Rational Unified Process

Dokładny opis metodyki i procesu produkcji oprogramowania

Rational Unified Process (RUP) .

RUP jest iteracyjnym procesem rozwoju oprogramowania. Definiuje szkielet postępowania, który należy dostosować do uwarunkowań konkretnego projektu programistycznego. Powstał na bazie analizy najczęstszych przyczyn niepowodzeń istniejących procesów wytwarzania oprogramowania. Oparty jest o pewne podstawowe zasady oraz fazy wytwarzania oprogramowania.

Proces RUP nie jest pojedynczym, ściśle określonym procesem, ale raczej szablonem procesu. Został on zaprojektowany w celu przystosowania do charakteru konkretnej organizacji (przedsiębiorstwa), konkretnego zespołu projektowego lub nawet charakteru konkretnego projektu. Z szablonu RUP można wybrać elementy w zależności od konkretnych potrzeb.

Zasady RUP:

- iteracyjne i przyrostowe tworzenie oprogramowania; sterowane ryzykiem i priorytetami, ułatwiające integrację całości kodu i dostosowanie do zmieniających się wymagań
- zarządzanie wymaganiami; we współpracy z klientem i w oparciu o przypadki użycia,
- stosowanie architektury opartej na komponentach,
- graficzne modelowanie oprogramowania; różne perspektywy spojrzenia na system, użycie UML,
- kontrola i weryfikacja jakości oprogramowania przez cały czas procesu wytwarzania,
- zarządzanie zmianami w oprogramowaniu.

Metodyka_RUP

- Jest oparta na doświadczeniach największych firm w branży

Informatycznej.

- Opiera się na zestawie praktyk:

- Iteracyjne wytwarzanie oprogramowania,
- Zarządzanie wymaganiami,
- Architektura bazująca na komponentach,
- Wizualne modelowanie oprogramowania,
- Kontrola i weryfikacja jakości oprogramowania,
- Zarządzanie zmianami w oprogramowaniu.

Iteracyjne wytwarzanie oprogramowania

Wymagania podczas procesu wytwarzania oprogramowania ulegają częstym zmianom, z powodu ograniczeń architektury, zmiany potrzeb użytkownika lub lepszego zrozumienia problemu. Wytwarzanie oprogramowania w kolejnych iteracjach, pozwala skupić się w pierwszej kolejności na obszarach najbardziej ryzykownych (np. najmniej rozpoznanych). W idealnym przypadku każda iteracja kończy się stworzeniem wykonywalnego artefaktu – pomaga to zredukować ryzyko w projekcie, otrzymujemy szybciej opinie od odbiorców oprogramowania a programistom pozwala skupić się na węższej dziedzinie.

Iteracyjne wytwarzanie oprogramowania

RUP używa podejścia iteracyjnego i przyrostowego z następujących powodów:

- Integracja oprogramowania robiona krok po kroku podczas wytwarzania oprogramowania, ograniczając go do mniejszej liczby elementów ,
- Integracja jest prostsza i mniej kosztowna ,
- Składowe oprogramowania są projektowane oddzielnie i łatwiej użyć je ponownie ,
- Łatwiej wykrywać zmiany wymagań i łatwiej nimi zarządzać ,
- Zagrożenia identyfikowane i atakowane są wcześnie ponieważ każda iteracja pozwala wykryć kolejne zagrożenia ,
- W iteracjach ulepszana jest architektura oprogramowania .

Projekt wykorzystujący model iteracyjny będzie posiadał jeden główny plan faz, a zarazem wiele planów iteracji.

Zarządzanie wymaganiami

Zarządzanie wymaganiami jest skupione na zaspokojeniu oczekiwań użytkowników końcowych systemu poprzez identyfikację i specyfikację ich potrzeb oraz **wykrywanie zmiany** tych wymagań.

Zalety zarządzania wymaganiami:

- Poprawnie zidentyfikowane wymagania tworzą prawidłowy produkt, potrzeby użytkownika są zaspokojone.
- Tworzy istotną dla użytkowników funkcjonalność, redukując późniejsze koszty dobudowywania zapomnianej (niezidentyfikowanej podczas tworzenia) funkcjonalności.

Zarządzanie wymaganiami zawiera:

Analiza problemu – uzgodnienie problemu i stworzenie miar, które dowiodą jego istotności dla klienta.

Zrozumienie potrzeb udziałowców (stakeholders) – konsultacja problemu i jego wartości z głównymi udziałowcami (stakeholders) i rozpoznanie w jaki sposób koncepcja rozwiązania zaspokaja ich potrzeby.

Definicja systemu – tworzenie projektu funkcjonalności na podstawie potrzeb użytkowników, identyfikacja przypadków użycia – które prezentują ogólne wymagania (high-level requirements) i użyteczność modelu systemu.

Zarządzanie wymaganiami zawiera:

Zarządzanie zakresem systemu (Scope Management) – modyfikowanie zakresu prac nad systemem bazując na analizie wymagań, wybór kolejności realizacji (atakowania) przypadków użycia.

Zawężanie definicji systemu – uszczegóławianie scenariuszy przypadków użycia razem z użytkownikami systemu w celu stworzenia dokładnej specyfikacji wymagań (ang. Software Requirements Specification – SRS), która może służyć (i na ogół służy) jako umowa pomiędzy wykonawcą systemu a klientem. Na podstawie dokumentu SRS tworzony jest projekt systemu oraz scenariusze testów.

Zarządzanie zmianami wymagań – zarządzanie zmianami wymagań lub nowozidentyfikowanymi wymaganiami w czasie trwania projektu.

Architektura bazująca na komponentach

Użycie architektury bazującej na komponentach pozwala na stworzenie systemu, który jest łatwo rozszerzalny, intuicyjnie zrozumiały i wspomaga reużywalność. Komponentem nazywamy zbiór powiązanych obiektów (w sensie programowania obiektowego).

Architektura oprogramowania zyskuje na znaczeniu w miarę jak systemy informatyczne stają się coraz większe i bardziej złożone. RUP skupia się na stworzeniu prostej architektury w początkowych iteracjach. Staje się ona prototypem dla pierwszej fazy implementacji (development). Ewoluuje ona w każdej iteracji zbliżając się do architektury finalnej. RUP zakłada reguły i ograniczenia projektowe w celu uchwycenia reguł architektury. Poprzez iteracyjne wytwarzanie oprogramowania zyskujemy możliwość stopniowej identyfikacji komponentów, które mogą być w dalszej części: zakupione, zbudowane, lub użyte ponownie. Komponenty są często budowane na bazie istniejących technologii typu CORBA, COM, JEE.

Wizualne modelowanie oprogramowania

Abstrakcja projektowania od kodu i przedstawienie koncepcji za pomocą bloków graficznych może być efektywnym sposobem aby pokazać perspektywę rozwiązania. Używając takiej reprezentacji, techniczni członkowie zespołu mają możliwość wybrania najlepszego sposobu implementacji zbioru powiązanej funkcjonalności. Reprezentacja graficzna jest także produktem pośrednim pomiędzy analizą procesu biznesowego, a implementacją. Model w tym kontekście jest formą wizualizacji oraz uproszczeniem bardziej skomplikowanego projektu. RUP specyfikuje wymagane modele i opisuje dlaczego są wymagane.

Kontrola i weryfikacja jakości oprogramowania

Ocena jakości jest najczęstszym słabym punktem projektów programistycznych ponieważ jest często planowana po fakcie budowy systemu i czasami obsługiwana przez inny zespół. RUP pomaga w planowaniu kontroli jakości i jej ocenie poprzez wbudowanie jej w cały proces i zaangażowanie w nią wszystkich członków zespołu. Nie ma pracowników przypisanych tylko do jakości – RUP zakłada, że każdy członek zespołu jest odpowiedzialny za jakość w ciągu całego procesu. Proces koncentruje się na spełnieniu wymaganego poziomu jakości i zapewnia mechanizmy (workflows) do pomiaru tego poziomu.

Zarządzanie zmianami w oprogramowaniu

We wszystkich projektach programistycznych pojawiają się z czasem zmiany i są one nieuniknione. RUP definiuje metody śledzenia, ewidencji i kontroli zmian. Zdefiniowane są także tzw. *secure workspaces* (bezpieczne przestrzenie robocze), które pozwalają na zagwarantowanie, że zmiany w innych systemach nie wpłyną na system tworzony. Koncepcja ta jest ściśle powiązana z tworzeniem architektury zorientowanej komponentowo.

Cykl tworzenia oprogramowania

Procesy, jakie są realizowane w czasie budowy oprogramowania, są zazwyczaj cykliczne. Systemy, w zależności od zastosowanej metody, charakteryzują się spiralnym, kaskadowym (wodospadowym) lub strukturalnym cyklem wytwarzania. Praktycznie każda metodyka preferuje swój cykl wytwórczy.

Cykl wytwórczy RUP jest charakterystyczny, gdyż pokazuje procesy w dwóch płaszczyznach. W pionie przedstawione są statyczne aspekty wytwarzania oprogramowania, takie jak czynności, role, przepływy oraz artefakty jakie im towarzyszą. W poziomie natomiast przedstawione zostały dynamiczne aspekty wytwarzania oprogramowania takie, jak fazy oraz iteracje.

Podstawowe wiadomości o Rational Unified Process (RUP).

Hierarchiczna struktura RUP:

- Cykle (ang. Cycles)
- Fazy (ang. Phases)
- Iteracje (ang. Iterations)
- Prace (ang. Workflows)
- Czynności (ang. Activities)

Cykle i fazy

Wytwarzanie oprogramowania następuje w cyklach:

- Cykl początkowy
- Cykle ewolucyjne

Fazy życia oprogramowania :

- Rozpoczęcie (Inception)
- Opracowanie (Elaboration)
- Konstruowanie (Construction)
- Przekazanie (Transition)

Rozpoczęcie (Inception)

Faza ta koncentruje się na wygenerowaniu opisu z punktu widzenia potrzeb użytkownika (business case), oceny czy jest to wykonalne i opłacalne. Zdefiniowane są tu przypadki użycia, zakres działania systemu, a także nowe, ryzykowne bądź trudne elementy systemy, mogące mieć wpływ na końcowy sukces.

W fazie tej powstaje koncepcja architektury (na najwyższym poziomie), dla oceny stopnia komplikacji. Należy też ocenić wykonalność projektu oraz oszacować nakłady czasu, siły roboczej, finansowe.

Podstawową pracą są tu prace nad wymaganiami, jednakże by rzetelnie ocenić architekturę konieczne są pewne prace analityczne, projektowe, a nawet implementacyjne. Podstawowym pytaniem jest jednak wykonalność przedsięwzięcia.

Opracowanie (Elaboration)

Podstawowym zadaniem tej fazy jest zrozumienie (i formalne zapisanie) jak wymagania przekładają się na podstawową architekturę systemu i jak mogą przebiegać dalsze prace. Wszystkie przypadki użycia powinny być dokładnie opisane. Zidentyfikowane powinny być elementy obarczone ryzykiem (nieznane). Powinny one być opracowane jako pierwsze w następnej fazie (construction). Rozważeniu powinny podlegać też zagadnienia związane z niezawodnością oraz szybkością działania. Odpowiednio wysokie wymagania mogą mieć istotny wpływ na architekturę systemu.

Ta faza obejmuje zaprojektowana, zaimplementowana i przetestowana, przewidywana architekturę systemu i określone elementy o najwyższym ryzyku (najwyższej trudności).

Konstruowanie (Construction)

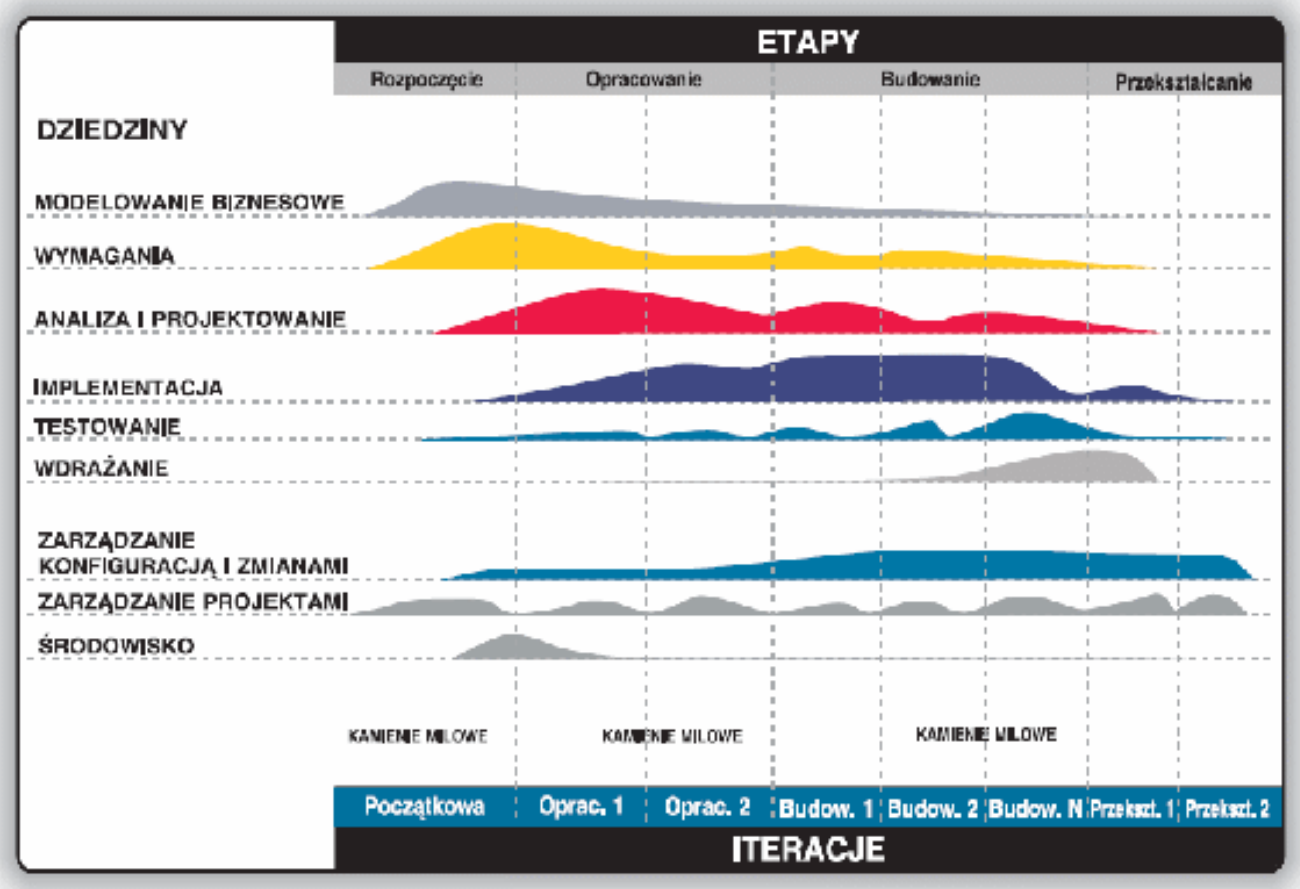
Produktem końcowym tej fazy jest w pełni funkcjonalna wersja beta systemu. Może ona zawierać pewne błędy, wymagać pewnych niedużych poprawek. Poprawki te nie powinny zmieniać podstawowej funkcjonalności.

Powodzenie tej fazy zależy głównie od rozwiązania problemów wykrytych we wcześniejszych fazach. Faza ta zawiera prace głownie projektowe i implementacyjne, chociaż prace nad wymaganiami i analityką są dopuszczalne.

Przekazanie (Transition)

Faza ta zaczyna się instalacji beta wersji systemu, uzyskaniu opinii użytkownika i wprowadzeniu żądanych zmian i usprawnień. Mogą być tu prowadzone prace projektowe i implementacyjne. Faza ta kończy się oddaniem wersji produkcyjnej systemu.

Cykl tworzenia



Rysunek 1. Cykl tworzenia oprogramowania w RUP (załącznik[1]).

Liczby iteracji dla projektu o średnim stopniu złożoności:

- **Rozpoczęcie -** Jedna iteracja, składająca się głównie z prac nad wymaganiami.
- **Opracowanie -** Dwie iteracje, pierwsza identyfikująca przypadki użycia i zarys architektury, druga uszczegółowiająca przyjętą architekturę.
- **Konstruowanie -** Trzy, cztery lub pięć iteracji w zależności od wykrytych zagrożeń i złożoności systemu. Każda z iteracji zawiera prace projektowe, implementacyjne i testowanie. Mogą też wchodzić prace analityczne (wprowadzanie zmian).
- **Przekazanie -** Jedna lub dwie iteracje w zależności od sukcesu wersji beta.

Dyscypliny

RUP bazuje na zestawie klocków lub elementach treści, opisujących to, co ma być produkowane, niezbędne umiejętności, wymagane wyjaśnienia krok po kroku, opisujące konkretne cele rozwoju, które mają zostać osiągnięte. Podstawowe elementy, treści lub elementy, są następujące:

- Role (kto) - Rola definiuje zbiór powiązanych umiejętności, kompetencji i odpowiedzialności.
- Produkt (co) - Wyrób reprezentuje coś wynikające z zadań, w tym wszystkie dokumenty i modele produkowane podczas pracy przez cały proces.
- Zadania (jak) - Zadanie opisuje jednostkę pracy przypisaną do roli, jaką przewiduje znaczący wynik.

Dyscypliny

W każdej iteracji zadania podzielone są na dziewięć dyscyplin:

a. sześć "inżynieryjnych" (Modelowanie biznesowe, wymagania, analiz i projektowania, implementacja, testowania, wdrażania)

b. trzech dyscyplinach wspomagających (konfiguracja i zarządzanie zmianami, zarządzanie projektami, Środowisko).

Dyscyplina modelowanie biznesowe

Modelowanie biznesowe wyjaśnia, jak opisać wizję organizacji, w którym system będzie wdrażany i jak następnie użyć tej wizji jako podstawę do przedstawienia procesu, ról i obowiązków. Organizacja stają się coraz bardziej zależne od IT systemów, co jest konieczne, aby inżynierowie systemu informacji wiedzieli w jaki sposób aplikacje są dopasowane do rozwoju organizacji. Firmy inwestują w IT, gdy zrozumieją przewagę konkurencyjną i wartość dodaną tej technologii. Celem modelowania biznesowego jest najpierw ustalenie lepszego zrozumienia i kanału komunikacji między inżynierią biznesu i inżynierią oprogramowania . Zrozumienie biznesu oznacza, że inżynierowie oprogramowania muszą zrozumieć strukturę i dynamikę organizacji docelowej (klienta), obecne problemy w organizacji i ewentualną poprawę. Muszą one także zapewnić wspólne zrozumienie docelowej organizacji pomiędzy klientami, użytkownikami i programistami.

Dyscyplina Wymagania

Celami dyscypliny wymagań są:

- Osiągnięcie konsensusu wśród uczestników projektu: „co i dlaczego powinien robić projektowany system”.
- Uzyskanie lepszego zrozumienia wymagań dla systemu przez członków zespołu projektowego.
- Określenie granic systemu.
- Ustanowienie bazy dla planowania iteracji przy pracach projektowych.
- Ustanowienia bazy dla szacowania kosztów i czasu niezbędnego dla realizacji projektu.
- Zdefiniowanie interfejsu użytkownika w oparciu o cele i potrzeby użytkowników.

Dyscyplina analiza i projekt

Zamiana wymagań w specyfikację implementacji systemu :

- Ustanowienie stabilnej architektury
- Przystosowanie projektu do środowiska implementacji
- Uwzględnienie własności systemu

Systemy są realizowane poprzez wdrożenie elementów. Proces opisano w sposób ponownego wykorzystania istniejących elementów, lub wprowadzenia nowych elementów odpowiedzialnych dobrze zdefiniowanych, tworząc system łatwiejszy do utrzymania i zwiększenia możliwości ponownego wykorzystania.

Dyscyplina implementacja

Celami dyscypliny implementacji są:

- Definicja organizacji kodu systemu, z uwzględnieniem podziału na warstwy,
- Implementacja projektu (programowanie)
- Przygotowanie i testowanie zaprojektowanych komponentów jako jednostek
- Integracja komponentów (być może wytworzonych przez różne grupy) w system docelowy

Dyscyplina testów

Celami dyscypliny testów są:

- Sprawdzenie, interakcji pomiędzy obiektami.
- Sprawdzenie właściwej integrację wszystkich elementów oprogramowania.
- Sprawdzenie, czy wszystkie wymagania zostały wykonane prawidłowo.
- Zidentyfikować i zapewnić, że wady są skierowane przed oddelegowaniem oprogramowania.
- Upewnij się, że wszystkie wady są stałe, testowane i zamknięte.

Rational Unified Process proponuje podejście iteracyjne, co oznacza, że "badanie ma miejsce w całym projekcie„. Pozwala to na wykrycie wady tak wcześnie, jak to możliwe, co radykalnie zmniejsza koszty ustalające wady. Badania są wykonywane w jakości czterech wymiarach: niezawodność, funkcjonalność, wydajność aplikacji i wydajności systemu.

Dyscyplina wdrożenie

Celem wdrożenia jest z powodzeniem produkować wersje produktu, jak i dostarczać oprogramowanie do użytkowników końcowych. Obejmuje ono szeroki zakres działań, w tym produkcji zewnętrznych wydań oprogramowania, opakowania aplikacji i biznesu, dystrybucji oprogramowania, instalacji oprogramowania oraz zapewnienie pomocy i wsparcia dla użytkowników. Chociaż działania wokół systemu są najczęściej przejściowe, wiele działań musi być uwzględnionych we wcześniejszych fazach przygotowania do wdrożenia na końcu fazy budowy. Wdrożenie i Środowisko przepływów pracy z Rational Unified Process zawiera mniej szczegółów niż inne procesy.

Dyscypliny Środowiska

Działalności tej dyscypliny są:

- Dostosowanie obróbki materiałów dla poszczególnych zespołów projektowych
- Identyfikacji i oceny narzędzi
- Instalowanie i konfigurowanie narzędzi dla zespołu projektowego
- Wspieranie narzędzi i procesów w całym projekcie

Dyscyplina konfiguracja i zarządzania zmianami

Dyscyplina zarządzania zmianą w RUP dotyka trzech obszarów:

- Zarządzanie konfiguracją: Jest odpowiedzialne za systematyczne kształtowanie produktów. Artefakty takie jak dokumenty i modele muszą być pod kontrolą wersji i te zmiany muszą być widoczne. Zarządzanie konfiguracją również śledzi zależności pomiędzy artefaktami, tak wszystkie związane artykuły są aktualizowane po wprowadzeniu zmian.
- Zarządzanie zleceniami zmian: W procesie rozwoju systemu oprogramowania istnieje wiele artefaktów z różnymi wersjami. Zarządzanie polega na trzymaniu rejestru propozycji lub zleceń zmian.

Dyscyplina Konfiguracja i zarządzania zmianami

- Zarządzanie stanem i miarami – Zlecenia zmian mają stany takie jak nowy, zalogowany, zatwierdzony, przypisany i zakończony. Zlecenia zmian mają także atrybuty takie jak przyczyna (root cause) oraz natura (jak defekt lub rozszerzenie), priorytet itp. Te stany i atrybuty powinny być przechowywane w bazie danych, tak aby umożliwić tworzenie użytecznych raportów na temat postępów prac. Firma Rational posiada produkt, który umożliwia utrzymywanie takiego rejestru ClearQuest. Czynność ta wiąże się z procedurami, które trzeba wykonywać.

Dyscypliny zarządzania projektami

Planowanie projektami występuje na dwóch poziomach – zgrubnym (coarse-grained) zwanym planem faz, który opisuje cały projekt oraz serii szczegółowych planów iteracji, które opisują iteracje.

Ta dyscyplina skupia się głównie na ważnych aspektach iteracyjnego procesu wytwarzania oprogramowania. **Nie próbują objąć** natomiast wszystkich aspektów zarządzania projektami, na przykład:

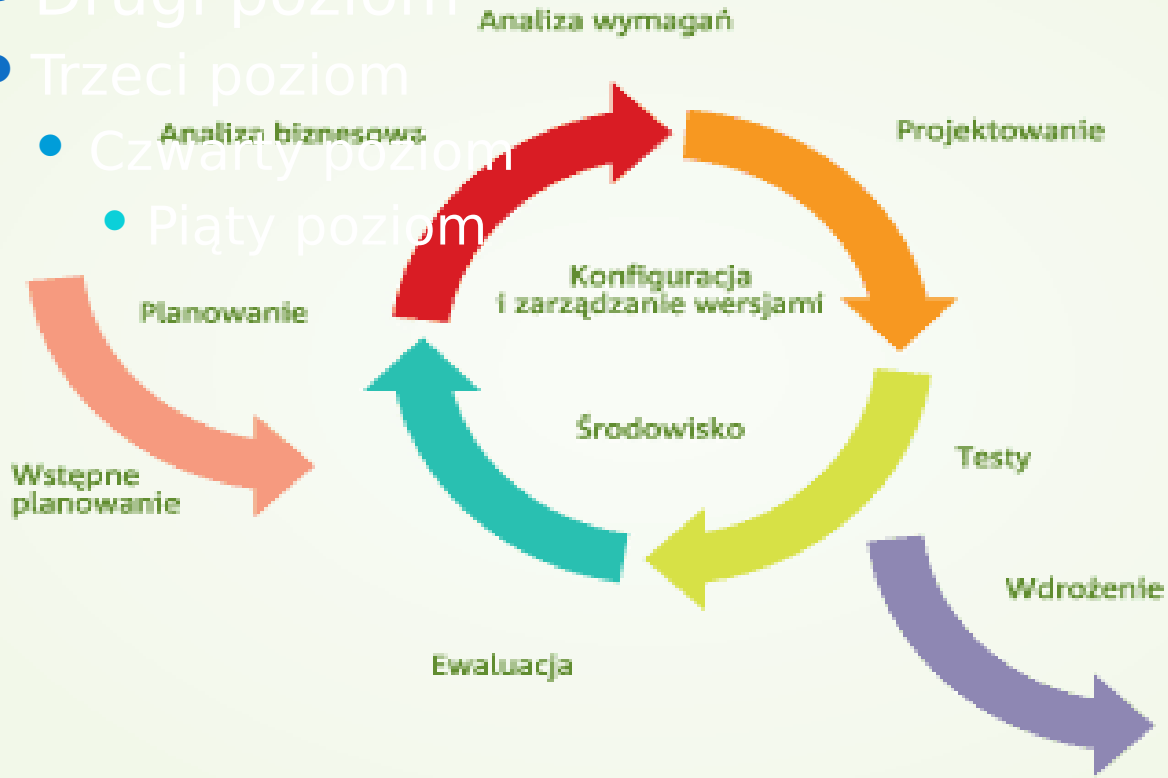
- Zarządzania zespołem: zatrudniania, szkoleń, opieki;
- Zarządzania budżetem: definiowania, alokowania itp.;
- Zarządzania umowami ze sprzedawcami i klientami;

Dyscypliny zarządzania projektami

Główne obszary dyscypliny:

- Zarządzanie ryzykiem;
- Planowanie projektu iteracyjnego, w ramach całego cyklu i pojedynczych iteracji;
- Monitorowanie postępu projektu iteracyjnego, miary;

- Kliknij, aby edytować style wzorca tekstu
  - Drugi poziom
  - Trzeci poziom
    - Czwarty poziom
      - Piąty poziom



Rysunek 2. Iteracyjny proces rozwoju oprogramowania (załącznik[1]).

Plan faz

Każda faza traktowana jest jako projekt, kontrolowany i mierzony poprzez **Software Development Plan** pogrupowany w podzbiór planów kontrolnych:

- **Plan miar** (Measurement Plan) – definiuje cele pomiarów, skojarzone miary, i proste miary, które będą gromadzone w projekcie w celu monitorowania jego postępu.
- **Plan zarządzania ryzykiem** (Risk Management Plan) – uszczegóławia w jaki sposób zarządzać ryzykami związanymi z projektem. Wymaga uszczegółowienia zadań zarządzania ryzykami, które będą wykonywane, przypisania do nich odpowiedzialności oraz dodatkowych wymaganych zasobów. W projektach mniejszej skali plan może być powiązany z Software Development Plan.
- **Lista ryzyka** (Risk list) – lista znanych i otwartych ryzyk posortowanych według ważności i skojarzonych z akcjami minimalizacji oraz planami awaryjnymi (mitigation and contingency actions).

Plan iteracji

Plan iteracji jest drobnoziarnistym planem, czasowo-sekwencyjnym, zestawem działań i zadań, przypisanych mu zasobów, zawierające zależności między zadaniami, do powtórzenia.

Są to zazwyczaj dwa plany iteracji aktywne w danym momencie:

- Obecny plan iteracji jest używany do śledzenia postępu w bieżącej iteracji.
- Kolejny plan iteracji jest używany do planu nadchodzących iteracji. Ten plan jest przygotowany pod koniec bieżącej iteracji.

Plan iteracji

Aby określić zawartość iteracji trzeba:

- planu projektu
- obecny stan projektu (zgodnie z planem, pod koniec, dużą liczbę problemów, pełzanie wymagania itp.)
- listę scenariuszy lub przypadków użycia , które muszą być zakończone do końca iteracji
- listę zagrożeń, które muszą być uwzględnione w końcu iteracji
- listę zmian, które muszą być zawarte w produkcie (poprawki błędów, zmiany w wymaganiach)
- wykaz dużych klas lub pakietów, które muszą być w pełni uwzględnione

Listy te powinny być w rankingu. Cele iteracji powinny być agresywne, tak że gdy pojawią się trudności, przedmioty mogą zostać usunięte z iteracji na podstawie ich szeregów. Dlatego też istnieje zestaw obsługiwanych artefaktów, które pomagają w budynku pomiarowym i w każdym planie iteracji.

Biblografia:

- http://translate.google.pl/translate?hl=pl&langpair=en%7Cpl&u=http://www.ambysoft.com/downloads/managersIntroToRUP.pdf
- http://brasil.cel.agh.edu.pl/~10sdczerner/page/RUP
- http://translate.google.pl/translate?hl=pl&sl=en&u=http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf&ei=d8CdTcHGMMyZOvHJ5NoE&sa=X&oi=translate&ct=result&resnum=4&ved=0CEAQ7gEwAw&prev=/search%3Fq%3DRational%2BUnified%2BProcess%26hl%3Dpl%26biw%3D985%26bih%3D503%26rlz%3D1R2SUNC_plPL389%26prmd%3Divnsb
- http://pl.wikipedia.org/wiki/Rational_Unified_Process

Rysunki:

- Załącznik 1 :http://www.michalwolski.com/tag/rational-unified-process/