogarnięcie go w całości i efektywne zarządzanie nim. Co więcej zespół ma dużo większe szanse niż pojedyncza osoba by znaleźć lepsze rozwiązania, by poradzić sobie z presją, by nie przenosić swoich emocji na projekt. Zwinne metodyki zwracają uwagę na odpowiedzialność ludzi. Mówimy o samoorganizujących się zespołach, o tym, że zaufanie i przenoszenie władzy / kontroli wraz z odpowiedzialnością z kierownictwa na sam zespół jest sposobem na zwiększenie zaangażowania, zadowolenia z pracy, a w skutkach również jakości projektów. Ważni są sami członkowie zespołu, ponieważ ich jakość w sensie zawodowym (wiedza, doświadczenie, ale również łatwość współpracy z nimi i inne



tzw. „miękkie” cechy) przekłada się na jakość samego oprogramowania, które tworzą.

Z drugiej strony znane powiedzenie „*a fool with a tool is still a fool*” zdaje się dobrze wyrażać dlaczego narzędzia nie są najważniejsze. Nie chodzi o to, że są one zupełnie nieważne – w szczególności są pewne narzędzia, które prawie wykluczają możliwość efektywnej pracy. Jednak problemy techniczne są zwykle dużo łatwiejsze do rozwiązania niż problemy osobowe w zespole, czy problemy komunikacyjne między członkami zespołu. Również same procesy rozwoju oprogramowania nie są kluczowe, choć mogą znacząco ułatwiać lub utrudniać pracę / osiągnięcie sukcesu w projekcie. Proces jest elementem modyfikowanym i dostosowywanym przez zespół w zależności od sytuacji. Jeśli zespół nie dostosowuje swojego procesu do zmian zachodzących dokoła niego (projekt, ludzie), nie jest tak naprawdę zwinny.

Działające oprogramowanie NAD szczegółową dokumentacją

Co wyznacza sukces projektu? Co powoduje, że klient będzie z niego zadowolony? Niektóre podejścia zdają się promować ideę, że jest to dokumentacja. Zalecają dokumentowanie każdego elementu systemu, każdej decyzji, często wraz z rozumowaniem które doprowadziło do jej podjęcia. W wielu projektach powstają setki stron dokumentów, które rzadko przynoszą klientowi prawdziwą wartość. W zwinnych metodykach promuje się podejście, by dokumenty tworzone były wyłącznie w minimalnym niezbędnym zakresie. Staramy się pisać je wyłącznie wtedy, gdy jest ktoś, kto ich potrzebuje. Spisujemy tylko tyle, ile jest faktycznie potrzebne. Tworzymy więc dokumentację tylko, jeśli przynosi ona klientowi konkretną wartość. Może to być dokumentacja dla użytkownika, może być również dokumentacja architektoniczna, tak by wytłumaczyć podstawowe założenia i decyzje co do systemu. Dużo rzadziej wymagana jest dokumentacja programistyczna. Programiści wolą czytać kod niż dokumenty. Nigdy nie wiadomo czy dokumenty były na bieżąco aktualizowane – kod jest zawsze aktualny. Dokumenty nie koniecznie odzwierciedlają kompletny stan systemu – kod nie posiada tej wady. Oczywiście kod musi być czytelny, musi w klarowny sposób mówić czytelnikowi co robi. Mówimy więc, że od szczegółowej dokumentacji wolimy działające oprogramowanie. To ono przynosi wartość jego użytkownikom i jest zawsze najaktualniejszą dokumentacją projektu.



Czasem mówi się, że kod nie jest dobrą dokumentacją, ponieważ jest nieczytelny, każdy programista biorący udział w jego tworzeniu miał inny styl, lub po prostu jest pełen błędów. Ja twierdzę, że taki kod to najlepsza dokumentacja techniczna projektu – dokładnie pokazuje jaki jest stan projektu i powinien być sygnalizatorem tego, że projekt jest na drodze do klęski i niedługo ten stan osiągnie, jeśli natychmiast nie rozpocznie się jego ratowania. Czytelny kod nie tylko dowodzi stabilności projektu i kontroli projektu przez zespół, ale również odzwierciedla stan samego zespołu i jego wiarygodność.

Współpraca z klientem NAD negocjowanie kontraktu

Większość projektów tworzonych jest w trybie *fixed price, fixed time, fixed scope*. Takie przynajmniej są początkowe założenia. Dostaje się od klienta zestaw wymagań lub zbiera się je od niego. Następnie oszacowuje się ich pracochłonność i na jej podstawie przedstawia koszt. By zmieścić się w zadanym czasie, opierając się na doświadczeniu, określa się (zgaduje?) właściwy rozmiar zespołu. Jednak podczas rozwoju systemu ZAWSZE okazuje się, że zakres tak naprawdę nie był zamknięty. Zrozumienie wymagań przez zespół może być niepełne, czasem zupełnie złe, albo klient potrzebuje dodać/zmienić jakieś wymaganie. Przy projektach, które trwają parę miesięcy jest to zupełnie naturalne. Co więcej uniemożliwienie klientowi zmiany wymagań co do aplikacji przez parę miesięcy jest po prostu nieuczciwe. Nikt nie jest w stanie wyłącznie na podstawie wstępnych wyobrażeń szczegółowo określić wszystkich wymagań. Dlatego dużo efektywniejsze i uczciwsze jest wypracowywanie szczegółów w trakcie powstawania projektu. Oczywiście niezbędne jest wstępne zarysowanie wymagań, ponieważ zespół musi w końcu wiedzieć co tworzy. Zależy od tego również dobór technologii, w których tworzony będzie system. Jest to jednak tylko wysokopoziomowe określenie celu i kierunku. Wszelkie szczegóły ujawniają się w trakcie rozwijania oprogramowania. Wymaga to ścisłej współpracy z klientem, ponieważ to właśnie jemu najbardziej zależy na spełnieniu przez nasz produkt jego oczekiwań.

Z naszych doświadczeń wynika, że klientom odpowiada częste oglądanie aplikacji w trakcie jej rozwoju. Mogą dzięki temu zgłaszać uwagi do tworzonych części systemu i wpływać na jego docelowy kształt. Bliska współpraca z klientem nad aplikacją buduje wzajemne zaufanie.



Prowadzone zwinnie projekty realizowane są zwykle z użyciem umów typu *time & materials*. Wymaga to jednak kontroli kosztów przez klienta, by nie przekroczyć zakładanego dla projektu budżetu. Unika się w ten sposób przerzucania ryzyka związanego z szacowaniem między odbiorcą i dostawcą aplikacji. W przypadku umów *fixed price*, *fixed scope* odbiorca oczekuje konkretnej ceny i terminu dla danego zakresu, podczas gdy dostawca chce zbilansować ryzyko przez zawyżenie czasochołonności lub ogromne stawki za realizację zmian. Umowy *time & materials* są w tym względzie dużo bardziej skupione na współpracy. Unika się tu konfliktu interesów polegającego na tym, że za tę samą cenę klient chce dostać jak najwięcej, a dostawca spędzić jak najmniej czasu nad projektem. Z drugiej strony klient może nie chcieć tego typu współpracy, w obawie przed nadużywaniem czasu (a więc pieniędzy) przez dostawcę. W przypadku zwinnych metodyk nie ma takiego ryzyka z prostego powodu – to klient wraz z zespołem określa zakres iteracji a na koniec weryfikuje wykonanie zadań do których zobowiązał się zespół. Związanie płatności z akceptacją iteracji przez klienta, powoduje wyeliminowanie ryzyka ewentualnej nieuczciwości dostawcy.

Staramy się więc umowy mieć maksymalnie proste i promujące uczciwość i równość relacji, by zapewnić jak najlepszą współpracę z klientem. W końcu tak naprawdę gramy w tej samej drużynie i mamy ten sam cel – chcemy mieć dobre oprogramowanie przynoszące wartość klientowi i stałą dobrą współpracę.

Reagowanie na zmiany NAD postępowanie zgodnie z planem

Planowanie jest niezbędne. Bez niego nie bylibyśmy w stanie tak naprawdę określić czego potrzebujemy. Również zespół lubi mieć plan, bo czuje się wtedy pewnie co do tego ile ma czasu, jaki ma zakres prac, co będzie robił i kiedy. Taka potrzeba poczucia bezpieczeństwa jest czymś zupełnie normalnym a jej wypełnienie skutkuje zwiększoną równomiernością projektu. Musimy jednak brać pod uwagę fakt, że plany się zmieniają. Zmieniają się wymagania, prawo, warunki, ludzie itp. Te zmiany nieodzwrotnie mają wpływ na oprogramowanie, które tworzymy. Zakładanie, że zmiany nie będą potrzebne, że wszystko jest wiadome na samym starcie projektu było by naiwnością. Jego początek to przecież moment gdy najmniej o nim wiemy. W projektach prowadzonych zwinnie staramy się za-



planować tylko ogólnie przebieg projektu. Dokładniej planujemy część związaną z pracą przez pierwsze 1-2 miesiące. Szczegółowo planujemy 1-3 tygodnie. W ten sposób minimalizujemy szanse na pomyłkę. Oczekujemy od klienta decyzji tylko co do fragmentu, którym będziemy się w najbliższym czasie zajmować. Do tego, staramy się poszeregować wymagania zgodnie z ich wartością dla klienta i ryzykiem, tak by najważniejsze, najtrudniejsze i najbardziej ryzykowne elementy aplikacji wykonać jak najwcześniej. W ten sposób nawet jeśli nie wyrobimy się w zadanym czasie, będzie to dotyczyło tych najmniej ważnych funkcji. Natomiast jeśli w czasie pracy nad trudnymi elementami okaże się, że są one znacząco trudniejsze niż wstępnie zakładaliśmy, dajemy klientowi bardzo dużo czasu na reakcję i przeplanowanie działań, zwiększając własną wiarygodność.



Zasady Zwinnego Rozwijania Oprogramowania

Przedstawione wcześniej założenia znajdują swoje rozwinięcie w dwunastu zasadach znajdujących się na drugiej stronie Manifestu.

Naszym najwyższym priorytetem jest satysfakcjonowanie klienta przez wczesne i ciągle dostarczanie wartościowego oprogramowania.

Zwinne metodyki kładą nacisk na dostarczanie oprogramowania. Dostarczanie jest wykonywane permanentnie. Zaczyna się już na koniec pierwszej iteracji i jest wykonywane na koniec każdej następnej. Ja w projektach pilnuję, by nie było to rzadziej niż raz na tydzień, a z technicznego punktu widzenia moglibyśmy dostarczać klientowi aplikację z nowymi funkcjami prawie codziennie. Wymaga to co prawda dbania o jakość i dobrą współpracę zespołu, ale daje w zamian ogromny zysk. Dla klienta elastyczność co do wdrażania, prezentacji aplikacji użytkownikom, wczesnego wpływania na jej kształt. Dla zespołu pewność siebie i tworzonej przez siebie jakości. Dostarczanie oprogramowania, które już na wczesnych etapach ma dla klienta wartość, jest jeszcze ważniejsze, ponieważ dajemy mu szansę na częściowy zwrot kosztów tworzenia aplikacji jeszcze przed oficjalnym zakończeniem tego procesu. Częste dostarczanie nie oznacza oczywiście, że klient musi co tydzień zainstalować oprogramowanie produkcyjnie. Przez większość iteracji dotyczy to wyłącznie naszego środowiska developerskiego, lub środowiska w którym prezentujemy aplikację klientowi. Jednak kluczowa jest pewność i możliwość wdrożenia aplikacji w dowolnym momencie jej rozwoju.



Akceptuj zmieniające się wymagania nawet na zaawansowanych etapach projektu. Zwinne procesy wykorzystują zmiany dla uzyskania przewagi konkurencyjnej klienta.

Dzięki utrzymywaniu projektu przez cały czas w zintegrowanej formie, tzn.: zawsze kompilującego się, mającego automatyczne testy potwierdzające i zapewniające jego jakość i zgodność z wymaganiami, oraz automatycznie budowanego do postaci umożliwiającej wdrożenie go w dowolnym momencie, możemy akceptować, a nawet zachęcać do zmian w wymaganiach przez cały czas trwania projektu. To podejście odróżnia projekty realizowane przez zwinne zespoły od tradycyjnego podejścia, w którym próba wprowadzenia jakiegokolwiek zmiany jest tym kosztowniejsza i bardziej ryzykowna im bardziej zaawansowany jest etap projektu. Dzięki ciągłemu dostarczaniu możemy reagować na zmiany (np. w prawie, warunkach konkurencyjnych, itp.) zwykle o całe miesiące wcześniej niż tradycyjne zespoły. Dzięki temu umożliwiamy klientom natychmiastową reakcję na takie sytuacje.

Dostarczaj działające oprogramowania często, od paru tygodni do paru miesięcy, z preferencją dla krótszych okresów.

Zwinne metodyki to metodyki iteracyjne. Działamy w krótkich iteracjach, które zaczynają się od szczegółowego planowania, a kończą na prezentacji klientowi działającego oprogramowania, zawierającego dodane w czasie iteracji funkcjonalności. Ja w projektach nie używam iteracji dłuższych niż dwutygodniowe. Najbardziej odpowiadają mi tygodniowe. Czas trwania pojedynczej iteracji nie powinien przekraczać miesiąca. Krótsze, 1-2 tygodniowe, pozwalają na bardzo szczegółowe określenie zakresu iteracji (nie jest trudno znaleźć i szczegółowo określić zadania na tydzień pracy) oraz dają zespołowi lepszą kontrolę nad projektem. Do tego tydzień jest naturalnym dla nas okresem, pozwalającym na maksymalne skupienie się na wybranych zadaniach. W ten sposób otrzymujemy również bardzo szybką reakcję klienta na nowe funkcjonalności. Jest więc dużo łatwiej zrealizować dokładnie to, czego oczekuje. Przy tygodniowych iteracjach, spotkania nie-



zbędne na zaplanowanie pracy i zaprezentowanie jej klientowi są odpowiednio krótkie, co powoduje, że nie ma zwykle problemu, by je skutecznie przeprowadzać i elastycznie do nich podchodzić.

Ludzie odpowiedzialni za biznes oraz programiści muszą współpracować codziennie w trakcie projektu.

W przypadku projektów prowadzonych fazami, biznes zaangażowany jest w ogromnym stopniu na początku projektu, a następnie wymagania przekazane programistom mają im wystarczyć do stworzenia kompletnego systemu. Ten model z oczywistych powodów jest nierealny. Wymagania nie są wystarczająco jasne i kompletne, a czasem nawet są sprzeczne, więc kontakt z biznesem w czasie tworzenia oprogramowania i tak jest niezbędny.

W zwinnych projektach nie staramy się udawać, że spisane wymagania są w stanie dokładnie odzwierciedlić oczekiwania biznesu co do tworzonego systemu. Co więcej, nie udajemy, że jesteśmy ekspertami od wszystkich dziedzin z jakimi przydarza nam się spotykać w naszej pracy. Sam pisałem m.in. aplikacje ubezpieczeniowe, telekomunikacyjne, system portalowy, systemy finansowe. Nie możliwe było by znać się na wszystkim. Staramy się więc wykorzystać wiedzę ekspertów, czyli oczywiście naszych klientów, często użytkowników naszych aplikacji. Dzięki temu nie tylko zwiększamy szanse na odzwierciedlenie w aplikacji wymagań biznesu, ale również zmniejszamy prawdopodobieństwo wystąpienia pomyłek.

Twórz projekty wokół zmotywowanych osób. Daj im środowisko i wsparcie którego potrzebują i zaufaj, że wykonają swoją pracę.

Nie jest prawdą, że wystarczy wprowadzić zwinną metodykę, by zapewnić sukces projektu. Dobór i właściwe prowadzenie osób pracujących w projekcie są o wiele ważniejsze niż wszelkie procesy. Dlatego podstawowym wymaganiem jest zmotywowanie osób zaangażowanych w projekt. Zmotywowany zespół jest wielokrotnie bardziej wydajny niż grupa sfrustrowanych i znudzonych projektem programistów. To jest pułapka w którą wpada



wiele zespołów wprowadzających samodzielnie zwinne metodyki. Wielokrotnie dzieje się to poprzez odgórne zalecenie kierownictwa i poza wprowadzeniem dodatkowych spotkań i formalności nie daje żadnych efektów, a nawet może doprowadzić do pogrążenia projektu.

Zespół programistyczny musi być zmotywowany, chętny do pracy, lubiący ją i ceniący się nawzajem. Ale, żeby zespół mógł osiągnąć taki stan, musi mieć dobre środowisko do swojego rozwoju. Jest tu więc miejsce dla trenerów – osób z doświadczeniem we wprowadzaniu zwinnych metodyk. Mogą oni pomóc zespołowi, a czasem nawet całej organizacji w zmianie. Poza tym zespół musi mieć wsparcie i zaufanie kierownictwa. Oznacza to, że poza odpowiedzialnością za projekt musi dostać odpowiadające jej uprawnienia. Jeśli zespół ma brać odpowiedzialność za jakość, musi móc na tę jakość wpływać – musi więc mieć prawo do eksperymentowania, wprowadzania wybranych przez siebie narzędzi i praktyk, w końcu musi mieć czas by o tę jakość zadbać. Jeśli ma brać odpowiedzialność za terminowość, musi mieć wpływ na terminy w projekcie – na oszacowania pracochłonności, na rozkład zadań, w końcu nawet na terminy wdrożeń (lub zakres funkcjonalności, która powinna być wdrożona). Niestety rzeczywistość zwykle wygląda zupełnie inaczej. Terminy i zakres funkcjonalności często dane są z góry, bez żadnych konsultacji z zespołem, a wynikają z ustaleń np. działu sprzedaży z klientem. Jakość ma niby być najwyższa, ale zespół działa cały czas pod presją czasu i utraty premii w przypadku opóźnień. W takiej sytuacji premia dla zespołu będzie elementem działającym demotywująco. Zespół tak naprawdę nie będzie miał możliwości jej otrzymania i to ze świadomością, że tak naprawdę nie z własnej winy.

Zwinne zespoły i ich pracę staramy się więc organizować inaczej – a dokładniej staramy się jej nie organizować w ogóle. Staramy się pozwolić im zorganizować się samym, w sposób kontrolowany. Dzięki temu możemy liczyć na dużo większą kreatywność i zaangażowanie. Często używa się tu terminu *servant leadership* – prowadzenie poprzez służenie. Kierownicy takich zespołów muszą świadomie pozbyć się części swojej władzy i przekazać ją zespołowi, by umożliwić mu samoorganizację i wypełnianie danej mu odpowiedzialności. Ich rola w takich zespołach to wspieranie („A może potrzebujecie eksperta od Oracle’a?”), pomaganie w rozwiązywaniu konfliktów („Panowie, obniżcie trochę emocje i przedstawmy sobie konkretne argumenty.”), usuwanie problemów z którymi nie mogą sami sobie poradzić („Pogadam z adminami, będziecie mieć SVN’a działającego za godzinę.”),



a nawet angażowanie zespołu w zadania pozaprogramistyczne („Jutro o dwunastej mam spotkanie z klientem, ktoś ma ochotę wziąć udział? Potrzebuję kogoś technicznego, żeby wytłumaczył im dlaczego integracja z ich systemami tyle trwa.”). Co więcej, taki kierownik musi wielokrotnie powstrzymywać się, lub nawet pozwolić się powstrzymać zespołowi, przed narzucaniem mu nadgodzin pracy, kolejności zadań, czasem nawet praktyk programistycznych.

Najwydajniejszym i najskuteczniejszym sposobem przekazywania informacji do i w ramach zespołu programistów jest rozmowa twarzą w twarz.

Wiele firm ma kulturę przekazywania sobie wszelkich informacji emailem lub nawet na papierze. Ma to co prawda swoje dobre strony, ale tych złych jest niestety chyba więcej. Informacja przekazywana słownie jest najszybsza. Co więcej pozwala na jej uszczegółowienie i wyjaśnienie bez tracenia czasu. Dlatego w zwinnych zespołach promuje się zwykłą rozmowę a programiści rzadko wysyłają do siebie maile. Również w komunikacji biznesu z zespołem staramy się zmaksymalizować ilość informacji przekazywanych ustnie i to najlepiej twarzą w twarz. Chcemy być pewni, że dostajemy informacje pełne, dokładne i aktualne, a to najłatwiej wyjaśnić w bezpośredniej rozmowie. Co więcej chcemy dostawać jak najwięcej informacji właśnie wtedy, kiedy ich potrzebujemy, by nie musieć tracić czasu na ich odszukiwanie i zastanawianie się nad ich aktualnością. Bezpośrednia rozmowa jest też mechanizmem budującym wzajemne zaufanie i ułatwiającym współpracę. Czym jest jej więcej, tym łatwiej nam się nawzajem zrozumieć i tym efektywniej pozyskujemy wiedzę niezbędną do napisania dobrej aplikacji.

Nie oznacza to, że oczekujemy, że połowa firmy klienta będzie teraz tylko czekać, aż ktoś z zespołu zada im pytanie. Większość rzeczy ustala się w trakcie spotkań w czasie których planujemy naszą pracę. Te spotkania są dla nas momentem pozyskiwania aktualnej wiedzy o projekcie, szacowania czasochłonności zadań i prowadzenia dyskusji na tematy związane z implementacją nowych funkcjonalności. Pytania zespołu zadawane w czasie iteracji to zwykle więc tylko prośby o wyjaśnienia i uszczegółowienia wymagań, oraz wątpliwości (a nawet sprzeczności) pojawiające się w czasie rozwijania oprogramowania. Te zwykle są dość drobiazgowo i nietrywialne, formułowanie ich pisemnie często



trwa długo, a odpowiedź na nie potrzebna jest bardzo szybko. Stąd staramy się o bliską współpracę z klientem, by móc takie wątpliwości wyjaśniać na bieżąco, co zwiększa tempo naszej pracy i pozwala nam lepiej dopasowywać aplikacje do potrzeb ich użytkowników.

Działające oprogramowanie jest podstawową miarą postępu

Ani dokumentacja, ani wykresy gantt'a, ani żadne narzędzia nie określą prawdziwego stanu i etapu zaawansowania projektu. Jediną wiarygodną miarą tego, że projekt postępuje w dobrym kierunku jest stan tworzonego oprogramowania. Dlatego w zwinnych projektach informujemy o postępach poprzez prezentację działającej aplikacji przynajmniej raz w każdej iteracji. Dzięki temu na bieżąco dostarczamy klientowi dane dotyczące faktycznego zaawansowania projektu. Sam może sprawdzić co już jest zaimplementowane, jak to co jest ma się względem planów, jakie są szanse na wykonanie wszystkich oczekiwanych funkcjonalności w terminie.

Pomimo tego, zwinne zespoły przez cały czas śledzą i mierzą swoje postępy. Prawdopodobnie robią to częściej i z większym przekonaniem niż tradycyjnie prowadzone zespoły. Regularnie mierzymy naszą prędkość, by móc wiarygodnie szacować czasochłonność przyszłych funkcjonalności oraz pomóc klientowi planować wydania pod kątem ich zakresu funkcjonalnego i terminów. Weryfikujemy wpływ naszych działań i pomysłów na naszą wydajność, by poprawiać proces i praktyki wykorzystywane do tworzenia oprogramowania.

Zwinne procesy promują równomierne rozwijanie oprogramowania. Sponsorzy, programiści i użytkownicy powinni być w stanie utrzymywać równomierne tempo nieskończenie.

Klasyczne metodyki skupiają się na dostarczeniu projektu. Celem jest jednorazowy sukces i wszelkie praktyki skupiają się właśnie na tym. Takie podejście promuje nierównomierne skupienie na projekcie. Na początku jest dużo czasu, leniwa analiza, powolny start developmentu. Potem projekt postępuje przez jakiś czas niby całkiem dobrze, choć



dokładny status znają tylko programiści. Na miesiąc przed wdrożeniem zaczyna się wyścig z czasem, kosztem jakości oprogramowania i życia prywatnego programistów.

W zwinnych projektach staramy się odejść od klasycznego modelu z wydaniem, CR'ami, nieprzejrzystym przez większą część projektu stanem aplikacji. Przyjmujemy w miejsce tego zasadę, że aplikacja jest gotowa do wdrożenia na koniec KAŻDEJ iteracji – nawet tej pierwszej. My staramy się mieć zintegrowaną aplikację przez cały czas tzn.: zadbane środowisko *Continuous Integration*, aktualna dokumentacja, zautomatyzowane budowanie pakietu gotowego do wdrożenia. W zamian otrzymujemy równomierną pracę, pozwalającą na wiarygodne określanie terminów faktycznych wdrożeń, na stałe godziny pracy bez maratonu przed wdrożeniem, na przezroczysty proces i jasny stan aplikacji na dowolnym etapie jej rozwoju. Niestety aby osiągnąć taki efekt, cała organizacja projektu musi działać podobnie. Sponsorzy muszą równomiernie zarządzać budżetem, biznes znajdować czas na spotkania z zespołem, wdrożenia muszą być możliwe zawsze kiedy biznes sobie tego będzie życzył lub w krótkich okresach, by móc zdobywać często wdrażaną aplikacją przewagę konkurencyjną.

Ciągłe skupienie na technicznej doskonałości i dobrym zaprojektowaniu aplikacji zwiększa zwinność.

Doskonałość oraz ciągle doskonalenie się techniczne zespołu to niewątpliwie podstawa dobrego projektu. Nie jest łatwo zagwarantować jakąkolwiek rozsądną jakość w zespole składającym się wyłącznie z młodych, niedoświadczonych programistów. Trudno też o możliwość ciągłego rozwoju projektu w stałym tempie, gdy sposób w jaki został zaprojektowany uniemożliwia jakiegokolwiek zmiany. Zwinność to możliwość reagowania na zmiany nie tylko w sensie realizacji zmieniających się wymagań, ale czysto technicznie dotyczy również ciągłej możliwości implementacji wymagań w równomiernym tempie. Zwinność techniczna to takie strukturalizowanie kodu, by nie zagrozić możliwości implementacji przyszłych wymagań. Nikt niestety nie umie tego od urodzenia. To wiedza i doświadczenie które zdobywa się pracując nad kodem. Pracując z lepszymi od siebie. Pracując cały czas nad sobą.

W zwinnych zespołach odnosimy się do japońskiej filozofii *kaizen* – staramy się per-



manentnie poprawiać swoją wiedzę, zwiększać swoją jakość jako programiści. Staramy się robić to świadomie, a nie jako element ciekawości czy młodzieńczej fascynacji nowinkami. Uczymy się sami (książki, fora dyskusyjne, grupy) i douczamy siebie nawzajem poprzez pracę w parach. Dbamy w ten sposób o jakość naszych rozwiązań, ponieważ w parze dużo trudniej o niedopatrzenie, pomyłkę, czy zwykłe lenistwo a łatwiej wspólnie znajdować lepsze i czytelniejsze rozwiązania.

Prostota – sztuka maksymalizowania pracy niewykonanej – jest zasadnicza.

W tradycyjnym podejściu do projektów i tworzenia oprogramowania zakłada się maksymalizację ogólności rozwiązań, tak by nie ograniczać przyszłych możliwości ich rozwoju. Było by to rozsądne podejście, gdyby nie fakt, że większość z tych uogólnień i generycznych rozwiązań nigdy nie znajduje praktycznego wykorzystania. Marnowany jest za to czas i budżet na rzeczy, co do których nie ma pewności czy będą potrzebne.

W zwinnych projektach staramy się skupić na zadaniach, które faktycznie są niezbędne. Na takim ich wykonaniu, by zapewnić ich poprawność i kompletność dla przypadków których istnienia jesteśmy aktualnie pewni. Oczywiście cały czas myślimy też o tym by nie zamknąć sobie drogi do zmian. Dlatego promujemy rozwiązania proste w wykonaniu, a wystarczające z punktu widzenia wymagań, oraz pozwalające na ich modyfikację i zmianę w przyszłości. W ten sposób możemy dostarczać oprogramowanie szybciej (implementujemy tylko to co faktycznie jest niezbędne), zostawiając sobie pole do reagowania na zmiany. Zupełnie konkretnie oznacza to modułową architekturę, skupienie na prostocie i jakości kodu (m.in. zasady S.O.L.I.D.)

Najlepsze architektury, wymagania i projekty powstają w samoorganizujących się zespołach.

Zwinne metodyki stawiają na zespołowość. Więcej osób ma po prostu większą szansę na znalezienie lepszego rozwiązania niż jedna. Poza tym dyskusje i wymiana argumentów w zespole powoduje zwiększenie jego jakości poprzez wzajemne uczenie, a więc również jakości tworzonych przez niego produktów. Ponadto praca w zespole mobilizuje, nie po-



zwala oddać się znużeniu i dużo trudniej poddaje się obniżeniu jakości pod wpływem zmęczenia. Przynosi więc lepsze i pewniejsze efekty. Samoorganizujące się zespoły potrafią ponadto wypracować sobie własne bardzo skuteczne sposoby pracy co nie pozostaje bez wpływu na projekt.

Nie oznacza to, że nie pracujemy indywidualnie. W każdym projekcie można znaleźć dużo pracy, której wykonanie jest skuteczniejsze w pojedynkę. Staramy się zawsze podchodzić do tego pragmatycznie. Kod produkcyjny powstaje prawie wyłącznie w parach. Natomiast dokumentacja, konfiguracja środowisk, poznawanie nowych technologii, i tym podobne, wykonywane są w jednoosobowo.

W regularnych odstępach czasu zespół zastanawia się jak poprawiać swoją efektywność, a potem dostraja i zmienia odpowiednio swoje zachowania.

Aby reagować na zmiany, zespół musi dostosowywać własny kształt, procesy i praktyki do zmieniających się okoliczności. Może się zdarzyć, że trzeba zmienić skład zespołu, by działał efektywnie – np. dodać do niego specjalistę od interfejsów użytkownika. Może to również dotyczyć wprowadzania pewnych praktyk (np. programowanie w parach), czy zmian terminów spotkań, tak by wszyscy mogli na równych zasadach w nich uczestniczyć. Może też wymagać przeorganizowania pokoju tak, by zapewnić lepszy kontakt między członkami zespołu. Żeby dostrzec takie potrzeby, zespół musi regularnie patrzeć wstecz i starać się dostosowywać do zachodzących zmian.

My staramy się również zastanawiać jak reagowaliśmy na różne sytuacje w czasie iteracji, jakie emocje były z tym związane, jaki jest stan projektu z punktu widzenia naszego zespołu. Te spotkania są prywatne dla zespołu tak więc staramy się być ze sobą maksymalnie szczerzy i uczciwi, przez co budujemy wzajemne zaufanie. To właśnie ono przekłada się potem na jakość naszej współpracy a w efekcie na jakość aplikacji i skuteczność jej tworzenia.

Podsumowanie

Zwinne rozwijanie oprogramowania to dużo więcej niż tylko jakiś proces. To praca nad zespołem i nad jakością oprogramowania. To przezroczystość działań i uczciwość wobec klienta. To także droga ciągle wymagająca weryfikacji, pracy i zaangażowania. Wymagająca nieustannego skupienia na pracy, na zespole w którym się pracuje, na wdrażanych praktykach i wyznawanych wartościach. Polegająca na wyszukiwaniu ciągle nowych, lepszych sposobów na wykonywanie tego co jest naszym zawodem: wytwarzania oprogramowania. To również dużo dobrej zabawy, zadowolenie z wykonywanej pracy, szacunek klientów.

Nic jednak nie przychodzi samo. Transformacja zespołu na zwinne myślenie, efektywne wykorzystywanie zwinnych praktyk i procesów zajmuje dość dużo czasu. Pokonywanie starych nawyków i poznawanie nowych praktyk jest trudne i często mozolne. Mówi się nieraz, że ważniejsze od tego czego trzeba się nauczyć jest to czego trzeba się oduczyć. Do tego dochodzi pokonywanie tradycji i przyzwyczajzeń społeczno-kulturowych. Dla zespołu może to być branie odpowiedzialności, a dla kierownictwa przekazanie jej wraz z uprawnieniami niezbędnymi do jej wypełnienia. Przychodzi to dużo łatwiej gdy ma się pomoc w postaci doświadczonego trenera, który pomagał już zespołom w transformacji i wie jak powinna wyglądać praca zwinnego zespołu.



Pragmatists Sp. z o.o.

Jesteśmy firmą zajmującą się tworzeniem oprogramowania oraz świadczeniem profesjonalnych usług dla zespołów developerskich.

Naszą działalność skupiamy w czterech podstawowych obszarach:

- **rozwój oprogramowania**

wykorzystujemy zwinne metodyki – Scrum / eXtreme Programming

staramy się pracować możliwie blisko Klienta, by jak najlepiej zrealizować w stworzonych przez nas produktach jego oczekiwania

- **coaching zespołów**

pomagamy wdrażać zwinne metodyki oraz przekształcać zespoły i organizacje na pracę w zwinnym modelu

- **audyt istniejących aplikacji, weryfikacja architektur względem wymagań**

weryfikacja jakości i odpowiedniości względem wymagań aplikacji tworzonych przez innych podwykonawców, weryfikacja architektur proponowanych dla nowych (dużych) systemów pod kątem wymagań stawianych tym systemom

- **ratowanie projektów**

przejmujemy development całych projektów lub dołączamy do istniejących zespołów na zasadzie dodatkowych sił i mentoringu

Wierzymy, że jakość produktu odzwierciedla odpowiedzialność jego twórców. Dlatego przy tworzeniu oprogramowania na jakość właśnie kładziemy szczególny nacisk. Stosujemy rygorystyczną weryfikację zarówno na poziomie funkcjonalnym (poprawność działania aplikacji) jak i w samym kodzie (jakość kodu stale weryfikowana jego statyczną analizą oraz testami jednostkowymi). Dzięki temu tworzymy oprogramowanie prawie nie wymagające utrzymania, co potwierdzamy w umowach serwisowych – nasze stawki są tu minimalne, pokrywające wyłącznie koszt utrzymania środowisk developerskich i testowych.



Zatrudniamy wyłącznie doświadczonych programistów. Stosujemy iteracyjny proces rozwoju oprogramowania z częstą i regularną kontrolą poprawności i zgodności aplikacji z oczekiwaniami, dzięki czemu Klient ma zawsze możliwość zmieniać zdanie i modyfikować/precyzować wymagania. Używamy systemów ciągłej integracji, dzięki czemu Klient ma możliwość sprawdzania postępów i przetestowania aplikacji na dowolnym etapie jej rozwoju, a nawet zdecydować o jej wdrożeniu w dowolnym momencie, jeśli dojdzie do wniosku, że oferowana przez aplikację na danym etapie funkcjonalność jest wystarczająca do dostarczenia jej użytkownikom końcowym.

Zapraszamy do współpracy!

Paweł Lipiński

Prezes zarządu Pragmatists Sp. z o.o.

telefon: +48 691-512-774

email: pawel.lipinski@pragmatists.pl

web: www.pragmatists.pl

skype: [pawel.lipinski](https://www.skype.com/people/pawel.lipinski)