Konferencja naukowa KNWS'04
„Informatyka - sztuka czy rzemiosło"
23-25 czerwca 2004, Czocha

CYKL ŻYCIA PROJEKTU INFORMATYCZNEGO

Tomasz Gratkowski

Instytut Informatyki i Elektroniki, Uniwersytet Zielonogórski
65-246 Zielona Góra, ul. Podgórna 50

e-mail: t.gratkowski@iie.uz.zgora.pl

STRESZCZENIE

W artykule przedstawiono najbardziej znane modele oraz metodologie cyklu życia projektu informatycznego. Artykuł ma charakter przeglądowy. Przegląd modeli zostanie wykorzystany przy wyborze i adaptacji wybranego modelu, dopasowanego do metodologii wytwarzania aplikacji komponentowych.

1. WPROWADZENIE

Wytwarzanie oprogramowania stało się bardzo złożonym procesem, a wzrost funkcjonalności oferowanej przez programy wymusza na twórcach narzędzi programistycznych opracowywania bardziej zaawansowanych środowisk programistycznych. Stosowane obecnie języki obiektowe stały się mało wydajne, przy dużych projektach informatycznych. Dlatego prowadzone są badania w kierunku udoskonalania technik komponentowych, które mają ulepszyć proces implementacji systemów informatycznych [4]. Podstawowymi zaletami stosowania komponentów są:

- zwiększenie wydajność wytwarzania aplikacji,
- zredukowanie złożoność,
- umożliwienie wielokrotnego użycie kodu,
- zmiana sposobu implementowania, raczej montaż (produkcja przemysłowa) niż wytwarzanie (inżynieria),
- redukcja wymaganych umiejętności wymagane doświadczenie obejmujące pojedynczą domenę problemu,
- poprawa możliwości oprogramowania.

Poprawa samego procesu implementacji aplikacji, nie jest wystarczającym krokiem. Dla technologii komonentowych należy opracować metodologię tworzenia systemu informatycznego (ang. *software project management*), dopasowaną do wymagań stawianych przez proces implementacji komponentów. Przy opracowaniu metodologii wykorzystane zostaną modele cyklu życia projektu informatycznego dedykowane głównie dla technologii

obiektowych. W projektowanej metodologii położony będzie nacisk na automatyczne przejście pomiędzy poszczególnymi procesami.

2. MODELE CYKLU ŻYCIA PROJEKTU INFORMATYCZNEGO

Proces wytwarzania projektu informatycznego, w którym można określić moment początkowy i końcowy, nazywany jest cyklem życia projektu. Metodologię postępowania w procesie wytwórczym określają modele cyklu życia [4]. Niezależnie od modeli można wyodrębnić następujące procesy, które mogą być realizowane w ramach projektu [1].

Specyfikacja. Proces ten służy do określenia wymagań stawianych wytwarzanemu oprogramowaniu. W ramach procesu przeprowadzane są analiza problemu, wyodrębnienie potrzeb użytkownika, definiowanie systemu, zarządzanie zakresem [10]. Końcowym efektem specyfikacji są modele biznesowe przedsiębiorstwa, wymagania zapisane w formie dokumentacji oraz modeli. Specyfikacja wymagań jest etapem bardzo istotnym, lecz często bagatelizowany.

Inicjowanie. Przed rozpoczęciem projektu kadra kierownicza podejmuje decyzje dotyczące samego projektu, w celu osiągnięcia sukcesu w projekcie. W ramach procesu inicjacji dokonuje się planowania przebiegu projektu, zatwierdzenia podjęcia rozwiązania określonego w wymaganiach celu, określenia budżetu, ustalenia dostępności wymaganych zasobów sprzętowo-programowych oraz sprecyzowania zakresu projektu. Końcowym efektem inicjowania jest umowa pomiędzy kontrahentami oraz dokumentacja projektowa przydatna przy analizie finansowej przedsięwzięcia.

Projektowanie. Na bazie danych wejściowych, jakie uzyskiwane są z procesu specyfikowania dokonywane jest projektowanie systemu informatycznego na różnych poziomach abstrakcji. W procesie projektowania wykorzystywane są powszechnie różne techniki modelowania. Końcowym efektem projektowania są modele o wymaganej szczegółowości umożliwiającej implementację poszczególnych modułów systemu.

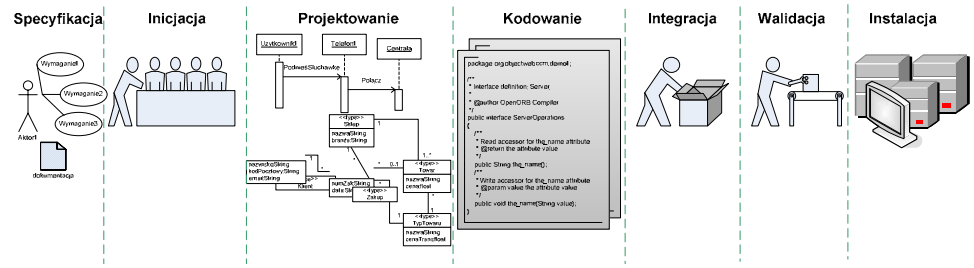
Kodowanie. Proces, w którym zespoły lub pojedynczy programiści dokonują implementacji oraz wstępnego testowania modułu kodu źródłowego. Sposób testowania powinien być opracowany w procesie projektowania. Daje to dodatkowy mechanizm sprawdzający czy programiści poprawnie zrozumieli i zaimplementowali moduł. W procesie kodowania wykorzystywane są powszechnie zintegrowane narzędzia programistyczne umożliwiające pisanie kodu źródłowego w wybranym języku programowania. Końcowym efektem kodowania jest kod źródłowy modułu lub binarny moduł.

Integracja. Zakodowane moduły łączy się w większe struktury (podsystemy), które tworzą wytwarzany system. W zależności od rodzaju modułów wejściowych (kod źródłowy lub kod binarny), dokonywana jest kompilacja lub tylko łączenie. Opracowywany jest również

mechanizm instalacji systemu. W procesie integracji wykorzystywane są powszechnie zintegrowane narzędzia programistyczne dedykowane dla stosowanej technologii, umożliwiające integrację modułów. W ramach integracji dokonywane jest również testowanie, czy poszczególne moduły poprawnie ze sobą współpracują oraz czy wspólnie realizują stawiane modułowi cele. Końcowym efektem integracji jest system przygotowany do instalacji u klienta.

Walidacja. W procesie walidacji dokonywane jest sprawdzenie czy otrzymany system odpowiada zdefiniowanym w procesie specyfikacji wymaganiom. Procedury testowe stosowane w procesie walidacji, opracowywane są we wstępnym procesie specyfikacji. W procesie walidacji wykorzystywany jest utworzony system oraz zaprojektowane procedury testowe. Końcowym efektem walidacji jest przetestowany system przygotowany do instalacji u klienta.

Instalacja. Proces instalacji ma na celu wdrożenie wytworzonego systemu w środowisku docelowym u klienta. W ramach instalacji dokonywane jest również przeniesienie lub uzupełnienie wszystkich potrzebnych do pracy systemu danych. Klient zobowiązany jest do przeprowadzenia testów akceptacyjnych, aby system mógł być formalnie wprowadzony do eksploatacji. W procesie instalacji wykorzystywany jest przygotowany mechanizm instalacji oraz docelowe środowisko programowo-sprzętowe u klienta. Końcowym efektem instalacji jest działający u klienta system oraz dokumentacja odbioru.



Rys. 1. Procesy wykorzystywane przy tworzeniu systemu informatycznego

Na bazie przedstawionych procesów utworzono szereg modeli cyklu życia, w odpowiedni sposób łącząc i dzieląc wybrane procesy. Poniżej zaprezentowano wybrane modele.

2.1. Tradycyjne modele cyklu życia projektu informatycznego

Przykładami tradycyjnych modeli ją: buduj i poprawiaj, kaskadowy, V, spiralny, przyrostowy.

Model buduj i poprawiaj. W modelu tym można wyodrębnić dwa etapy: 1) kodowanie, kompilacja i wykonanie, 2) wykrywanie błędów, naprawa błędów. Etapy są tak długo powtarzane, aż do osiągnięcia poprawności działania oraz spełnienia oczekiwań klienta. Model ten nie sprawdza się przy dużych projektach. Zaletą jest szybkość wykonania zadania.

Model kaskadowy. Jednym z najbardziej znanych modeli jest model kaskadowy. Wyodrębniono w nim następujące etapy: 1) definiowanie wymagań, 2) projektowanie systemu, 3) implementacja i testowanie jednostek, 4) integracja i testowanie systemu 5) działanie i pielęgnacja. Etapy wykonywane są cyklicznie i zanim rozpoczęty zostanie kolejny etap wcześniejszy musi zostać zakończony.

Model V. Model ten jest odmianą modelu kaskadowego. Dodano kolejne etapy testowania i walidacji po każdym ukończonym etapie.

Model spiralny. Model spiralny [2] został opracowany w celu wyeliminowania wad modelu kaskadowego. Model stał się bardziej elastyczny wskazując, że nie najważniejsze jest trzymanie się ścisłego planu, skupiając uwagę kadry zarządzającej na konieczność zarządzania ryzykiem oraz pozytywnego zrealizowania celu. W modelu spiralnym można wyodrębnić następujące etapy: 1) wyznaczenie celów, alternatyw i ograniczeń, 2) ocena alternatyw, identyfikacja i oszacowanie ryzyka, 3) wyznaczenie i weryfikacja następnego przybliżenia produktu, 4) planowanie kolejnej fazy. Każdy z wymienionych etapów jest wielokrotnie powtarzany w cyklu zegarowym, ale w zależności od numeru obrotu wykonywane są dodatkowe czynności w każdym z czterech wymienionych etapów.

Model przyrostowy. W modelu przyrostowym, który bazuje na modelu kaskadowym, skrócono cykl produkcyjny dzięki dostarczeniu klientowi niekompletnego systemu, stopniowo uzupełniając go o pozostałe moduły. Dzięki takiemu podejściu klient we wcześniejszym etapie wytwarzania systemu ma możliwość wpływania na końcowy wygląd systemu. Wadą modelu jest trudne zarządzanie projektem.

2.2. Nowoczesne modele cyklu życia projektu informatycznego

Nowoczesne modele są kompromisem pomiędzy przesadnym skupianiu uwagi nad poszczególnymi procesami użytymi w modelu a sytuacjami gdzie nie stosowano żadnego nadzoru nad przebiegiem projektu. Przykładami nowoczesnych modeli są: Adaptive Concept Development, Crystal, Extreme Programming, Rational Unified Process.

Adaptive Software Development (ASD). Highsmith w [5] przedstawił adaptacyjną naturę nowej metodologii cyklu życia projektu informatycznego. Zaproponowana metodologia jest alternatywą dla tradycyjnych modeli i bardziej odpowiada potrzebom nowoczesnych procesów biznesowych. Zamieniono statyczny plan działań na trzy dynamiczne etapy: rozważanie, współpracę oraz naukę. Rozważanie składa się z następujących czynności:

- inicjalizacja projektu – służy określeniu celu projektu
- określenie ram czasu całego projektu
- określenie ilości oraz czasu trwania iteracji w projekcie
- rozwój tematu oraz celu każdej iteracji
- wyznaczenie właściwości dla każdej iteracji przez klienta i wytwórcę

Współpraca jest etapem, w którym zespół programistów dostarcza gotowy system. Menadżer projektu ma za zadanie wspomagać zespół programistów oraz tworzyć dokumentację projektu. Ostatni etap nauka, skoncentrowany jest głównie na czterech dziedzinach: jakości z punku widzenia klienta, jakości z technicznego punktu widzenia, funkcjonowania i optymalizacji zadań zespołu programistycznego oraz statusu projektu.
Wykorzystując z ASD aplikacja jest bliższa wymaganiom klienta. Wymagane zmiany w programie są łatwiejsze do przystosowania. W procesie wytwórczym wykorzystywane są metody analizujące jakość. Dzięki zastosowaniu analizy ryzyka, metodologia zwiększa pewność prawidłowego zakończenia projektu informatycznego już we wstępnej fazie projektu. Metodologia ASD stała się pierwowzorem dla całej gruby metod nazywanych Agile Methodology [6].

Crystal. Przykładem metodologii opartej o Alige Methodology jest metodologia Crystal [3]. Metodologia Crytal zawiera w sobie więcej niż jedną metodologię, ponieważ główną myślą jest dobieranie odpowiedniej metodologii do wymagań implementowanego projektu. Przed wyborem metody dokonywany jest podział projektów na dwa typy: ze względu na liczbę programistów oraz ze względu na ryzyko inwestycji. W Crystal zastosowano kolorowe pasma. Kolory „przezroczysty", „żółty", „pomarańczowy", „czerwony", „bordowy", „niebieski" i „fioletowy" wskazują po kolei, iż realizowany projekt jest większy i wymaga zastosowania bardziej rozbudowanych metod. Każda użyta na wskazanym poziomie złożoności projektu metodologia jest minimalna dla poprawnego zrealizowania celów projektu. Całość oparta jest na czterech zasadach: używaj dużej metodologii dla dużych zespołów, zastosuj bardziej złożonej metodologii dla projektów o większym ryzyku, interakcja i bezpośrednia rozmowa jest bardziej efektywna, wielkość i złożoność jest kosztowna.

Extreme Programming (XP). Metodologia XP [7] jest jedna z najbardziej znanych i używanych, ponieważ jest bardzo dokładna oraz prosta w zrozumieniu. Jest przeznaczona głownie dla małych zespołów. Składa się z trzytygodniowych cykli, częstych uaktualnień, wspiera zarówno priorytety biznesowe jak i techniczne oraz wyznacza linię postępowania. XP składa się z dwunastu praktyk wykorzystywanych podczas projektu: planowanie, mała wersja, metafora, prosty projekt, ponowne rozłożenie na czynniki, testowanie, programowanie w parach, wspólne posiadanie kodu źródłowego, ciągła integracja kodu źródłowego, czterdziesto godzinny tydzień, zaangażowanie klienta, standardy pisania kodu źródłowego. W metodologii XP położono duży nacisk na testowanie, programiści zobowiązani są do napisania testów zaraz po napisaniu fragmentu aplikacji. Zaletą stosowania metodologii XP jest efektywne wykorzystanie zespołów programistycznych.

Rational Unified Process (RUP). Metodologia RUP [9] została opracowana przez firmę Rational Software na bazie metodologii Unified Software Development Process [8]. W RUP

przedstawiono sześć głównych praktyk: rozwijaj oprogramowanie iteracyjnie, zarządzaj wymaganiami, wykorzystuj architekturę komponentową, wizualnie modeluj oprogramowanie, weryfikuj jakość oprogramowania, kontroluj zmiany oprogramowania. W celu zrealizowania wymienionych praktyk opracowano dwa wymiary opisujące etapy postępowania: poziomy – reprezentujący czas oraz dynamiczne aspekty procesów, wyrażone poprzez cykle, fazy, powtórzenia oraz kroki milowe; poziomy – reprezentujący statyczne aspekty procesów, wyrażone poprzez działania, artefakty, pracowników oraz przepływ prac.

3. Wnioski

Artykuł miał na celu przegląd modeli oraz metodologii cykl życia projektu informatycznego. W artykule nie przedstawiono wszystkich istniejących modeli i metodologii, które będą brane pod uwagę przy wyborze i adaptacji metodologii na potrzeby implementacji systemu zrealizowanego w technologii komponentowej. Dalszym kierunkiem prac będzie określenie kryteriów wyboru odpowiedniego metodologii oraz analiza dodatkowych czynności niezbędnych przy procesie wytwórczym systemu informatycznego.

LITERATURA

[1] *The Department of Justice Systems Development Life Cycle Guidance Document*. United States Department of Justice, January, 2003, http://www.usdoj.gov/jmd/irm/lifecycle/table.htmc

[2] B.W. Boehm: *A spirtal Model of Software Development and Enhancement*, Computer, May 1988

[3] M. Fowler: *The New Methodology*, ThoughtWorks, November 2000 http://www.martinfowler.com/articles/newMethodology.html

[4] pod redakcją J. Górski: *Inżynieria oprogramowania w projekcie informatycznym*, MIKOM, 2000

[5] J. Highsmith: *Adaptive Software Development*, New York: Dorset House Publishing, 1999

[6] J. Highsmith: *Agile Project Management : Creating Innovative Products*, Addison-Wesley, 2004

[7] J. Highsmith: *Extreme Programming*. Cutter. Retrieved April 8, 2000, http://www.cutter.com/ead/ead0002.html

[8] I. Jacobson, G. Booch, J. Rambougt: *The Unified Software Development Process*, Addison-Wesley, 1999

[9] P. Kruchten: *The Rational Unified Process An Introduction*, Second Edition, Addison-Wesley, 2000

[10] D. Leffingwell, D. Widrig: *Zarządzanie wymaganiami*, Wydawnictwa Naukowo-Techniczne, 2003