



Zapewnienie jakości w projekcie informatycznym w oparciu o iteracyjne modele wytwórcze –Risk Driven Developing

Autor: Rafał Dobrosielski

O autorze:

Absolwent Politechniki Gdańskiej, wydziału Elektroniki Telekomunikacji i Informatyki, ukończył również studia podyplomowe z zakresu Inżynierii Oprogramowania i Zarządzania Projektami. Zawodowo, na co dzień, zajmuje się czynnie planowaniem, zapewnianiem i sprawdzaniem jakości w projektach informatycznych. Jest autorem szkoleń i warsztatów z zakresu zarządzania projektem informatycznym, zarządzania i zapewniania jakości produktów informatycznych oraz modeli procesów inżynierii oprogramowania.

Interesuje się analizą i modelowaniem procesów biznesowych w organizacjach jak również psychologią biznesu.

Email: rdobrosielski@paop.com.pl

Intermediate

Level

4

Magazine Number

Testowanie oprogramowania

Section in the magazine

„Każdy biznes może być lepszy!

Najbardziej zadowolony klient może być bardziej zadowolony!

Dążmy do perfekcji!”

Cel prezentacji

Celem prezentacji było przedstawienie mechanizmów zapewniania jakości w projekcie informatycznym realizowanym przy wykorzystaniu iteracyjnych modeli wytwarzania oprogramowania. Temat przedstawiono poprzez pryzmat zarządzania ryzykiem projektowym.

Wstęp

Sprzęt:

„XXX warrants to you as an end-user purchaser all XXX hardware products against defects in material and workmanship for a period of one/two year(s)”

Oprogramowanie:

„You expressly acknowledge and agree that use of the XXX software is at your sole risk. The XXX software and related documentation are provided „AS IS” and without warranty of any kind.....

.....When the XXX software prove defective, you (and not XXX) assume the entire cost of all necessary servicing, repair, or documentation.”

Powyższe stwierdzenia często spotykane wśród punktów umowy gwarancyjnej podczas zakupu sprzętu komputerowego czy AGD oraz podczas instalacji oprogramowania nadal skłaniają do głębokiej refleksji.

Producenci sprzętu, hardware'u, mimo jego rosnącej złożoności, wciąż zapewniają nas, że w momencie sprzedaży jest on bez wad i jest dla nas bezpieczny. Co więcej, biorą za niego pełną odpowiedzialność przez najbliższy rok czy nawet dwa.

Z oprogramowaniem jest wciąż „troszeczkę” inaczej. Producenci oprogramowania, zamiast zapewniać nas o jego niezawodności i nieszkodliwości, starają się zabezpieczyć na okoliczność poniesienia strat przez kupującego po użyciu ich dzieła.

Jeden z programów antywirusowych poinformował mnie podczas instalacji, że robię to na własną odpowiedzialność i ryzyko - jako użytkownik poniosę całkowity koszt związany z ewentualnymi stratami, które mogą powstać na skutek używania oprogramowania, którego zadaniem jest mnie zabezpieczać.

Oczywiście zgodziłem się z tymi jasno postawionymi warunkami i używam tego oprogramowania do dziś.

Producenci oprogramowania zdają sobie sprawę, że ich produkty, jak również cały projekt informatyczny, są obciążone wysokim ryzykiem. Nawet tzw. dojrzałe organizacje wciąż nie potrafią tego ryzyka dokładnie oszacować a następnie zneutralizować podczas realizacji projektu do dostatecznie niskiego poziomu – zanim produkt zostanie przekazany na rynek.

Ryzyko

Ryzyko – popularne definicje:

„Możliwość poniesienia straty”

[Software Engineering Institute]

„Ryzyko to niepewne wydarzenie, które – jeśli zajdzie – może mieć negatywny albo pozytywny wpływ na projekt”

[Project Management Institute]

„Ryzyko to niepewność rezultatu (wyniku)”

[The Office of Government Commerce, PRINCE2]

Mówiąc dalej o ryzyku, chciałby przypomnieć, że organizacje definiują ryzyko często jako możliwość poniesienia straty, czy też jako poziom niepewności przy osiągnięciu oczekiwanego rezultatu.

Źródła ryzyka w projekcie informatycznym

Źródła ryzyka w projekcie informatycznym związane są najczęściej z wykorzystywaną technologią. Aby „utrzymać” się na rynku i tworzyć oprogramowanie oparte na nowoczesnych rozwiązaniach wciąż używamy nowych technologii, których wydajność, skuteczność czy efektywność nie jest nam znana.

Procesy wytwórcze dla projektu informatycznego, mimo ciągłych prób formalizacji (czy też skrajnych uproszczeń), są skomplikowane – nie są sekwencyjne. Jeśli ta skomplikowana sieć działań zostanie choć trochę zaniedbana, zamiast do sukcesu poprowadzi nas prosto do porażki.

Do ponoszenia zwiększonego ryzyka motywuje nas również sam rynek i jego wysoki poziom konkurencyjności.

W projekcie informatycznym jesteśmy mocno uzależnieni od tzw. czynnika ludzkiego i zasady propagacji błędów – to głównie w naszej branży mały błąd powoduje tak duże, katastrofalne skutki.

Obszary ryzyka w projekcie informatycznym

Projekt informatyczny może stać się projektem bardzo wysokiego ryzyka, jeśli już na początku cele projektowe zostaną nadmiernie rozdmuchane, będą niemierzalne a korzyści, które projekt ma przynieść, zostaną rozmyte.

Podczas realizacji projektu, poziom ryzyka zależy od dynamiki informatyzowanej dziedziny – od częstości zmian usług, strategii biznesowej czy polityki u naszego klienta. Inaczej wygląda ryzyko w projekcie dla amerykańskiej armii, gdzie plany i wymagania są stałe i określone na najbliższych kilka lat a inaczej dla dynamicznej instytucji finansowej na rynku o dużej konkurencji.

Zwykle wprowadzona dzięki informatyzacji zmiana biznesowa pociąga za sobą kolejne zmiany. Patrzymy zatem szeroko na otoczenie projektu, gdyż nie tylko od możliwości adaptacyjnych klienta, ale i od spójności naszego produktu z resztą procesów w informatyzowanej organizacji może zależeć to, jak nasz produkt zostanie ostatecznie przyjęty przez odbiorcę, czy końcowych użytkowników.

Kolejnym źródłowym obszarem zagrożeń, na który (wbrew pozorom) mamy duży wpływ, jest fakt, że klient na początku projektu nie wie, czego chce – nie potrafi zdefiniować swoich oczekiwań. Iteracyjne modele

wytwórcze pozwalają nam skutecznie tym zagrożeniem zarządzać.

W branży informatycznej projektowanie jest fazą projektu. Na początku zdefiniowany jest zwykle jedynie czas i budżet projektu – zakres i jakość są, niestety, tylko ich wypadkową!

Rozpoczynając projekt informatyczny nie wolno nam też nie zdefiniować precyzyjnych, mierzalnych wymagań jakościowych, przejść do realizacji projektu bez ustalenia określonego poziomu np. wydajności, standardu interfejsu, profilu użytkowników itd.

Sukces projektu informatycznego

Sukces projektu informatycznego jest pojęciem wielowymiarowym; oczywiście inaczej może być rozumiany z punktu widzenia kierownika projektu (np. nie przekroczyć zaplanowanego budżetu i czasu), użytkownika (np. otrzymać oprogramowanie wyręczające go w powtarzalnej, trudnej pracy), testera (wysoka stopa wykrytych błędów w stosunku do błędów zgłoszonych po wdrożeniu) – każda z tych grup może odnieść swój własny sukces a mimo to produkt będzie nadawał się jedynie do kosza.

Otóż sukces projektu, nie tylko informatycznego, polega na tym, aby klient, zamawiający czy sponsor odniósł określone, zaplanowane korzyści biznesowe. Oczywiście jest to weryfikowalne dopiero po pewnym czasie od wdrożenia oprogramowania, ale jest to najbardziej obiektywna miara sukcesu i dojrzałości firmy informatycznej.

Cel biznesowy zleceniodawcy musi być mierzalny, jednakowo rozumiany przez wszystkich interesariuszy projektu – jest to według mnie krytyczny czynnik sukcesu.

Zarządzanie ryzykiem a model wytwórczy

Zarządzanie ryzykiem, dostępne mechanizmy, które możemy zastosować w celu jego neutralizacji zależą od zastosowanego modelu procesu.

Model procesów:

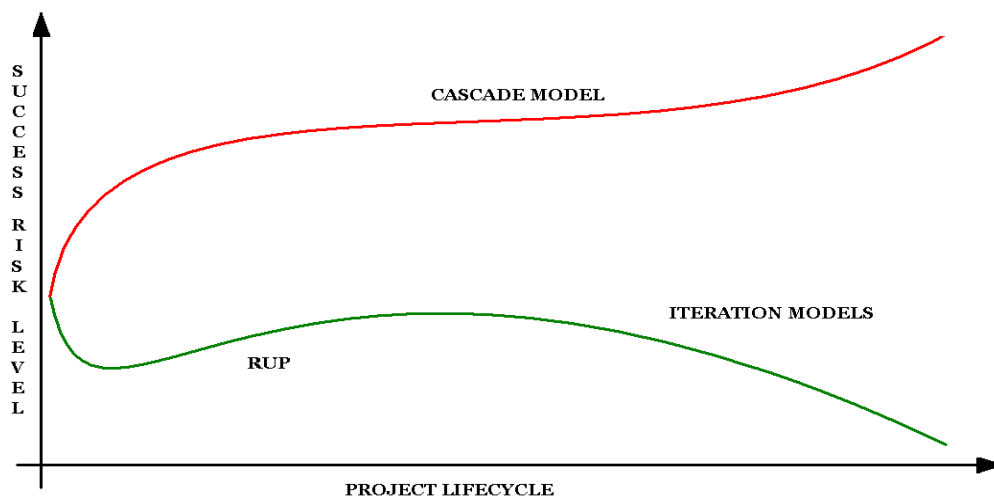
Model procesu jest usystematyzowanym zbiorem praktyk, które opisują cechy efektywnego procesu a ich efektywność została potwierdzona przez doświadczenie.

„Wszystkie modele procesów są złe

ale niektóre są użyteczne”

(George Box, Quality and statistics engineer)

Wszystkie modele procesów są „złe”, a przynajmniej niedoskonałe. Dojrzałe organizacje zwykle narzekają, że są one zbyt mało elastyczne, nie pozwalają rozwijać skrzydeł. Mniej dojrzałe mówią, że są zbyt mało sformalizowane, nie zawierają wszystkich potrzebnych wskazówek i dlatego tak łatwo się zgubić, czy popełnić błąd.



Zakładając, że realizujemy projekt najprostszym, intuicyjnym modelem kaskadowym, krzywa ryzyka osiągnięcia sukcesu w projekcie, wg mojej oceny, ciągle rośnie. Wciąż rośnie ryzyko, że przyjęte na początku założenia oraz zaproponowane i zaimplementowane rozwiązania (w przypadku braku ich wczesnej, cyklicznej, dostatecznie częstej weryfikacji) okażą się niezgodne z oczekiwaniami projektantów i klienta.

Naszym zadaniem jest - jak najwcześniej, zanim zainwestujemy duże pieniądze w projekt - uzyskać pewność, iż posiadamy wykonywalną i przetestowaną architekturę oczekiwanego rozwiązania, która z dużym prawdopodobieństwem jest w stanie sprostać sprecyzowanym wymaganiom jakościowym.

Zapewnienia jakości

Chciałbym bardzo mocno podkreślić, że wysoką jakość produktów w projekcie informatycznym osiągniemy wyłącznie dzięki ścisłej, częstej współpracy zamawiającego i wykonawcy – oraz wewnętrznej współpracy poszczególnych zespołów projektowych.

Osiąganie jakości musi być zintegrowane ze wszystkimi zadaniami, czynnościami procesów zachodzących w projekcie – zapewnianie jakości nie powinno stanowić osobnej dyscypliny inżynierii oprogramowania.

Coraz częściej słyszymy o tworzeniu przez dojrzałe organizacje interdyscyplinarnych zespołów zapewniania jakości (QA), złożonych nie tylko z testerów, ale także z analityków, projektantów, developerów... . To ich wspólnej ocenie podlega wówczas każdy ważny, wyjściowy artefakt projektu.

Każdy, indywidualnie, w projekcie musi czuć, iż odpowiada za jakość wytwarzanych w nim produktów – i, co ważniejsze, że ma na nią wpływ.

Zadania procesu zapewniania jakości (QA)

Do najważniejszych zadań procesu zapewniania jakości powinno należeć:

- zidentyfikować i zdefiniować wskaźniki (mieralne) akceptowalnej jakości,
- zidentyfikować i zaplanować odpowiednie pomiary jakości (min. plany i harmonogram testów)
- zidentyfikować i odpowiednio rozwiązać zagadnienia i problemy jakościowe (tym samym poszczególnie ryzyka projektu) tak szybko i efektywnie, jak to możliwe.

Najlepsze praktyki wytwarzanie oprogramowania

Chciałbym przedstawić kilka najważniejszych, tzw. najlepszych praktyk wytwarzania oprogramowania, które stały się podwaliną opracowywania iteracyjnych modeli twórczych, jak również tzw. metodyk zwinnych.

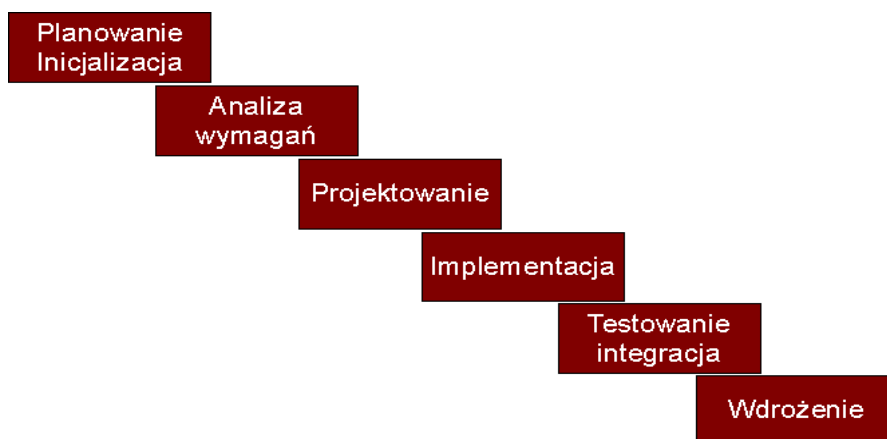
Dobre praktyki – to sprawdzone, komercyjne metody tworzenia oprogramowania, które – używane razem i dobrze skoordynowane – skutecznie radzą sobie z podstawowymi problemami tworzenia systemów informatycznych.

Do najważniejszych z nich należą:

- atakuj ryzyko najwcześniej, jak to możliwe – co oznacza: testuj najwcześniej, jak to możliwe oraz tak często, jak to możliwe – czyli tak często, jak tylko potrafisz tym procesem efektywnie zarządzać;
- buduj produkt cenny dla klienta – oznacza to, że należy zbierać wymagania nie tylko funkcjonalne, ale również dotyczące użyteczności, wydajności, pielęgnowalności poprzez pryzmat uzgodnionego sukcesu projektu; oznacza to również, aby zebrane wymagania i wysokopoziomowe projekty systemu dokumentować w sposób zrozumiały dla klienta – dzięki temu będą one mogły być poddane wcześniejszej (przed implementacją) i pełniejszej ocenie przez klienta;
- koncentruj się na częstym uzyskiwaniu wykonywalnego oprogramowania – dokumenty i projekty są ważne, ale nie są one najlepszym wskaźnikiem postępu projektu; najlepszym, najbardziej obiektywnym wskaźnikiem postępu jest wykonywalne oprogramowanie, które pomyślnie przeszło zaplanowany (nawet częściowy) zbiór i rodzaj testów;
- zaakceptuj fakt i miej pewność, że zmiany wystąpią – możliwość wprowadzania zmian potraktuj jako szansę na dokonywanie korzystnych zmian w projekcie;
- zaprojektuj i zaimplementuj architekturę systemu tak wcześnie, jak to możliwe i przetestuj ją!
- zbuduj swój system z komponentów.

Model kaskadowy a model iteracyjny

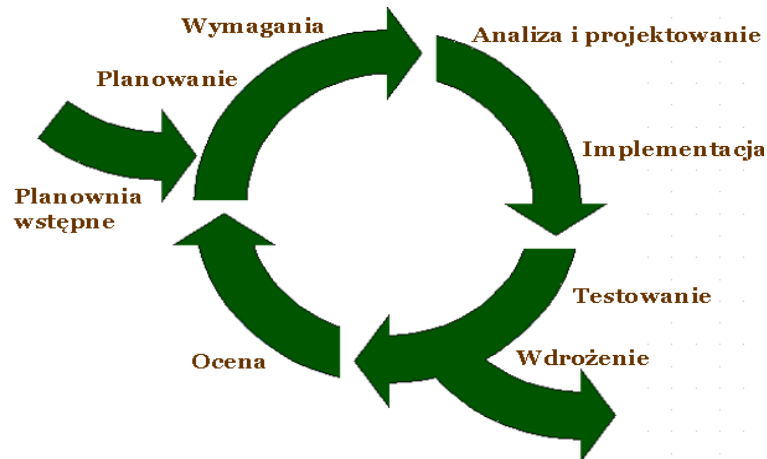
Model klasyczny:



Ponownie, nie negując użyteczności modelu kaskadowego dla pewnego typu projektów, chciałbym podkreślić, że często to on prowadzi nas prosto do porażki, gdyż:

- wymagań nie można zamrozić;
- nie jest możliwe jednorazowe poprawne zaprojektowanie systemu;
- nie jesteśmy w stanie przewidzieć wszystkich zagrożeń w fazie planowania i projektowania;
- projekty informatyczne działają na zbyt dynamicznym rynku – zmiany prawa, technologii, usług;
- wprowadzanie zmian w końcowej fazie projektu w takim podejściu wiele kosztuje a każda zmiana pociąga zmiany następne i dodatkowe koszty.

Model iteracyjny:

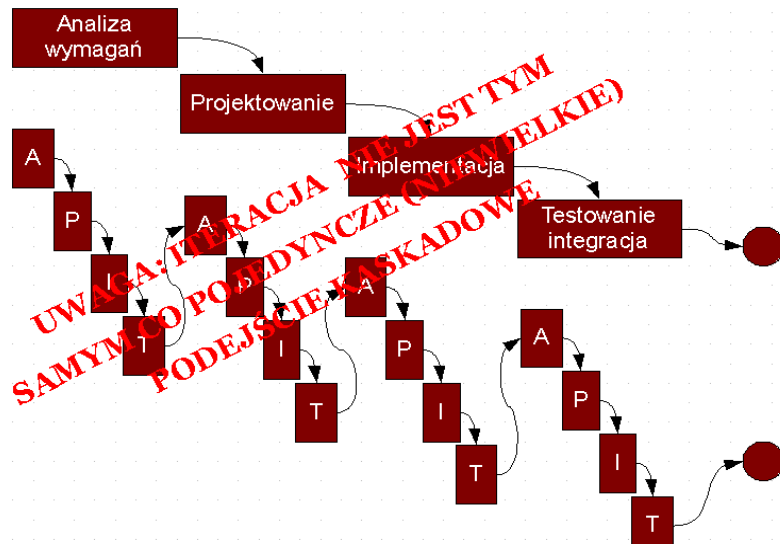


Znając wady modelu klasycznego, chciałbym pokrótce przypomnieć cechy modelu iteracyjnego, mianowicie:

- proces iteracyjny składa się z ciągu kroków przyrostowych;
- każda iteracja obejmuje większość dyscyplin (często wszystkie);
- do każdej iteracji przypisany jest wyraźnie określony zbiór jasnych, mierzalnych celów;
- wynikiem iteracji jest działająca implementacja systemu końcowego;
- każda iteracja opiera się na wynikach iteracji poprzedniej;
- występuje zmiana zaangażowania poszczególnych dyscyplin w różnych fazach projektu - w początkowych iteracjach duży nacisk kładzie się na analizę i projektowanie, w końcowych na implementację i testowanie.

Jak zjeść słonia? Kęsami?

W tym miejscu chciałbym stanowczo podkreślić, że poszczególna iteracja nie jest tym samym, co pojedyncze podejście kaskadowe.



Niekaskadowe uszeregowanie czynności w ramach iteracji zależy od:

- indywidualnych okoliczności, w jakich prowadzony jest projekt – chwilowych uwarunkowań wewnętrznych i zewnętrznych;
- wyników iteracji poprzednich;
- strategii;
- naszej taktyki;
- możliwości adaptacji;
- ale głównie od zidentyfikowanych ryzyk i zagrożeń.

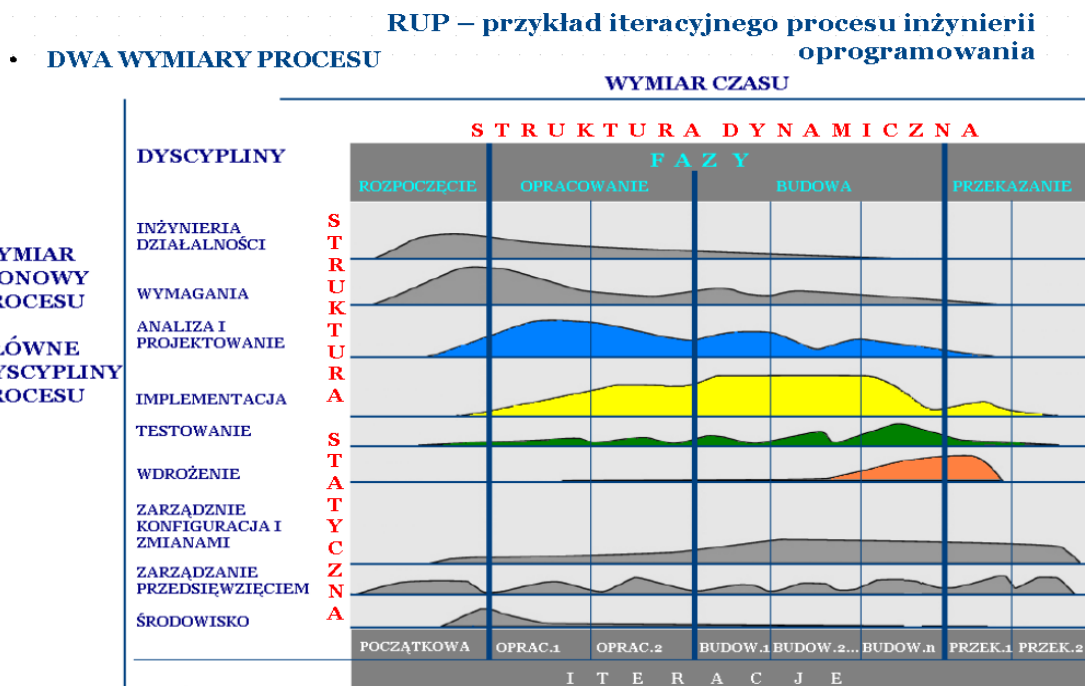
Realizując projekt modelem iteracyjnym musimy zadać sobie następujące pytania:

- „W jaki sposób te działania (iteracja po iteracji) poprowadzą nas w sposób systematyczny do uzyskania końcowego produktu?”
- „Jak uniknąć tego, by w każdej iteracji nie zaczynać wszystkiego od początku?”

Iteracyjne modele procesów dostarczają nam na te pytania szczegółowej odpowiedzi.

Rational Unified Process – jako przykład modelu iteracyjnego

Na poniższym rysunku widzimy tzw. wykres wielogarnbny przedstawiający istotę procesu RUP jako przykład modelu iteracyjnego.



Do najważniejszych cech tego modelu należy to, iż proces iteracyjny w RUP jest podzielony na etapy/fazy i iteracje.

Proszę zauważyć, że poszczególne fazy i iteracje nie są tworzone poprzez tradycyjną sekwencję dyscyplin inżynierii oprogramowania (planowanie analiza, projektowanie, implementowanie, scalanie) – ale są do nich całkowicie ortogonalne.

Aby skutecznie kierować pracami i postępami w projekcie realizowanym modelem iteracyjnym, trzeba wyznaczyć tzw. punkty kontrolne dla poszczególnych etapów/faz projektu z jasno określonymi kryteriami osiągnięcia poszczególnych kamieni milowych. Są to tzw. cele długoterminowe. Natomiast sekwencje czynności w ramach pojedynczej iteracji trzeba podzielić i zorganizować tak, aby móc podporządkować je konkretnym, krótkoterminowym celom.

Wykres wielogarnbny wyraźnie podkreśla, że w mniejszym lub większym stopniu, niemal wszystkie grupy projektowe uczestniczą w pracach projektu poprzez wszystkie jego fazy. Często zwraca uwagę również fakt, że proces testowania rozpoczyna się jeszcze przed implementacją a analiza trwa również wtedy, gdy produkt jest wdrażany.

Ustanawiane w poszczególnych etapach i iteracjach kamienie milowe są efektem współpracy zespołów projektowych (ze szczególnym wkładem zespołu QA) i wynikiem wspólnego, ciągłego szacowania ryzyka.

Autorem zidentyfikowanego ryzyka może być każdy interesariusz projektu! Wszyscy powinni mieć dostęp do tzw. rejestru ryzyka.

W przedsięwzięciu iteracyjnym, miarą postępu prac jest łącznie:

- lista rozważonych przypadków użycia, czy zrealizowanych właściwości użytkowych systemu,
- lista przebytych wariantów testów (wraz z wynikami),
- lista spełnionych wymagań eksploatacyjnych,
- (przede wszystkim) lista wyeliminowanych zagrożeń!

Korzyści płynące ze stosowania modeli iteracyjnych

Ścisła współpraca zespołów projektowych i klienta przy wykorzystaniu możliwości i zalet iteracyjnego podejścia do wytwarzania oprogramowania zapewni, że największe zagrożenia będą wykrywane i neutralizowane podczas początkowych operacji scalania – testowanie architektury (już po kilku tygodniach), pozwala określić, które z przewidywanych zagrożeń są rzeczywiste (narzędzia lub ich brak, umiejętności, możliwości technologiczne, zakupione komponenty), odkryć nowe, niespodziewane zagrożenia a zapobieganie nim stanie się mniej kosztowne i łatwiejsze.

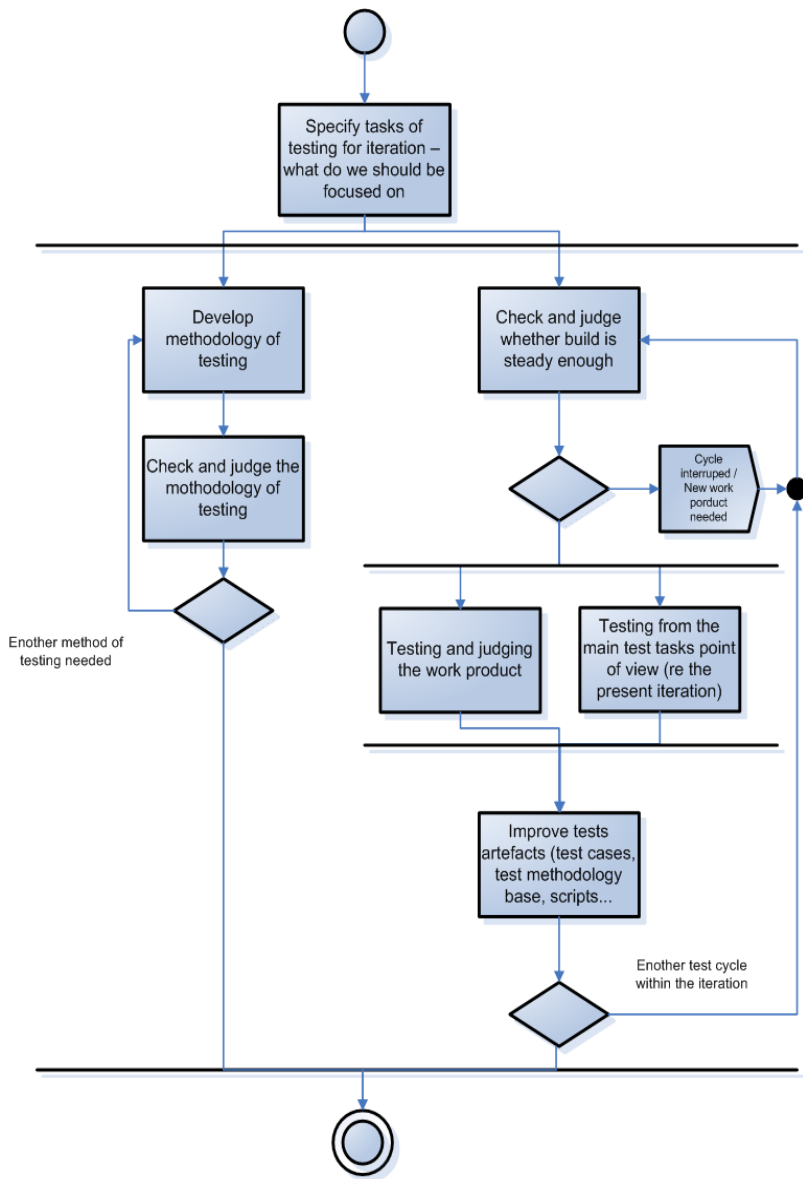
Zespół twórców zmuszony będzie do koncentrowania się na sprawach najbardziej istotnych – małe przyrosty, lepiej zdefiniowane zadania, mierzalne wyniki. Dzięki temu jest on chroniony przed sprawami, które odwracają jego uwagę od rzeczywistych zagrożeń przedsięwzięcia.

W takim przedsięwzięciu jest możliwość reagowania na zmieniające się wymagania – wczesne wykrywanie niezgodności pomiędzy wymaganiami, wynikami projektowania i wynikami implementacji. Weryfikacja wymagań z ich źródłem jest prostsza. Staje się możliwe przedstawianie klientowi przyrostowych wyników pracy w postaci wykonywalnego oprogramowania, a to sprzyja szybszemu dochodzeniu do wymagań rzeczywistych.

Scalanie nie jest już wielkim wybuchem, który następuje na koniec przedsięwzięcia, gdyż każda iteracja kończy się scalaniem, co minimalizuje późniejsze przeróbki (często około 40% wysiłku marnowano właśnie na końcowe scalanie).

Proces testowy w modelu iteracyjnym

Na poniższym rysunku pokazano przykładowy proces testowy (na podstawie RUP) w iteracyjnym modelu inżynierii oprogramowania w ramach jednej iteracji.



Jak wykazałem wcześniej (za pomocą wykresu wielogarnego), w modelu iteracyjnym testy są projektowane i wykonywane tak szybko/wcześnie, jak to możliwe – jeszcze przed rozpoczęciem implementacji. W procesie tym większość użytecznych przypadków testowych (szczególnie zautomatyzowanych) jest akumulowanych jako testy regresji. Rezultaty testowania mogą być przedstawiane interesariuszom bardzo wcześnie, kiedy poprawki są tanie. Dzięki małym, iteracyjnym przyrostom, testerzy lepiej rozumieją obiekt testowania. Plan testów i przypadki testowe rozwijają się wówczas razem z produktem, gdyż poszczególne przypadki testowe są projektowane na podstawie planu zgrubnego – planu testów dla przedsięwzięcia (część Planu jakości) oraz Planu testów dla iteracji i są rozwijane razem z kodem oprogramowania. Zwiększa to szanse na automatyzację testów.

Z rysunku można odczytać, iż każda iteracja (a tym samym „Plan (zgrubny) testów”) powinna zawierać tzw. główne zadanie testowe dla iteracji.

Zadanie testowe dla iteracji jest to kluczowy aspekt planowania, krótka deklaracja na poziomie zarządzania, oparta na oszacowaniu ryzyka, pozwalająca zrozumieć testerom cel iteracji, który powinien pokrywać się z aktualnym kontekstem projektu i przyświecać zespołom testowym w ramach całej iteracji, oprócz normalnego testowania postępów funkcjonalnych.

Podsumowanie

W swojej prezentacji starałem się wykazać, że modele iteracyjne pozwalają na skuteczne zarządzanie projektem informatycznym. Dostarczają one wielu narzędzi neutralizowania ryzyka, powodując jego obniżenie na początku projektu i skuteczne nim zarządzanie w dalszej jego części.

Dzięki modelom iteracyjnym w naturalny sposób następuje zacieśnienie współpracy pomiędzy zespołami projektowymi a zapewnianie jakości w projekcie staje się odpowiedzialnością każdego. Poszczególne artefakty projektu są częściej testowane, podlegają wspólnym przeglądom a wcześnie wykrywane błędy są tańsze do naprawienia.

Iteracje dają możliwości adaptacji przebiegu i zakresu projektu do zmieniających się czynników zewnętrznych, zidentyfikowanych zagrożeń i napotkanych trudności.

Modele iteracyjne pozwalają na automatyzację procesu testowego – skrypty testowe są rozwijane wraz z rozwojem oprogramowania.

Wszystkie powyższe korzyści pozwalają nam na łatwiejsze i bardziej skuteczne zapewnianie wysokiej jakości produktów przedsięwzięcia informatycznego.