Inne spojrzenie na wytwarzanie SI – ZWINNE METODOLOGIE.

Maciej Socha

Instytut Elektroniki i Technik Informacyjnych
Politechnika Warszawska

msocha@stud.elka.pw.edu.pl

Streszczenie:

Klasyczne metody wytwarzania systemów informatycznych opierają się na 12 fazowym cyklu życia projektu. Tworzone w ten sposób systemu są bardzo sformalizowane, metodologie inżynierii oprogramowania kładą nacisk na silne algorytmiczne postępowanie opisane w specyfikacji danej metodologii. Taka droga wytwarzania jest bardzo systematyczna i uporządkowana, w założeniu ukierunkowana na sukces. Jednakże sukces definiowany jest jako realizacja zdobytego kontaktu. Wymaga przy tym ścisłego dokumentowania. Doba współczesnego biznesu ukierunkowana jest na jakość produktu, który ma być wykorzystywany a nie tylko wytwarzany, modne staje się certyfikowanie swoich produktów. Kryteria ISO 9001 mówią między innymi o istocie czynników wpływających na sukces wytwarzanego oprogramowania. Jedną z podstawowych cech dobrego procesu wytwórczego jest współpraca z klientem (często użytkownikiem). W myśl za tą zasadą rozwija się grupa metodologii pod nazwą „agile software development methods". W niniejszej pracy przedstawiona będzie zasada funkcjonowania takich metod na przykładzie dwóch najpopularniejszych i szeroko stosowanych, bardzo elastycznych i nadających się zarówno do dużych projektów jak też małych zastosowań: metodyka SCRUM i FDD.

POLITECHNIKA WARSZAWSKA 2008
Praca zrealizowana na zajęcia SDM 2

1. Wprowadzenie.

Obecnie najczęściej wykorzystywane systemy informacyjne w dziedzinie ekonomii i zarządzania ukierunkowane są głównie na usprawnianie zarządzania w celu lepszego zaspokajania potrzeb wszystkich uczestników procesów gospodarczych.

Dlatego niezbędna jest właściwa metodologia projektowania i wdrażania systemów informatycznych. Inżynieria oprogramowania jest praktycznym zastosowaniem wiedzy naukowej do projektowania i tworzenia systemów informacyjnych i informatycznych oraz dokumentacji wymaganej do ich opracowania, uruchomienia oraz pielęgnacji.

Metodyki klasycznej inżynierii oprogramowania bywają czasem nadmiernie sformalizowane, dlatego w ostatnich latach zaczęto lansować bardziej swobodną metodykę projektowania systemów informatycznych, nazywaną „agile software development methods". W polskiej literaturze odpowiednie metody zwykło się określać jako „zwinne".

2. Metodyka SCRUM.

Zespół Scrum liczy zazwyczaj od 5 do 9 osób i ma charakter interdyscyplinarny, oznacza to ze również poza programistami może uczestniczyć w pracach zespołu kontroler jakości, dokumentalista, grafik, a także specjalista biznesowy lub psycholog, ma to duże znaczenie jeśli osoby w zespole posiadają różne umiejętności. Uczestnicy zespołu zajmują się tylko i wyłącznie pracą nad projektem, nie są zaangażowani w żadne inne prace, a wyjątek stanowić mogą tylko pracownicy usługowi jak np. administrator.

Zespołem kieruje osoba zwana Scrum Master`em (mistrzem młyna). Kierownik projektu pełni wyjątkową i dość specyficzną funkcję, ponieważ w jego głównym celem jest usuwanie przeszkód w pracy zespołu oraz zapewnienie zgodności wyników pracy zespołu z celami biznesowymi sponsora projektu. Kierownik zespołu nie planuje, nie przydziela ani nie kontroluje wykonywanych zadań, musi jednak dbać o przestrzeganie obowiązujących reguł (dba o samoorganizację zespołu i ogranicza powstawanie zamkniętych wewnątrzzespołowych środowisk, które prowadzą do niepowodzenia podczas realizacji projektu). Reprezentuje zespół na zewnątrz.

Istotną rolę w zespole pełni również odbiorca wytwarzanego systemu informatycznego, którego nazywa się właścicielem produktu (Product Owner). W przypadku realizowania projektu zewnętrznego, jest nim reprezentant klienta, natomiast gdy realizowany jest projekt wewnętrzny (z przeznaczeniem na rynek masowy) jest nim zwykle osoba z działu marketingu. Właściciel produktu reprezentuje głos klienta, jest dla niego gwarancją, że zespół pracuje nad właściwymi zagadnieniami z perspektywy biznesowej dla wytwarzanego systemu. Zajmuje się pisaniem „historyjek", priorytetuje ich ważność oraz rejestruje je w rejestrze wymagań (product backlog).

Jednostką pracy zespołu jest przebieg (sprint) trwający zwykle od 2 do 6 tygodni i istotne jest to, aby przebieg przez cały czas trwania projektu był jak

najbardziej stały, mobilizując tym samym cały zespół do regularnego rytmu związanego z postępem prac. Przewodnią zasadą pracy młyna jest wersja produktu, oznacza to fakt, iż każdy kolejny przebieg musi dostarczyć działającą wersję produktu dla klienta, którą ten będzie mógł namacalnie ocenić. Jest mniej istotne jak bardzo rozbudowana została funkcjonalność systemu w każdej następnej wersji w sensie ilościowym ważne jest to, że kolejna wersja musi dodać wartość którą da się zweryfikować w praktycznym użytkowaniu.

Sprint poprzedza faza przygotowawcza, w trakcie której uaktualniana jest lista wymagań użytkownika w postaci „historyjek" (user stories), które raczej są życzeniami użytkownika, jak właściciel wyobraża sobie swój przyszły produkt.

Historyjki muszą być krótkie i treściwe, najtrafniej jest gdy z jednej historyjki wynika jedna cecha systemu. Pomysły gromadzone są na kartonikach, lub jako kolejne wiersze tabeli. Zebrane historyjki są klasyfikowane według ważności. Klasyfikacja jest banalnym podziałem mówiącym o tym czy system:

- mija się z celem bez danej cechy (must be)
- ułatwi znacząco pracę w przypadku posiadania cechy (should be)
- sprawi przyjemność właścicielowi jeśli będzie posiadał daną cechę (nice to have)

Lista oczekujących na spełnienie życzeń razem z ustalonymi priorytetami tworzy rejestr wymagań na produkt (product backlog). Spośród wszystkich określonych w ten sposób wymagań wybiera (określa) się cel główny na dany przebieg. Takie wybranie celu jest konieczne, ponieważ wybór „automatyczny" kolejnego najwyżej zapriorytetowanej cechy może prowadzić do realizacji słabo skorelowanych ze sobą, czy nawet wykluczających się zadań. Poza tym często lista wymagań jest doprecyzowywana przez zespół wykonawczy w dziedzinie technicznej, i takie cechy (wymagania) systemu muszą być realizowane przy okazji innych, są przez nie wymuszane do poprawnego funkcjonowania.

Cel główny wybrany zostaje w porozumieniu z właścicielem produktu i zaakceptowany przez zespół, który logicznie określa powiązania cech, i możliwość wprowadzenia danego wymagania na faktycznym poziomie zaawansowania wytwarzania bieżącej wersji. Cel główny wywiesza się w widocznym miejscu pokoju w którym odbywa się praca zespołu młyna.

Istotą realizacji jednego pełnego przebiegu jest ustalona lista zadań wraz z szacowana pracochłonnością, która nazywa się rejestrem zadań przebiegu (sprint backlog), podobnie jak w przypadku całego rejestru zadań, wystarczy aby taki rejestr był tabelą w arkuszu kalkulacyjnym, gdzie każdy wiersz to kolejna cecha.

Taki rejestr powstaje w ramach oddzielnego spotkania poprzedzającego przebieg aktualnie rozpoczynanej iteracji.. Wygląda to w ten sposób, że zespół zdejmuje ze stosu rejestru wymagań tyle cech, ile będzie w stanie zrealizować w czasie najbliższego przebiegu. Wybiera się zadania o najwyższym priorytecie, zgodnie z wyznaczonym celem głównym dla danego przebiegu.

Stworzenie rejestru zadań przebiegu wymaga ciągu czynności dających precyzyjny wyznacznik co do wyboru cech. Wewnętrzne planowanie polega na:

- rozbiciu życzeń klienta, na elementarne czynności techniczne, konieczne do realizacji analizowanego celu (zawartego w „historyjce" klienta)

- oszacowaniu każdej czynności technicznej na koszt roboto-godziny potrzebnej do zrealizowania funkcjonalności
- przyporządkowaniu odpowiednich czynności do realizacji osobom najbardziej kompetentnym do jej wykonania, co ustala sam zespół, nie kierownik,
- zsumowaniu wszystkich roboto-godzin z wszystkich wybranych czynności i sprawdzeniu czy łączna ich liczba przekracza, czy nie zapełnia godzin jednego pełnego cyklu,
- dopełnieniu lub ujęciu wybranych czynności, aby możliwie jak najdokładniej zmieścić się czasowo w przebiegu jednego cyklu, czyli 30 dni.

Rejestr zadań powstaje więc na wspólnym spotkaniu planistycznym, uczestniczą w nim wszyscy członkowie „młyna" właściciel produktu, doradcy biznesowi i techniczni, przedstawiciele wyższego kierownictwa itd.
Przedstawiciele biznesu i właściciel produktu są konieczni do stworzenia celu przebiegu, a przedstawiciele techniczni są pomocni przy uściślaniu rejestru zadań i oszacowaniu pracochłonności.

2.1. Przebieg SCRUM

Wraz z zakończeniem spotkania planistycznego rozpoczyna się przebieg, który ma dostarczyć gotowej rozszerzonej wersji systemu. W trakcie przebiegu nie wolno modyfikować jego zadań, zmieniać priorytetów zadań, zmieniać składu zespołu, ani wtrącać swoich do pracy zespołu.

Jedynym elementem kontrolującym pracę i postępy zespołu jest wykres spalania (sprint burndown chart). Jednostką spalania są godziny pracy zespołu, na wykresie odnotowywana jest codzienna liczba roboto-godzin jaka jeszcze pozostała do realizacji. Aproksymując punkty „spalenia" godzin określa się czy linia powstałego wykresu przetnie punkt (dzień) zakończenia prac nad przebiegiem. Jeśli przewidywany punkt „wypalenia" (zrealizowania) roboto-godzin wypadnie poza datą zakończenia przebiegu zespół obcina listę zadań przewidzianą do realizacji w danym przebiegu, jeśli natomiast planowany punkt zakończenia wypadnie przed dniem zakończenia przebiegu zadania są dokładane z rejestru zadań produktu, po to aby cały przebieg trwał zgodnie z metodą - 30 dni, i aby modyfikacje rejestru zadań dawały wersję systemu zamkniętą i uzupełnioną o nową część funkcjonalności.

Cały proces przebiegu opiera się zasadniczo, w kwestii organizacyjnej, na praktycznej samokontroli całego zespołu. Związane to jest z realizacją codziennego spotkania (Daily Scrum meeting), trwającego około 15 minut i organizowanego zawsze o tej samej porze, zazwyczaj przed rozpoczęciem prac. Na codziennym spotkaniu wymaga się odpowiedzi od wszystkich czynnych członków zespołu na 3 zasadnicze pytania:

1. Co robiłem wczoraj,
2. Co będę robił dzisiaj
3. Co mi przeszkadza w pracy.

Pytania 1 i 2 mają za zadania pozwolić zorientować się wszystkim członkom zespołu, na jakim etapie zaawansowania znajduje się zespół w danym przebiegu,

dodatkowo 2 jest publiczną deklaracją, możliwą do zweryfikowania na kolejnym spotkaniu dotyczącą jednostkowego zaangażowania w proces wytwórczy i daje efekt mobilizujący do dotrzymania słowa na następnym spotkaniu.
Pytanie 3 jest ewidentnym zadaniem dla mistrza młyna, stanowiącym o jego zasadniczej roli w procesie wytwórczym, która polega na utrzymaniu warunków pozwalających zrealizować projekt do końca, kładąc nacisk przede wszystkim na usunięcie powstających w trakcie prowadzenia prac przeszkód wszelakiej natury, przez techniczną, emocjonalną i biznesową związaną z błędnym interpretowaniem oczekiwań klienta.

Razem z zagadnieniem związanym organizowaniem spotkań jest podział wszystkich członków zespołu na dwie grupy Prosiaków i Kurczaków.
Prosiakami określa się ścisły zespół projektowy, tylko on wypowiada się i odpowiada na 3 powyżej przedstawione pytania, bo na tym etapie realizowane są zadania zmierzające do zaprogramowania elementu systemu. I tylko ten zespół prosiaków ma prawo głosu i wnoszenia zmian dotyczących zagadnień technicznych na bieżącym etapie zaawansowania przebiegu.
Kurczaki, to wszyscy, którzy chcą zapoznać się z zaawansowaniem prac, może to być właściciel produktu, prezes zarządu, osoby nie związane z pracami projektowymi.

Z zakończeniem przebiegu związane są zasadniczo: kolejna wersja produktu i spotkanie przeglądowe (Scrum Review meeting) na którym demonstrowana jest działajaca wersja systemu. Uczestniczą w nim oczywiście właściciel produktu oraz sponsor i przedstawiciele kierownictwa, mogą również realizatorzy innych równoległych projektów.
Różnica w zestawieniu z Daily meetingiem jest taka, że tu wypowiadają się przede wszystkim „kurczaki”, program poddawany jest wszechstronnej ocenie zewnętrznej, ale poza nimi oczywiście mają miejsce również standardowe techniczne procedury kontroli jakości i testy, które również odnotowuje się w rejestrze zadań właściwie zakończonego przebiegu.

3. Metodyka FDD.

W metodyce FDD wyróżnia się zasadniczo role kluczowych wykonawców budowanego systemu, są to:

- menadżer projektu, który odpowiada za całość projektu. Zarządza zakresem prac, harmonogramem działań, oraz zasobami.
- eksperci dziedzinowi, którymi mogą być użytkownicy, klienci, sponsorzy, analitycy biznesowi. Ich zadaniem jest przekazywanie programistom wiedzy na temat wymaganej funkcjonalności systemu.
- główny architekt, odpowiedzialny za ogólny projekt systemu. To funkcja techniczna wymagająca umiejętności technicznych i analitycznych jak również zdolności komunikacyjnych z ekspertami i pozostałymi członkami zespołu projektowego.
- menadżer programistów odpowiedzialny za zarządzanie całym zespołem programistów, jego zadaniem jest rozwiązywanie konfliktów o zasobu między pracującymi współbieżnie zespołami.
- główny programista, odpowiedzialny za implementację przydzielonego mu zbioru cech produktu. Bierze aktywny udział w analizie wymagań oraz

projektowaniu architektury rozwiązania. Kieruje podzespołem złożonym z 3-6 programistów przypisanych do realizacji danego zbioru cech.
- właściciele klas to programiści pracujący pod kierownictwem głównego programisty. Ich zadaniem jest szczegółowe projektowanie, kodowanie, testowanie i dokumentowanie cech produktu.

Future Driven Development (FDD) to metodyka tworzenia oprogramowania, wspomagająca zarządzanie fazami analiz, projektowania i konstrukcji oprogramowania.

Zasadniczym elementem procesu FDD jest zagadnienie cechy (feature) produktu. Cechą jest mały fragment funkcjonalności produktu, mający wartość z punktu widzenia klienta, co oznacza, że cecha ta dostarcza klientowi interesujących dla niego wyników. Cechy są sposobem opisu wymagań funkcjonalnych klienta, specyfikowane są strukturą: <action>the<result><by|of|to|from|for>a(n)<object>

Cechy grupowane są w zbiory cech (features set) w celu schematyzowania ich zależności, które mogą być realizowane w postaci jednego komponentu. Mogą one być realizowane iteracyjnie i mają najistotniejsze, z punktu widzenia biznesowego, znaczenie dla klienta. Struktura specyfikowana jest zgodnie ze schematem: <action>-ing<buisness deliverable><by|of|to|from|for>a(n)<object>

W rozbudowanych systemach FDD wyróżnia podsystemy nazywane obszarami przedmiotowymi (subject area), charakteryzowane są zgodnie z konwencją: <object>management

3.1. Przebieg FDD

Tworzenie modelu ogólnego. W tym etapie tworzony jest ogólny model funkcji biznesowych realizowanego produktu (an overall model). Model ogólny powinien dostarczać wiedzy o wszystkich wymaganych funkcjach bez specjalnego zagłębiania się w szczegóły. Jest on dziełem analityków i programistów oraz przyszłych użytkowników, którzy są najlepszymi ekspertami z dziedziny produktu. Zaangażowanie użytkownika, to kluczowy czynnik wpływający na sukces produktu. Pozwala wspólnie rozumieć problemy przez użytkowników i programistów oraz dzięki temu eliminuje „niejawne” założenia czynione przez jedną lub drugą stronę.

Realizacja tej fazy składa się z:
- stworzenia zespołu projektowego pod kierownictwem Głównego Architekta,
- przeprowadzenia przeglądu dziedziny problemu,
- studiowanie dokumentów z wymaganiami i z dziedziny problemu,
- przygotowanie alternatywnych modeli w oddzielnych małych grupach projektowych,
- wypracowanie wspólnego modelu,
- Zatwierdzenie ogólnego modelu,
- Zdokumentowanie istotnych założeń dotyczących projektu i alternatywnych rozwiązań.

Budowanie listy cech. W oparciu o model opracowany w fazie 1 tworzona jest lista cech produktu (feature list), które zapewnią dostarczenie wymaganej przez

klienta funkcjonalności. W pierwszym kroku wyodrębnia się obszary przedmiotowe odpowiadające głównym fragmentom funkcjonalnym. Następnie każdy obszar przedmiotowy dzielony jest na poszczególne aktywności, czyli zbiory cech. Każdy krok w aktywności identyfikowany jest jako cecha. W tym kroku powstaje hierarchicznie uporządkowana lista cech produktu.

Planowanie implementacji cech. Ta faza ma na celu przygotowanie planu określającego w jakiej kolejności cechy będą implementowane (plan by feature). W tym procesie brane są pod uwagę takie czynniki jak priorytet danej cechy, zależności między cechami, złożoność implementacyjna oraz stopień obciążenia zespołu programistów. Ważnym kryterium jest również podział całego projektu na zewnętrznie możliwe do zaobserwowania etapy, czyli tzw. kroki milowe (milestones) w projekcie. W fazie tej powstaje plan implementacji, określający termin realizacji każdego zbioru cech. Za każdy zbiór cech odpowiedzialny jest wyznaczony główny programista, określany jest również przydział klas programistom, którzy będą je realizowali.

Realizacja tej fazy składa się z:

- sformowania zespołu planującego
- określenia kolejności implementacji
- przypisania zbioru cech głównym programistom
- przypisania klas do programistów

Projektowanie i implementacja. Główne założenie metody FDD to dostarczenie kolejnej „działającej" wersji produktu w krótkich iteracjach. Iteracje te składają się z projektowania szczegółowego (design by ffeature) wybranego zbioru cech produktu. Cechy zakwalifikowane do realizacji w danej iteracji są przydzielane do realizacji dynamicznie tworzonym zespołom programistów. Zadaniem każdego zespołu jest implementacja określonego niewielkiego zbioru cech. Kod danej cechy pisany jest przez członka zespołu, któremu została przypisana klasa biznesowa związana z funkcjonalnością danej cechy. Napisany sprawdzany jest przez pozostałych członków zespołu, jeśli testy wypadną pomyślnie nowy kod zostaje integrowany z resztą produktu, a tym samym system staje się bogatszy o kolejną cechę.

Proces projektowania szczegółowego składa się z następujących zadań:

- sformowania zespołu programistów pod kierunkiem Głównego Programisty.
- opcjonalnego przeglądu dziedziny problemu i studiowania dokumentów referencyjnych
- stworzenia diagramów sekwencji
- uszczegółowienia modelu obiektowego
- zapisania nagłówków klas i metod
- inspekcji projektu

Natomiast w fazie implementacji programiści wykonują takie zadania jak:

- implementacja kodu klas
- przeprowadzenia inspekcji kodu
- testowania jednostkowego
- integracji nowego kodu z produktem

4. Dobre praktyki zwinnych metodyk.

Lekkie metody bazują na zbiorze najlepszych praktyk (Best practises), których stosowanie w połączeniu ze zdefiniowanym procesem tworzenia oprogramowania zapewnia efektywne wykonanie projektu. Kluczowe są następujące praktyki:

Oparcie procesu o wymagania klienta. Cały proces tworzenia oprogramowania koncentruje się wokół wymagań klienta. Architektura obiektowa systemu jest projektowana w oparciu o analizę wymagań użytkownika, które specyfikowane są jako cechy. Planowanie całego projektu bazuje na liście wymaganych cech produktu. Istotą inkrementalnej implementacji produktu jest dostarczanie wersji produktów zawierających wybrane cechy produktu.

Architektura systemu. Podobnie jak w Rational Unified Process, akcentowane jest znaczenie opracowania ogólnego modelu projektowania systemu. Zarysowana w ten sposób architektura stanowi podstawę podejmowania dalszych decyzji technicznych w projekcie. Wczesne opracowanie ogólnego modelu ma szereg zalet:

- Umożliwia lepsze zrozumienie całego systemu, gdyż pozawala na szybszą weryfikację problemów i założeń programistów w kontaktach z ekspertami z dziedziny problemu,
- Jest istotnym czynnikiem w opracowaniu inkrementalnego planu implementacji ponieważ uwidacznia zależności między głównymi składnikami systemu,
- Definiując szerszy kontekst dla projektowania szczegółowego klas implementowanych w danej iteracji powoduje, że są one lepiej przygotowane do użycia w kolejnych iteracjach,
- Ułatwia wprowadzanie modyfikacji do systemu wymagających ze zmian wymagań użytkownika.

Krótkie iteracje. Implementacja systemu powinna się odbywać w sposób iteracyjny. W kolejnych iteracjach jest realizowana kolejna porcja wymaganej funkcjonalności o najwyższym priorytecie. Zapewnia to zdecydowanie lepszą sterowalność projektem. Dostarczając klientowi działającą wersje produktu z jednej strony informujemy klienta o postępie w projekcie, z drugiej strony ewaluacja ze strony klienta upewnia nas ze realizujemy właściwą funkcjonalność zgodnie z jego oczekiwaniami. Istotną zaleta takiego podejścia jest to, że w razie konieczności po każdej iteracji klient może podjąć decyzję o wdrożeniu danej wersji do produkcji.

Inspekcje. Metodyki opierają się na inspekcjach, które są sposobem na zapewnienie wysokiej jakości projektów i kodów. Inspekcje są nie tylko mechanizmem wczesnego wykrywania błędów ale również przepływ wiedzy o produkcie między członkami zespołu oraz zapewniają przestrzeganie standardów kodowania w firmie.

Regularne budowanie produktu pomaga wcześniej wykrywać błędy integracyjne. Praktyka ta jest szczególnie efektywna jeśli możliwe jest automatyczne

wykonywanie testów regresywnych opartych o testy przygotowane przez zespoły realizujące konkretne cechy. Zaletą tego podejścia jest również, że zawsze możemy pokazać klientowi działający produkt zawierający całą zrealizowana dotychczas funkcjonalność. Częstotliwość budowania produktu jest uzależniona od złożoności tego procesu w konkretnym projekcie.

Raportowanie i widoczność wyników. Skuteczne zarządzanie projektem wymaga odpowiedniej informacji o aktualnym stanie jego realizacji. Metodyka zapewnia prostą i nie wymagającą specjalnego wysiłku metodę zbierania danych o stanie zadań. Monitorowanie starań projektu jest oparte o cechy produktu.

5. Podsumowanie.

Powyższy przegląd zwinnych metodyk projektowania, wskazuje na zasadniczą różnicę jaka pojawia się przy traktowaniu wytwarzania systemu informatycznego zgodnie ze standardem „Agile". Istotą całego procesu staje się użytkownik, już nie tylko jego potrzeby wyrażane w bardzo dla niego abstrakcyjny sposób. Warto wobec tego wrócić uwagę na manifest, jaki przyświeca wszystkim zwinnym metodologiom:

- ludzie, ich kontakty, zdolność do rozwiązywania problemów są ważniejsze niż sztywne procedury i narzędzia zarządzania,
- wynikiem projektu jest pracujące oprogramowanie a nie dokumentacja,
- z użytkownikiem się współpracuje a nie negocjuje kontrakt,
- ważniejsza jest umiejętność reagowania na zmieniające się warunki otoczenia niż podążanie za opracowanym na wstępie planem.

Taki zbiór zasadniczych priorytetów dla nowych metodyk projektowania ukazuje podstawową różnicę pomiędzy metodykami opartymi na klasycznej inżynierii oprogramowania, a metodykami zwinnymi. W założeniu tradycyjnych metod projektowania jest wytworzenie oprogramowania, w związku z klientem chodzi o sprzedanie mu gotowego produktu. Wszystkie metodyki, zarówno klasyczne jak i zwinne cały proces wytwórczy mogą dzielić na standardowe 4 etapy zarządzania cyklem życia projektu:

- studium wykonalności projektu (feasibility study)
- planowanie i projektowanie (planning and design)
- wykonanie (production)
- wdrożenie (turnover and startup)

Tradycyjne metodyki skupiając się na gotowym produkcie jako celu całego przedsięwzięcia cały proces zarządzania warunkują etapami studium wykonalności oraz projektowaniu. Efektem jest wdrożenie gotowego produktu.

Podejście zwinne natomiast zarządzanie skupia na planowaniu i wykonaniu, a efekt na bieżąco kontroluje poprzez wdrożenie. Są to podejścia cykliczne i przyrostowe, jednak specyficzna cecha tych metodyk wynika bezpośrednio z podejścia zarządzania a nie zastosowanej metody. Odmianą od tradycyjnych metod jest tu sprzedaż usługi, nie produktu. Kluczowymi czynnikami sukcesu staje się oczekiwanie klienta, ale weryfikowane na bieżąco, świadczy się więc usługę dla odbiorcy systemu. Usługę, która ma na celu wspieranie informatyzacji

przedsiębiorstwa poprzez wytworzenie systemu i wspomaganie w jego zaplanowaniu, aby niedoświadczony w dziedzinie IT odbiorca mógł otrzymać to, co będzie w pełni wykorzystywał, bo będzie to rozumiał, a nie to co sobie wyobrażał nie do końca zdając sobie sprawę z następstw tak wyobrażonej wizji systemu.

Pamiętać należy, że wskazane różnice regulują tylko procesem zarządzania, wykorzystywane metody wytwarzania są standardowe, wykorzystywane adekwatnie do fazy w której znajduje się zaawansowanie cyklu. Dąży się do stworzenia produktu, ale ma on być efektywny, z punktu widzenia odbiorcy efektywność to procent wykorzystania systemu. Chodzi o możliwości wykorzystania wszystkich zaimplementowanych funkcji, trzeba je dobrze zrozumieć, aby dobrze wskazać potrzeby. Praca z użytkownikiem daje już na wstępie takie możliwości właśnie odbiorcy, pokazuje to szereg zalet w całym procesie wytwórczym. Rozluźnia formalizm metodyk, redukuje dokumentacje.

Tu można mówić o wadzie tych metod i jeśli chodzi o duże systemy będzie to wada zasadnicza. System taki działa długo, a bieżące eksploatowanie może produkować błędy których się nie da przewidzieć. Założenie metod zwinnych mówi o oryginalności każdego projektu, skoro jest dedykowany to ciężko mówić schemacie, jeśli jeszcze do tego okrajamy dokumentację to jak wygląda konserwacja w długim przedziale czasu? Jeśli zespół wykonawczy przestanie istnieć, a tylko on zna cały system, to kto mógłby być odpowiedzialny za sprawność ostatecznej wersji? Czy warto będzie „uczyć się" kodu aby go naprawić lub poprawić? Jeśli metodyki są szybkie, minimalizują czas potrzebny na wytworzenie, to czy wobec tego nie szybsze będzie wytworzenie go od nowa? Jednak nowy system to nowe koszty.

Tak więc pamiętać trzeba, że efekty jakie osiąga się dzięki spojrzeniu metodyk zwinnych można również osiągnąć metodykami tradycyjnymi i w zasadzie trafnym wyborem metody wytwarzania na danym etapie cyklu życia systemu.

Można więc pomyśleć nad stwierdzeniem, że metodologie zwinne nadają się tam, gdzie system, jego strukturę - kod, można „ogarnąć" rozumiem maksymalnie kilku osób. Projekty , które mają objąć duży obszar dziedzinowy, mimo dużego ryzyka niepowodzenia trzeba prowadzić metodykami klasycznymi.

Celem moich rozważań jest jednak przedstawienie cech sposobu wytwarzania za pomocą metodyk zwinnych. Wskazać więc należy zasadnicze założenia :

- techniki zakładają ścisłą współpracę z użytkownikiem czy odbiorcą
- właściwie postuluje się włączenie użytkownika w proces projektowania oprogramowania.
- sprzedawana jest usługa tworzenia oprogramowania, a nie gotowy produkt - oprogramowanie, tak więc użytkownik jest tym, kto podejmuje decyzje co i w jakiej kolejności będzie w projekcie wykonywane.
- istotną wagę przywiązuje się do poprawnego szacowania kosztów prac, tak by inwestor/użytkownik mógł świadomie planować swe wydatki na rozwój oprogramowania.
- zobowiązuje się wytwórcę oprogramowania do tego, że każdym swym działaniem powinien udowadniać inwestorowi efektywne wykorzystanie czasu i powierzonych mu środków.
- sprzedając usługę programowania rezygnuje się z zysków z ponownego użycia kodu i modeli analitycznych, bo prace odniesione są do

niepowtarzalnego projektu. Przy takim założeniu rozległa dokumentacja projektowa staje się zbędnym kosztem obciążającym użytkownika i unika się jej.

- uproszczenia dokumentacyjne wymuszają specyficzny sposób porozumiewania się z użytkownikiem. W trakcie tworzenia oprogramowania często (na bieżąco) zadaje się mu pytania i prośby o potwierdzenie dotyczące niewielkiego zakresu funkcjonalności. Stąd wynikają inkrementalny sposób dostarczania oprogramowania oraz krótkie iteracje implementacyjne.
- nie specyfikuje się formalnych punktów kontrolnych w projekcie - nie są one potrzebne, gdyż zakończenie każdej iteracji jest punktem kontrolnym samym w sobie. Z drugiej strony wprowadzenie sformalizowanych punktów kontrolnych nie we wszystkich technikach jest możliwe.
- sekwencyjna realizacja wymagań użytkownika powoduje częste zmiany w architekturze systemu i konieczność przebudowy kodu.
- w nowych metodykach zadanie przebudowy kodu postrzega się nie jako wyjątek, lecz jako regułę.