[**Agile Software Development**](http://projekty2.4inno.pl/2010/04/zwinne-programowanie-czyli-agile-software-development/)

**Zwinne programowanie**

Woliński Radosław

Programowanie i Bazy danych

Nr Indeksu 14618

Programowanie zwinne (([ang.](http://pl.wikipedia.org/wiki/J%C4%99zyk_angielski)) *Agile software development*) to grupa metodyk wytwarzania oprogramowania opartego na [programowaniu iteracyjno-przyrostowym](http://pl.wikipedia.org/wiki/Model_przyrostowy). Metodyka ta jest oparta na zdyscyplinowanym zarządzaniu projektem, zakłada częste inspekcje wymagań i rozwiązań wraz z procesami adaptacji (zarówno specyfikacji jak i oprogramowania). Najczęściej znajduje zastosowanie w mniejszych firmach, małych zespołach programistycznych, w których nie występuje problem komunikacji, przez co nie trzeba tworzyć rozbudowanej dokumentacji kodu. Kolejne etapy wytwarzania oprogramowania zamknięte są w iteracjach, w których za każdym razem przeprowadza się testowanie wytworzonego kodu, zebranie wymagań, planowanie rozwiązań itd. Nastawiona jest na szybkie wytwarzanie oprogramowania wysokiej jakości.

W metodyce tej nacisk kładzie się również na otrzymywanie rentownych funkcjonalnych części, by wnosiły one dużo znaczenia do biznesu. Agile jest pewnego rodzaju spójnym procesem rozwoju a nie pojedynczym.

Metodyka ta ma także pewne reguły, których stara się przestrzegać. Najważniejsza tu jest:

* osiągnięcie satysfakcji klienta poprzez szybkość wytwarzania oprogramowania,
* działające oprogramowanie jest dostarczane okresowo (raczej tygodniowo niż miesięcznie),
* podstawową miarą postępu jest działające oprogramowanie,
* późne zmiany w specyfikacji nie mają destrukcyjnego wpływu na proces wytwarzania oprogramowania,
* bliska, dzienna współpraca pomiędzy biznesem a developerem,
* bezpośredni kontakt, jako najlepsza forma komunikacji w zespole i poza nim,
* ciągła uwaga nastawiona na aspekty techniczne oraz dobry projekt (design),
* prostota,
* samozarządzalność zespołów,
* regularna adaptacja do zmieniających się wymagań

Bez względu na wymaganą dyscyplinę rozwoju każdy zespół Agile musi posiadać w swoim składzie przedstawiciela klienta. Taką osobę powołują zainteresowane strony do działania w ich imieniu, sprawia że są oni zaangażowaniu i dostępni dla programistów, aby odpowiadać na pytania dotyczące projektu.

Na końcu każdej iteracji, interesariusze oraz przedstawiciele klienta dokonują przeglądu postępów oraz ponowną ocenę priorytetów w celu optymalizacji tzw. zwrotu z inwestycji i zapewniają dostosowanie do potrzeb klientów i celów firmy.

Zasady tej metodyki można odnaleźć w tzw. Lean Manufacturing i SixSigma. Warto także dodać, że samo pojęcie metodyki Agile eliminuje niepotrzebne straty (np. w czasie, tworzy odpowiedni przepływ, upełnomocnia pracowników). Termin ten po raz pierwszy został zastosowany w tzw. 14 zasadach „Toyota Way”, w dwóch filarach Systemu Produkcyjnego Toyoty.

Pojęcie *zwinnego programowania* zostało zaproponowane w [2001](http://pl.wikipedia.org/wiki/2001) w [Agile Manifesto](http://pl.wikipedia.org/wiki/Manifest_Agile) (Manifest Zwinnego Wytwarzania Oprogramowania).

Powstały Manifest Agile Software Development wyraża cztery podstawowe wartości, na których bazują metodologie Agile:

* ludzie i ich wzajemne oddziaływania ponad procesy i narzędzia,
* działające oprogramowanie ponad obszerną dokumentację,
* współpraca z klientem ponad negocjowanie kontraktów,
* reakcja na zmianę ponad realizację planu.

Wydaje się, że manifest marginalizuje znaczenie procesów i narzędzi, dokumentacji, twardych negocjacji z klientem oraz postępowania zgodnie z planem. Należy jednak pamiętać, że manifest nie neguje zasadności istnienia tych elementów, lecz kładzie nacisk na te aspekty wytwarzania oprogramowania, którym dotychczas nie poświęcano wystarczającej uwagi lub pomijano w klasycznej metodzie wodospadowej (waterfall).

**Ludzie i ich wzajemne oddziaływania ponad procesy i narzędzia**

W klasycznym podejściu zgodność z procesem lub narzędziem jest często celem samym w sobie. Postrzegana jako kryterium sukcesu projektu, a proces zwykle jest bardzo sztywny i trudny do zmiany. Pewną tendencją jest bezkrytyczne wdrażanie procesu w danym projekcie, pomimo że nie adresuje on jego potrzeb. Dodatkowo, w klasycznym podejściu do wytwarzania oprogramowania, komunikacja jest bardzo sformalizowana i odbywa się poprzez dokumenty lub systemy.

Zwinne metodologie traktują procesy i narzędzia jako wsparcie dla ludzi w podejmowaniu decyzji, zapewnianiu jakości, czy utrzymaniu kontroli nad projektem. Natomiast jeżeli zachodzi taka potrzeba, mogą w części lub całości zostać zmodyfikowane, a w skrajnych przypadkach pominięte. Zwinne metodologie kładą nacisk na bezpośredni kontakt miedzy ludźmi jako najefektywniejszą formę przekazywania informacji, promują wspólne, zespołowe podejmowanie decyzji w przeciwieństwie do eksperckich decyzji w klasycznym podejściu (np.: szacowanie rozmiaru wymagań, czasu realizacji danego wymagania, wyboru technologii lub zewnętrznego komponentu są dokonywane gremialnie, w oparciu o rzeczową dyskusję). Zaletą takiego podejścia jest podejmowanie lepszych decyzji, bazujących nie na jednej, ale na różnych perspektywach, a także poprawa integracji w zespole i identyfikacji z projektem.

**Działające oprogramowanie ponad obszerną dokumentację**

Wodospadowy (waterfall) model wytwarzania oprogramowania zakłada długą fazę analizy wymagań oraz projektowania całości systemu. Owocuje on dużą ilością dokumentacji opisującą implementację w bardzo dokładny sposób. Na etapie planowania bardzo często umysł ludzki nie jest w stanie przewidzieć wszystkich aspektów złożoności oprogramowania, jak i ograniczeń wybranej technologii. W fazie wytwarzania oprogramowania okazuje się, że cześć udokumentowanego projektu jest niemożliwa, bądź bardzo trudna do implementacji. To z kolei wymusza zmiany zarówno w projekcie, jak i w istniejącej już dokumentacji, powodując dodatkowy narzut czasowy. Zazwyczaj zmiany architektury projektu w jednym miejscu pociągają za sobą zmiany w innej, zależnej części. To sprawia, że czas potrzebny na aktualizację dokumentacji znacząco się wydłuża. Ostatecznie okazuje się, że nie jesteśmy w stanie odzwierciedlić w dokumentacji wszystkich zmian, co czyni ją nieaktualną, a czas poświęcony na szczegółowe projektowanie jest czasem zmarnowanym (nie mamy ani aktualnej dokumentacji ani działającego kodu).

Zwinne metodologie podchodzą do projektowania i dokumentowania w nieco inny sposób, nie umniejszając przy tym ich wartości. Projektowanie jest rozłożone w czasie trwania projektu, a dokumentacja zawiera tylko elementy istotne i warte dokumentacji. System jest projektowany i implementowany moduł po module, a dokumentowane są tylko jego najważniejsze elementy (bardzo złożone algorytmy, podstawowy model danych, interfejsy publiczne mające wpływ na inne części oprogramowania). W ten sposób zmniejszamy ryzyko związane z podejściem do integracji kodu. W klasycznym modelu integracja jest dokonywana na końcu cyklu wytwarzania oprogramowania, a do tego czasu każdy moduł jest implementowany i testowany osobno. Podejście takie zwiększa prawdopodobieństwo niespójności modułów oraz znalezienia znacznej ilości błędów związanych z ich integracją pod koniec cyklu. W metodzie zwinnej kod jest projektowany, implementowany i integrowany cześć po części, co powoduje, ze każda nowa funkcja jest projektowana z uwzględnieniem już zaimplementowanych modułów.

**Współpraca z klientem ponad negocjowanie kontraktów**

W klasycznym podejściu kontakt z klientem docelowym następuje na początku i na końcu cyklu wytwarzania oprogramowania. Na początku zbiera się wymagania klienta, które pozostają niezmienne aż do samego końca. Natomiast na końcu projektu następuje walidacja wytworzonego oprogramowania. Świat oprogramowania niestety nie jest statyczny i w ciągu trwania projektu klient często zmienia swoje wymagania. Źródłem zmiany wymagań klienta jest potrzeba adaptacji do zmian w jego środowisku (np.: wejście w życie nowej ustawy, wdrożenie nowego systemu lub procesu). W efekcie okazuje się, że wytworzone oprogramowanie nie spełnia całkowicie lub częściowo "nowych" wymagań klienta. Ponadto okazuje się, że część funkcjonalności, która na początku wydawała się ważna przestała być potrzebna. Wszystko to prowadzi do renegocjacji kontraktu lub dodatkowych zobowiązań ze strony wytwórcy oprogramowania w celu utrzymania dobrych stosunków z klientem, co z kolei odbija się na budżetach wytwórcy jak i klienta.

Zwinne metodologie bazują na zaangażowaniu i ciągłej współpracy z klientem oraz walidacji jego wymagań w czasie trwania projektu. Ma to na celu wyeliminowanie mało istotnych wymagań przy jednoczesnym utrzymaniu budżetu projektu. Ciągła współpraca z klientem realizowana jest poprzez dostarczanie klientowi w regularnych odstępach czasu działającego oprogramowania zawierającego najbardziej wymagane funkcje, a następnie analizowanie informacji zwrotnej w celu aktualizacji priorytetów. Jednym ze sposobów dostarczenia oprogramowania klientowi jest przeprowadzenie demonstracji najnowszej funkcji lub wdrożenie jej bezpośrednio u klienta. Aby dostarczać działające oprogramowanie zawierające najbardziej wymagane funkcje w regularnych odstępach czasu, zwinne metodologie zalecają implementowanie nowych funkcji w stałych odstępach czasowych trwających od 1 do 4 tygodni (iteracje) oraz integrowanie ich z poprzednio stworzonymi (inkrementacje).

**Reakcja na zmianę ponad realizację planu**

Wodospadowa metodologia zakłada, że planowanie występuje tylko na początku projektu a po nim następuje wykonanie wszystkich przewidzianych zadań. Na planowanie poświecą się bardzo dużą ilość czasu przeznaczonego na realizację jak najbardziej szczegółowego projektu. Takie podejście do planowania zakłada niezmienność wymagań klienta, nieomylność estymat (np.: wstępnych kosztów) podanych przez zespól lub eksperta, stabilność środowiska, pełną dostępność zespołu. Bardzo często jedno lub kilka z tych założeń nie wytrzymuje konfrontacji z rzeczywistością modyfikując plan: klient zmienia wymaganie, najbardziej doświadczony programista odchodzi do innej firmy itp. Ponadto zmiany w tak szczegółowym planie są bardzo trudne do wprowadzenia ze względu na liczbę zadań oraz zależności miedzy nimi. Zwinne zespoły wyróżniają trzy poziomy planowania: całość procesu wytwórczego (iteracja - pojedynczy dzień). Każdy kolejny poziom opisuje coraz krótszy okres czasu na coraz niższym poziomie abstrakcji. Planowanie, podobnie jak w przypadku projektowania rozłożone jest w czasie. Iteracyjne podejście do planowania następujących po sobie cykli wytwórczych umożliwia zespołowi stosunkowo szybką odpowiedź na zmieniające się warunki zewnętrzne (potrzeby rynku, nowe technologie, itp. ). Im krótsze iteracje, tym szybciej zespół ma możliwość reagowania na nowe fakty w planowaniu (zasadą jest, że iteracji nie przerywamy w połowie).

Podział planowania na trzy poziomy zapewnia dodatkowo dobrą kontrolę nad aktualnym i przewidywanym stanem projektu. Planowanie projektu w skrócie można opisać jako posortowanie wymagań według rozmiaru i ważności oraz przypisanie ich do iteracji. Planowanie iteracji to analiza i rozbicie wymagań na zadania. Planowanie dnia to zobowiązywanie się członków zespołu do wykonania zadania z listy iteracji. Postęp pracy zmierzony po każdym dniu daje nam wiedzę o stanie iteracji,a postęp pracy po każdej iteracji daje nam wiedzę o stanie projektu. Wraz z upływem czasu zespół z coraz większą dokładnością jest w stanie określić datę wypuszczenia produktu na rynek. Godzimy się z faktem, że w początkowej fazie data określona jest z dokładnością do kwartału, po upływie kilku iteracji, zespół jest w stanie zapewnić dostarczenie produktu w określonym miesiącu, by następnie określić datę z dokładnością do dnia.

Podsumowując, zwinne metodologie wytwarzania oprogramowania charakteryzują się bezpośrednią i otwartą komunikacją w zespole, wspólnym podejmowaniem decyzji oraz naciskiem na ciągły kontakt z klientem.

**Najbardziej popularne odmiany zwinnych metodologii to:**

* SCRUM
* Extreme Programming (XP)
* Feature Driven Development (FDD)
* Crystal
* Adaptive Software Development (ASD)

**eXtreme Programming (XP)**

Najbardziej znana z metodyk zwinnych. Kieruje się czterema wartościami takimi jak,

komunikacja (większość projektów nie udaje się z powodu jej braku), prostota (wykonanie najprostszego produktu, który odpowiada potrzebom klienta), informacja zwrotna (otrzymywanie wiadomości zwrotnej od klienta, systemu i reszty zespołu), odwaga (podejmowanie trudnych decyzji).

XP wymaga rygorystycznego trzymania się ustalonych praktyk:

* ciągłej obecności klienta,
* gry planistycznej – klienci i zespół negocjują wymagania w formie historyjek na karteczkach (stories),
* krótkich cykli – maksymalnie 3 tygodniowe,
* zbiorowej własności – każdy może pracować nad wszystkim,
* programowania w parach (pair programmig) – dwóch programistów pracuje nad jednym kodem,
* prostego projektu (simple design),
* refaktoringu – kiedy kod zaczyna być nieczytelny, porządkujemy go bez zmieniania jego działania,
* standardu kodowania – ustalone i ściśle przestrzegane przez zespół,
* testowanie w pierwszej kolejności (test first) – najpierw tworzy się przypadki testowe, potem przygotowuje kod, który te przypadki przejdzie,
* ciągłej integracji (continuous integration) – każdego dnia oprogramowanie jest scalane (z wersji każdego z deweloperów) i powinno działać,

**Adaptive Software Development (ASD)**

ASD składa się z cykli spekulowania, opracowywania i uczenia (speculate, collaboration, lern cycles). Najważniejsze jest ciągłe uczenie i adaptacja do obecnego stanu projektu. Skoncentrowany na misji, planujący w oparciu o funkcje, sterowany ryzykiem, tolerancyjny na zmiany. ASD ma za zadanie balansować na krawędzi chaosu, dlatego konieczny jest udział ekspertów. Może być trudno wykorzystać tę metodykę w większych projektach.

Praktyki wymagane przy ASD:

* iteracyjny rozwój (iterative development)
* planowanie oparte na funkcjach (feature-based planning)
* grupy focusowe klientów (focus groups)

**Crystal**

Inspiracja pochodziła z metodyk zwinnych, sterowanych planem, a także psychologii i wieloletnich badań autora nad rozwojem organizacji. Twórca (Alistar Cockburn) opisuje ją jako rodzinę metodyk opartych na ludziach, adaptujących się, ultralekkich i dopasowujących się do potrzeb.

Crystal występuje w różnych postaciach w zależności od wielkości zespołu i ryzyka w projekcie. Kryształ (crystal) charakteryzuje kolor i twardość. Każdy z wariantów kolorów (liczba ludzi) ma swoją nazwę (Clear, Yellow, Orange, Red).

Warianty twardości (ryzyko) to:

* komfort (Comfort),
* niekluczowe fundusze (Discretionary money),
* kluczowe fundusze (Essential money),
* życie (Life).

Przy dobieraniu opcji twórca poleca kierować się myślą: „lighter is better as long it lasts”.

**Scrum**

Scrum jest podejściem skoncentrowanym na zarządzaniu projektem. Jego nazwa wywodzi się z młyna (scrum) w rugby. Analogia ta nawiązuje do szybkiego dopasowywania się do okoliczności. Wprowadzenie Scruma jest proste – nie wymaga przeprowadzania zmian w organizacji. Odnotowano też sukcesy w wykorzystywaniu tej metodyki w większych projektach. W takich przypadkach jest jednak konieczny udział ekspertów od Scruma.

Praktyki wymagane przy Scrum:

* sprinty – równe okresy, w których przygotowuje się kolejne funkcje i prezentuje klientowi (np. 30 dni),
* backlog – lista wymagań (z priorytetami) do wykonania w projekcie (product backlog) lub sprincie (sprint backlog)
* codzienne 15-minutowe spotkania – na stojąco, każdy członek zespołu mówi o tym, co zrobił wczoraj, co będzie robił dzisiaj (monitorowanie statusu) i jakie napotkał problemy
* spotkania na rozpoczęcie (planowanie) i zakończenie sprintu (podsumowanie)
* cel sprintu (sprint goal) – kryteria uznania za zakończony sukcesem

Ważne role - Product Owner (określanie funkcji systemu, które najlepiej spełnią postawione przed nim cele), Scrum Master (dbanie o zgodność z metodyką, konsultant, trener), Scrum Team (wytwarzanie, programowanie).

**Feature Driven Development (FDD)**

FDD stawia na prosty proces, efektywne modelowanie i krótkie iteracje przynoszące wartościowy produkt. Proces powinien raczej wspierać zespół, niż nim sterować. Po starannym zebraniu i dobrym zrozumieniu wymagań ma miejsce pięć faz procesu:

* przygotowanie modelu produktu, by uchwycić zakres jego dziedziny,
* przygotowanie listy funkcji bazującej na potrzebach biznesu,
* przygotowanie planu realizacji na podstawie listy funkcji,
* przygotowanie projektu pakietów i przydzielenie funkcji do iteracji,
* zaprogramowanie funkcji

Zwinność polega na tym, że zdolni ludzie wychwycą moment, kiedy trzeba powrócić do poprzednich faz. Zakłada przydzielanie zadań do programistów i wykonanie za pierwszym razem właściwego projektu systemu. Dlatego też jest bardziej odpowiedni do systemów z większymi wymaganiami stabilności. Można go także łatwo skalować do większych projektów.

Bardzo ważne jest wykorzystywane notacji do modelowania architektury, np. UML lub innych metod obiektowych.

Ważne role - główny architekt (Chief Architect) i główny programista (Chief Programmer).