ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 1

Wersja 1

Organizacja zespołu projektowego i środowiska pracy

Spis treści

Informacje o module

Opis modułu

W tym module znajdziesz informacje dotyczące szeroko pojętej tematyki związanej z tworzenie oprogramowania, inicjowaniem projektu, tworzeniem zespołu projektowego, elementami zarządzania projektem informatycznym, tworzeniem środowiska pracy czy wyborem narzędzi programistycznych. Moduł stanowi w głównej mierze wprowadzenie teoretyczne zilustrowane przykładami, które jest niezbędne dla dalszego pogłębiania wiedzy w kolejnych modułach.

W tym module ograniczymy się jedynie do identyfikacji członków zespołu i ich cech. Ma to na celu poznanie się lepsze grupy i uświadomienie członkom zespołu ich mocnych i słabych cech. Zatem ta identyfikacja ograniczy się bardziej do organizacji zespołu jako organizacji/firmy, a nie zespołu konkretnego projektu. Jest o tyle naturalne, gdyż rzadko firma realizuje jeden projekt, znacznie częściej zespół firmy realizuje wiele projektów na raz i jego członkowie są "rozrywani" przez poszczególnych kierowników projektów.

Cel modułu

Celem modułu jest podanie gruntownych informacji, wyjaśniających podejście do procesu tworzenia oprogramowania w sposób metodyczny i całościowy.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- wiedział jakie fazy składają się na cały cykl życia oprogramowania,
- wiedział jakie role można wyróżnić wśród członków zespołu projektowego,
- potrafił zidentyfikować role członków zespołu,
- potrafił zorganizować zespół projektowy,
- rozumiał rolę inżynierii oprogramowania jako źródła różnego rodzaju dobrych praktyk w zakresie tworzenia oprogramowania,
- umiał określić potrzeby dla środowiska pracy,
- rozumiał rangę i rolę wyboru narzędzi pracy.

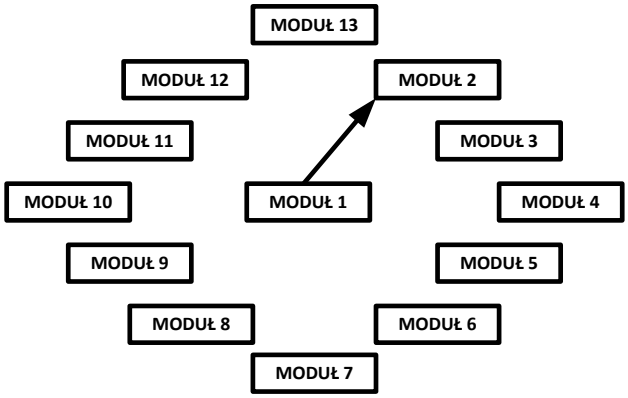
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać zagadnienia prezentowane na wykładach z przedmiotu Inżynieria oprogramowania, Zarządzanie projektem lub podobnych,
- znać ideę tworzenia oprogramowania za pomocą zintegrowanych środowisk programistycznych,
- rozumieć podstawowe pojęcia tworzenia oprogramowania, jak np. cykl życia oprogramowania.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu nie jest wymagana znajomość innych modułów.



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Kończymy właśnie studia, mamy fajną paczkę koleżanek/kolegów, posiadających różne kwalifikacje/kompetencje. Coraz częściej uświadamiamy sobie, że pomimo tego iż na studiach wpojono w nas wiedzę z różnych dziedzin informatyki, to jednak wcale nie każdy z nas jest świetnym programistą, a już zupełnie niewielu ma talent artystyczny i umie zaprojektować ładną stronę internetową (i na co nam były wykłady z Grafiki komputerowej?). Co więcej Janek – same piątki od góry do dołu, chyba specjalista od wszystkiego – obecnie nie ma ochoty projektować interfejsu użytkownika, nie mówiąc już o programowaniu aplikacji bazodanowy – coraz częściej przebąkuję, że chciałby zająć się wyłącznie projektowanie architektury systemów informatycznych oraz zarządzaniem. O dziwo Marysia, która do tej pory nie przejawiała większej ochoty do programowania, zmieniła zdanie od kiedy na ostatnim przedmiocie specjalizacyjnym mieliśmy zajęcia z Visual Studio – nie dość, że polubiła programowanie wizualne, to jeszcze wręcz z upodobaniem zaczęła zajmować się projektowaniem ergonomicznych interfejsów użytkownika, z czego zresztą napisała pracę magisterską.

Ostatecznie, podczas wspólnej imprezy naszej paczki, zaczęliśmy snuć plany, co też zrobimy zaraz po skończeniu studiów. Padały przeróżne propozycję, począwszy od wspólnego „najazdu" na jakąś „wielką korporację" i zażądanie, by nas wszystkich przyjęli, a skończywszy na założeniu wspólnej firmy, tworzącej oprogramowanie na potrzeby małych i średnich przedsiębiorstw. Z tą inicjatywą wyszła Basia – osoba, która zawsze miała najwięcej pomysłów i nie bała się wyzwań. Już na „trzeźwo" kilka dni później, zrobiliśmy spotkanie i przeprowadziliśmy głosowanie – wygrała koncepcja założenia firmy tworzącej oprogramowanie – postanowiliśmy założyć firmę „10-ciu wspaniałych".

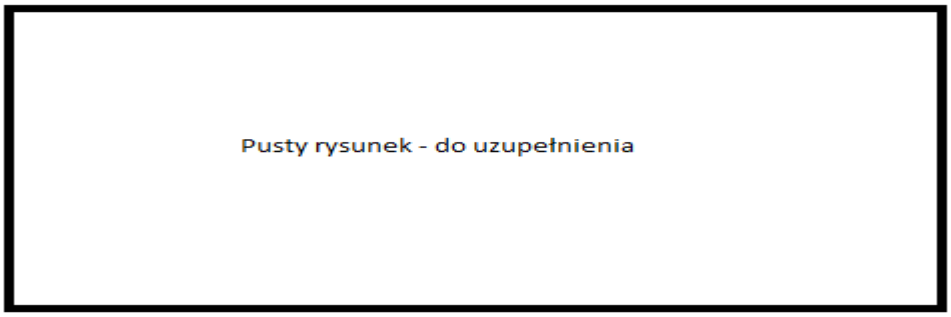
Na kolejnym spotkaniu przyszło „otrzeźwienie", no tak, same chęci nie wystarczą… zrodziły się następujące pytania:

1. Jaką formę działalności gospodarczej powinniśmy wybrać?
2. W oparciu o jakie technologie realizować nasze rozwiązania?
3. Jaką metodę/metodologię pracy należy przyjąć?
4. Jak podzielić się pracą/obowiązkami?
5. Jak zorganizować przyjazne środowisko pracy?
6. Jakie wybrać narzędzia do pracy?

Podstawy teoretyczne

Ogólnie o cyklu tworzenia oprogramowania

Jak wiemy, odpowiedzi na część powyższych pytań (odpowiedź na pierwsze z nich nie należy do zakresu naszych zainteresowań) daje Inżynieria Oprogramowania jako dziedzina naukowa zajmująca się procesem wytwarzania, wdrażania i pielęgnacji oprogramowania. W zależności od Autora i sposobu w jaki ujmuje temat wyróżniamy nieco odmienne sposoby podziału procesu realizacji i wdrażania oprogramowania na kolejne etapy. My zaproponujemy podział zaprezentowany poniżej (Rys. 1).



Rys. 2. Ilustracja etapów tworzenia, wdrażania i konserwacji oprogramowania.

Na rysunku tym wyróżnione etapy można scharakteryzować następująco:

1. **Faza strategiczna** – czas w którym, prowadząc rozmowy z klientem lub jego przedstawicielami – próbujemy odpowiedzieć na pytanie, czy realizacja tego przedsięwzięcia jest możliwa, a jeśli tak to jakie metody będą najlepsze?
2. **Faza tworzenia** – tworzenie oprogramowania, według przyjętej metodologii (stanowiące element tzw. cyklu życia oprogramowania) oraz przy użyciu określonych narzędzi w oparciu o wybrane technologie. Zatem będą w tym różne procesy wytwórcze oraz testy akceptacyjne.
3. **Faza wdrożenia** – oprogramowanie, właściwie niezależnie od wielkości, jeśli tylko jest tworzone na potrzeby klienta, to również w umowie o tworzenie zawarty jest punkt związany z wdrożeniem. Wdrożenie jest niczym innym jak zainstalowaniem, skonfigurowaniem (ewentualnie parametryzacją) oprogramowania w środowisku uruchomieniowym klienta.
4. **Faza testów produkcyjnych** – oprogramowanie po stworzeniu i przetestowaniu przez producenta (faza tworzenia) musi zostać uruchomione i przetestowane w środowisku docelowym (zwanym produkcyjnym), przy czym zwykle nie na rzeczywistych danych, a jeśli już to nie jako jedyny system (wtedy uruchamia się nowe oprogramowanie obok istniejącego).
5. **Faza utrzymania** – każde oprogramowanie, podobnie jak np. samochód wymaga serwisowania, przy czym w przypadku programowania nie wynika to z faktu jego starzenia się (co nie ma miejsca) a dezaktualizacji (np. w świetle zmieniających się przepisów prawa). Stąd też firma tworząca oprogramowanie zwykle od razu proponuję tzw. maintenance na określonych warunkach.

Jak zauważyliśmy wcześniej, różni Autorzy proponują odmienne koncepcje podziału, wynika to zwykle z odmiennego doświadczenia jak i też często chęci ukazania problemu z konkretnego punktu widzenia. Nie jest to miejsce, w którym będziemy rozstrzygać takie spory, dość zauważyć, że powyższy podział wyczerpuje, czy też pokrywa, całość życia oprogramowania, począwszy od pomysłu, aż po jego pielęgnację.

Zauważmy jednak, że fazy wymienione powyżej zostały dobrane w taki sposób, by wyraźnie podkreślić odmienność pewnych etapów życia oprogramowania – jedynie fazę testów produkcyjnych można by włączyć do fazy wdrożenia bez większej szkody. Z naszego punktu widzenia (tego podręcznika) najbardziej interesująca jest faza tworzenia i nią będziemy się dalej zajmować. Jednak by lepiej zrozumieć rolę fazy tworzenia, w tym miejscu poświęcimy trochę miejsca na omówienie pozostałych faz.

Faza strategiczna

Jest to faza z jednej strony szalenie istotna, z drugiej często lekceważona. Istotą tej fazy jest odpowiedź na pytanie, czy warto ten projekt realizować? A jeśli tak, to jakimi metodami i przy użyciu jakich technologii? Czemu zatem jest ona lekceważona? Głównie za sprawą chęci „ubicia interesu”. Zauważmy, że idąc na rozmowę z potencjalnym klientem, angażujmy pewne siły i zasoby, im więcej takich spotkań, tym większe są nasze koszty, które chcielibyśmy zrekompensować

poprzez podpisany i zrealizowany kontrakt. Często też handlowiec, który prowadzi takie rozmowy jest rozliczany od ilości podpisanych kontraktów. Stąd też podchodzimy do tej fazy niejako z pozycji „musi się udać”.

Lekceważenie realiów biznesowych i technologicznych, bywa bardzo często zgubne. Realizacja przedsięwzięcia informatycznego musi być traktowana jak każde inne przedsięwzięcie biznesowe, ma ono swoją cenę (koszty) i musi ona być zrekompensowana poprzez przychody z jego realizacji. Stąd też jest bardzo ważne, by w tej fazie zidentyfikować możliwie dokładnie:

1. zakres realizacji projektu,
2. szacunkowy budżet,
3. dopuszczalny czas realizacji.

Pamiętajmy jednak, że jest to faza wstępna (zwykle przed fazą analizy), na którą nie możemy poświecić za dużo czasu, gdyż wszystko wiąże się z kosztami.

Odpowiedź na drugie pytanie („Jakimi metodami i przy użyciu jakich narzędzi zrealizujemy ten projekt”) jest równie ważna. Pamiętajmy, iż nie musimy upierać się pisać wszystkiego od początku, można przecież wykorzystać istniejące narzędzia i rozwiązania do rozbudowy lub po prostu dostosowania do potrzeb klienta. Klient zawsze będzie patrzył poprzez pryzmat racjonalizacji wydatków, a projekt informatyczny jest inwestycją, jak każdy inny zakup, zatem jest istotna wartość ROI (zwrot z inwestycji). Co więcej, racjonalizacja kosztów i nasz w tym udział, może być bardzo cenione przez klienta, dzięki czemu nawiążemy długoletnią i owocną współpracą.

Odnieśmy się teraz do tego w jaki sposób faza ta odnosi się do, najbardziej nas interesującej, fazy tworzenia. Otóż o tworzeniu może być mowa jedynie, jeśli w fazie strategicznej podejmiemy wspólnie z klientem decyzję, że program ma być tworzony (w całości lub części). Może jednak się okazać, że zapadnie decyzja o wdrożeniu istniejącego produktu. W takim przypadku przeskakujemy od razu do fazy wdrażania. W przypadku podjęcia decyzji o tworzeniu w fazie strategicznej zapadają zwykle decyzje o:

1. Kluczowych wymaganiach wobec nowego systemu – tzw. wymagania funkcjonalne.
2. Preferowanych technologiach i posiadanych zasobach – tzw. wymagania niefunkcjonalne.
3. Kluczowych użytkownikach – spis osób, który w późniejszym czasie bardzo się przyda.

Na sam koniec, warto nadmienić, że faza strategiczna może być przeprowadzana przez inna firmę niż ta, która będzie tworzyć oprogramowania. Często dobrą praktyką jest powierzenie tego zadania, lokalnym służbą informatycznym klienta wspieranym przez zewnętrznych konsultantów.

Faza tworzenia

W fazie tej tworzymy oprogramowanie w cyklu i przy zastosowaniu metodyki, którą uznamy za odpowiednią lub którą posługuje się nasz zespół. Jak wiemy z teorii inżynierii oprogramowania istnieje wiele metodyk służących do jego wytwarzania. Nie będziemy tutaj ani ich przestawiali, ani też dokonywali porównań, która jest z nich lepsza. Wynika to z faktu iż w dalszej części kursu będziemy odnosili się do lekkiej metodyki zarządzania jaką jest Scrum, gdyż środowisko Visual Studio 2010 oraz Team Foundation Server 2010 dobrze wspiera cykl wytwarzania oparty o tę metodykę.

W tym miejscu poczynimy jedynie uwagę, że coraz więcej firm tworzących oprogramowanie skłania się w stronę metodyk lekkich w tym metodyki Test Driven Dvelopment (TDD), u podstaw której leżą testy. W metodyce TDD (o której więcej powiemy sobie w module ???) nie wolno napisać kawałka kodu, bez wcześniejszego przygotowania dla niego testów jednostkowych.

Faza wdrożenia

Zarówno w przypadku produktu gotowego, parametryzowanego (np. systemy ERP) jak i systemu przed chwilą przez nas stworzonego dochodzimy w pewnym momencie do etapu wdrożenia. W

tym miejscu produkt jest instalowany u klienta, zwykle w dwóch „środowiskach": testowym i produkcyjnym. Po instalacji jest konfigurowany, parametryzowany (o ile istnieje taka potrzeba), na koniec prowadzone są zwykłe testy uruchomieniowe, tworzona jest dokumentacja powykonawcza i prowadzone są szkolenia, tak dla zwykłych użytkowników jak i informatyków po stronie klienta.

Jasne jest, że jest to faza ważna. Co więcej wielu praktyków zauważa, że nawet najlepszy system można źle wdrożyć, a w takiej sytuacji opinia o tym systemie jest wydawana poprzez pryzmat wdrożenia. Stąd też ranga tej fazy w praktyce jest bardzo wysoka i większość firma zdaje sobie z tego sprawę.

Z naszego punktu widzenia wiedza o tej fazie ma znaczenie w kontekście połączenia jej z fazą tworzenia. Musimy zauważyć, że większość działań fazy wdrożenia nie będzie zrealizowanych w sposób właściwy, jeśli nie będziemy mieli np. dobrej dokumentacji technicznej projektu. Dla przykładu, aby przeprowadzić proces instalacji stworzonego systemu u klienta musimy mieć dokładną wiedzę o wymaganiach tego systemu wobec innych programów (wersji bazy danych, serwera aplikacji, itp.). Dlatego też proces dokumentowania w trakcie tworzenia oprogramowania przekłada się wprost na poprawność procesu wdrożenia.

Faza testów produkcyjnych

Jak to wcześniej zauważyliśmy, często ta faza nie jest wyróżniana osobno, a jedynie stanowi element fazy wdrożenia, co więcej wyjątkowo rzadko testy produkcyjne wydziela się jako osobny etap (choć oczywiście będzie to jeden z podetapów wdrożenia jako takiego). Pojawia się zatem naturalne pytanie czemu my tak uczyniliśmy? Wynika to z obserwacji, iż kluczowe znaczenie dla sukcesu startu produkcyjnego, a zatem pomyślnego przejścia testów produkcyjnych, ma testowanie już na etapie tworzenia oprogramowania. Stąd też uwypuklamy ważność testów produkcyjnych, jako momentu ostatecznej akceptacji systemu informatycznego przez klienta. Jednak nie należy sądzić, iż testy wykonywane w trakcie tworzenia (o czym pisaliśmy wyżej) są tym samym co testy produkcyjne.

Testy produkcyjne, to procesy, które są wykonywane już u Klienta w docelowym środowisku produkcyjnym, zatem na sam koniec wdrożenia. Z punktu widzenia Klienta i Wykonawcay mają być ostatecznym weryfikatorem poprawności wdrożenia. Obejmują one całość systemu wdrażanego jak i jego powiązania z systemami zewnętrznymi, Stąd też nie są tym samym co testy w trakcie wytwarzania. Łatwo to zilustrować następującym przykładem. Klient posiada system dostępu do konta bankowego z którego są ściągane przelewy, okazuje się, że przelewy te nie bardzo można zidentyfikować po stronie naszego systemu. Po pewnym czasie okazało się, że bank zmienił format pliku w zakresie reprezentacji numerów kont, co spowodowało, że nie „widzimy" tych kont. Czy to wina naszego systemu? Nie! Czy to wina Banku? Nie! A jednak nie działa połączenie – takie właśnie problemy wychodzą w przypadku testów produkcyjnych.

Faza utrzymania

W fazie tej firma tworząca oprogramowanie przechodzi w rolę wsparcia oraz świadczenia usług rozwiązywania trudnych problemów. W jaki sposób następuje rozliczenie pomiędzy firmą a klientem (płatność za interwencję, ryczałt, itd.) zależy od przyjętego modelu biznesowego i nie jest dla nas tu interesujący. Niemniej warto zwrócić uwagę na fakt, iż podobnie jak to było w przypadku fazy wdrażania, również i tu nie możemy jako wykonawca świadczyć tych usług na wysokim poziomie, o ile nie mamy rzetelnej dokumentacji projektowej (powstającej na etapie tworzenia programu). Poprzestaniemy jedynie na tym stwierdzeniu i nie będziemy dalej omawiać tej fazy.

Metodyki tworzenia oprogramowania

W teoria inżynierii oprogramowania uczono nas przeróżnych modeli cyklu życia oprogramowania, od sekwencyjnych, poprzez ewolucyjne, a skończywszy na tzw. metodykach lekkich. Zasadniczo wybór jest tak szeroki, że początkującej osobie, zwłaszcza takiej, która nie uczestniczyła w żadnym

rzeczywistym projekcie, trudno podjąć decyzję jaki model wybrać, by rozpocząć tworzenie oprogramowanie.

Nie będziemy w tym miejscu prowadzili szerokich dywagacji na temat wyższości jednej metodyki nad inną, zwrócimy jedynie uwagę na kilka rzeczy , które powinniśmy rozważyć. Zanim podejmiemy decyzję o wyborze metodyki dobrze jest odpowiedzieć sobie na kilka pytań (poniżej pytań wyjaśnienie do nich):

1. Jak duży zespół posiadamy, i czy członkowie tego zespołu znają już (mają wdrożoną) jakąś metodykę tworzenia oprogramowania?
   Wielkość zespołu wpływa na konieczność stosowania bogatszej dokumentacji, wynika to z prostej zależności, że im mniejszy zespół tym informacje łatwiej i szybciej rozpowszechniamy, przy większym zespole musimy więcej dokumentować, by w razie czego każdy, kto tego potrzebuje mógł z niej skorzystać. Jednak nie możemy odebrać mylnego wrażenia, że jeśli pracujemy w dwie osoby, to nie mamy dokumentacji w ogóle. Dokumentacja jest potrzebna zawsze jak chociażby wspominana wielokrotnie poprzednio dokumentacja projektowa. Możemy pokusić się o zmniejszenie ilości dokumentacji – nazwijmy ją pośredniej – która ma znaczenie przy większych grupach projektowych, gdzie dużo trudniej jest wymieniać informację, albo nie ma możliwości zebrania wszystkich członków zespołu. Druga część pytania wynika z czystego pragmatyzmu, jeśli zespół zna jakaś metodykę, to może nie warto uczyć się nowej.
2. Z jak dużym projektem mamy do czynienia?
   Tu znów zależność zwykle jest prosta, im większy projekt, tym zwykle jego realizacja trwa dłużej, a co za tym idzie należy dokumentować więcej i dokładniej, by coś nie umknęło w trakcie realizacji.
3. Czy projekt, który będziemy realizowali jest bardzo krytyczny (lub obarczony dużym ryzykiem)?
   Znów w tym miejscu zależność jest niejako wprost, im projekt jest bardziej krytyczny, tym wymaga więcej uwagi, dokładniejszych testów oraz dobrej dokumentacji.
4. Czy klient wymaga od nas spełnienia jakiś dodatkowych norm lub wymogów (np. ISO)?
   W przypadku ostatniego pytania, odpowiedź już nie jest prosta, i nie mam możliwości dać jej teraz, gdyż zależy ona właśnie od wymogów stawianych przez klienta.

Ostatecznie możemy wysnuć jeden wniosek z powyższego, nie zależnie od tego jaką metodykę wybierzemy ważne jest dokumentowanie naszych poczynań, w takim stopniu, by mieć dostateczną wiedzę o projekcie, a jednoczenie nie przesadzić. Pamiętać musimy jednak, by nie doprowadzić do sytuacji, w której dokumentacja zaciemnia projekt. Można wskazać metodyki leżące jakby na przeciwnych biegunach, bo z jednej strony metodyka Document Driven Architecure (DDD) jest mocno zorientowana na dokument, a dla przykładu wspomniana już metodyka TDD koncentruje się na testach, które stanowią część dokumentacji.

Zdecydowanie w tym kursie będziemy koncertowali się wokół tzw. metodyk lekkich w szczególności Agile modelling, XP czy TDD, wynika to z faktu, iż właśnie metodyki lekkie są w centrum zainteresowania wśród firm tworzących oprogramowanie, jako mniej obciążone dokumentacją i pozwalające na szybsze osiąganie celu. Wybór metodyki wytwarzania oprogramowania determinuje tzw. cykl życia oprogramowania – zagadnienia te omówimy szerzej w module 5.

Organizacja zespołu projektowego

Skład i organizacja zespołu projektowego zależy od charakteru i wielkości projektu, zwykle jednak pod pojęciem zespołu projektowego rozumiemy zarówno pracowników firmy informatycznej jak i kluczowych użytkowników po stronie klienta. Jeśli mamy do czynienia z dużymi przedsięwzięciami informatycznymi powołuje się zwykle dwa byty:

- **Komitet sterujący** – w skład którego wchodzą decydenci z obu firm, a rola jego sprowadza się zwykle do nadzoru projektu (podejmowania kluczowych decyzji) i wyboru Kierownika Projektu.
- **Zespół wykonawczy** – w skład którego wchodzą pracownicy firmy informatycznej, oraz kluczowi użytkownicy. Zadaniem tego zespołu jest realizacja powierzonych zadań.

Ponieważ w warunkach szkoleniowych nie można łatwo odtworzyć rzeczywistej złożoności struktury zespołu projektowego, ograniczymy się do mniej sformalizowanej struktury (przykład takiej podamy w sekcji „Przykładowe rozwiązanie").

Przeprowadźmy jednak dyskusję od innej strony, w jaki sposób dobrać pracowników do realizacji zadania, oraz obecności jakich użytkowników należy się domagać, by zmniejszyć ryzyko niepowodzenia.

Przy doborze własnych pracowników do zespołu tworzącego oprogramowania, może okazać się bardzo przydatna wiedza pozyskana w trakcie fazy strategicznej. W tej pierwszej fazie, jak pisaliśmy, powinny zostać określone główne założenia funkcjonalne, jak i niefunkcjonalne, stąd też będziemy niejako pośrednio, znać wymagania wobec naszego zespołu, zatem dobór pracowników powinien opierać się głównie w oparciu o tę wiedzę. Wspomnimy poniżej o czym należy jeszcze pamiętać przy doborze pracowników, jednak tematu tego nie będziemy rozwijać:

- Umiejętność współpracy z innymi (umiejętność pracy w grupie).
- Kompetencje, które pokrywają się z planowanymi zadaniami.
- Doświadczenie i znajomość metodyki pracy.
- Otwartość na zmiany, elastyczność.

Dość oczywiste jest, że im większy zespół tym trudniej jednej osobie (Kierownikowi) ogarnąć zadania wykonywane przez poszczególnych pracowników. Stąd też, należy rozważyć podział zespołu na mniejsze grupy, kierując się przede wszystkim obszarem odpowiedzialności tych osób. Oznacza, to, że mając 30 osobowy zespół nie można zakładać, że właściwy podział to 3 grupy po 10 osób, być może będzie wśród nich 6 osób od architektury i systemów, 5 od bazy, 15 programistów i 4 osoby zajmujące się wsparciem. Podział powinien odzwierciedlać rzeczywisty podział kompetencji powierzonym kierownikom (liderom) tych grup.

Należy również pamiętać o tym, że na podział zespołu i wydzielenie obszarów odpowiedzialności może mieć wpływ na zakres realizowanego projektu, jak i jego wielkość. Co więcej, sam klient może niejako „niechcący" narzucać pewne podziały. Za przykład wystarczy wziąć realizację dużego sytemu kadrowo-płacowego. Z jednej strony moduły tego systemu (kadry i płace) są mocno ze sobą powiązane, a z drugiej, po ustalenie interfejsów pomiędzy nimi, bez problemu można je realizować w dwu zespołach, czasem tylko się komunikujących.

Kolejnym czynnikiem, który będzie miał wpływ na organizację zespołu jest przyjęta metodyka zarządzania projektem, inne role występują np. w PRINCE 2, inne w PMI, a jeszcze inne w metodykach lekkich, takich jak np. SCRUM. W tym miejscu należy zaznaczyć, że czym innym jest metodyka wytwarzana oprogramowania (zasygnalizowane w poprzednim punkcie), a zupełnie czym innym metodyka zarządzana projektem Nie należy mylić tych dwu rzeczy, a z drugiej strony pamiętać, że niektóre metodyki zarządzana projektem są zupełnie nieadekwatne do danej metodyki wytwarzania - tak często jest w przypadku mieszania metodyk lekkich z innymi.

Środowisko pracy

Większość klasycznej literatury dotyczącej inżynierii oprogramowania czy też zarządzania projektami nie odnosi się w sposób szczególny do stworzenia właściwego środowiska pracy dla zespołu projektowego. Takie odniesienie można znaleźć dopiero w tzw. metodykach lekkich czy też zwinnych (sporo na ten temat można znaleźć w ramach Agile Modelling). W tym punkcie nie

będziemy odnosić się do konkretnej literatury, zasygnalizujemy jedynie ten problem, mając świadomość jego wagi.

Zauważmy, że możemy zacząć od trywialnej obserwacji, iż praca nad projektem powinna odbywać się w sprzyjającym środowisku. Dalej, zauważmy, że zespół musi mieć poczucie swobody, komfortu i wygody w miejscu pracy. Dobrze, gdy w firmie jest miejsce, gdzie ludzie mogą się spotkać, podyskutować, przedstawić swoje pomysły lub pracować wspólnie nad nowymi. Ważnym elementem wyposażenia jest tablica, flip chart, stół. Za ich pomocą można zilustrować pomysł bądź przedyskutować projekt. Tablica powinna być umieszczona w miejscu ogólnie dostępnym dla wszystkich członków zespołu, dając w ten sposób możliwość wspólnej, bardziej efektywnej, pracy nad pomysłami. W takim pomieszczeniu powinny znajdować się materiały piśmiennicze w różnych kolorach (pozwala to na uwydatnianie kluczowych informacji)Tego typu rozwiązania to coraz częściej spotykany widok w firmach sektora IT. Pomieszczenie do dyskusji powinno być przestronne i widne, skłaniające do swobody i poruszania się oraz powinno być wyposażone w nośniki multimedialne Obecnie nie stanowi problemu utrwalenie pomysłów z tablicy za pomocą zdjęcia z aparatu komórkowego, ale te zdjęcia dobrze jest wrzucać w ogólnie dostępny katalog projektowy.

Osobną sprawą są inne przedmioty, które nie oddziałują bezpośrednio na projektowanie, ale poprawiają samopoczucie pracowników i tym samym na efekty pracy, dla przykładu warto zadbać o ekspres do kawy albo dystrybutor z wodą.

Środowisko oczywiście silnie zależy od budżetu firmy czy konkretnego projektu, jednak należy pamiętać, że czasami pieniądze wydane na pozornie zbędne rzeczy (jak kawa) mogą zwrócić się w postaci skróconego czasu realizacji projektu.

Wybór narzędzi i technologii

Wybór narządzi do tworzenia oprogramowania jak i użytej technologii jest ogromnie ważnym elementem, gdyż wpływa, na wiele obszarów np.:

1. Często wybór narzędzi i technologii determinuje docelowa platformę produkcyjną i na odwrót. Zatem jeśli klient oczekuje stworzenia systemu działającego na komputerach Apple na ich systemie operacyjnym, to mamy określone ograniczenia.
2. Wybór konkretnego rozwiązania może mieć ogromny wpływ na produktywność zespołu i to co najmniej z dwu przeciwstawnych przyczyn:
    a) z jednej strony w jednych narzędziach pisze się „szybciej” i „łatwiej”.
    b) z drugiej strony, jeśli zespół zna już jedno narzędzie, to wymiana na lepsze wcale nie musi przynieść efektu od razu – już w pierwszym projekcie (obowiązuje czas nauki).
3. Każdy wybór związany jest z kosztami, zwykle im lepsze narzędzie tym droższe. Zatem nie zawsze możemy wybierać z czego tylko chcemy.
4. Ostatecznie wybór narzędzia zależy również od wielkości projektu, nie ma sensu wprowadzać bardzo skomplikowanego środowiska przy małych projektach.

W tym miejscu nie będziemy dalej rozwijali tego tematu, gdyż poświęcamy więcej miejsca na omówienie narzędzi współczesnego zespołu programistów w osobnym module.

Podsumowanie

W tym rozdziale przedstawione zostały przywołane najważniejsze zagadnienia związane z tworzeniem oprogramowania, inicjowaniem projektu, organizacją zespołu projektowego, zasygnalizowano istotę tworzenia właściwego środowiska pracy i wyboru właściwych narzędzi projektowych. Większość poruszonych w nim zagadnień należy traktować jako przypomnienie znanych wcześniej faktów i w razie wątpliwości wiedzę należy poszerzyć we wskazanej literaturze.

Przykładowe rozwiązanie

Wróćmy do problemu zasygnalizowanego w podpunkcie Przykładowy problem. Masz zgraną paczkę koleżanek i kolegów, z którymi postanowiliście rozpocząć działalność programistyczną. Pozostaje podjąć kilka ważnych decyzji:

1. Jak podzielić się pracą – organizacja zespołu.
2. Jakie narzędzia i technologie wybrać?
3. Jak przygotować ofertę na rynek i do kogo ją skierować?
4. Jak przygotować środowisko pracy?
5. Jakie narzędzia wybrać?
6. Wiele, wiele innych…

Organizacja zespołu

Zebrałeś wszystkich i chcesz przeprowadzić ocenę kompetencji poszczególnych osób, możesz to zrobić za pomocą zwykłej listy cech i kompetencji, jakie można przypisać każdej osobie wraz z ich siłą (dla cech: słaba, średnia, silna, dla kompetencji: słabo, średnio, dobrze), w efekcie sporządziłeś, wraz z resztą przyszłego zespołu, dwie następujące tabele:

Osoba/Cecha	Przywództwo	Kreatywność	łatwość nawiązywania kontaktu	Systematyczność	
Marysia	Dobre	Dobra	Dobra	Dobra	
Ania					
Basia					
Marek					
Andrzej					
Władek					
Krzysztof					
Jurek					

Osoba/ Kompetencja	Programowanie	Projektowanie systemu	Projektowanie UI	Projektowania bazy danych	Administracja
Marysia					
Ania					
Basia					
Marek					
Andrzej					

Władek					
Krzysztof					
Jurek					

W oparciu o te tabele możecie wykreować podstawowe role w firmie, ten podział może pozwolić Ci wyznaczać odpowiednich członków do przyszłych projektów. Pamiętaj jednak, że w ramach firmy występują również działania operacyjne (w projekcie również) i ktoś musi te czynności wykonywać, zatem spróbujmy podzielić osoby najprościej jak się da na:

- Przywódców
- Programistów
- Wsparcie

Środowisko pracy

W tym miejscu nie będziemy nadmiernie rozwijać tego tematu, należy jedynie zadbać by zespół posiadał sprzęt komputerowy z wymaganym oprogramowanie oraz rozsądna ilość materiałów biurowych.

Wybór narzędzi i technologii

W tym miejscu zupełnie pomijamy ten wybór, będzie to omówione w osobnym module.

Przygotowanie oferty usług

Firma LENIWIEC Sp.J. specjalizuje się w kompleksowej obsłudze informatycznej małych i średnich firm i instytucji na rynku ogólnopolskim w szczególności miasta Warszawa i Łódź. Pod opieką serwisową i informatyczną mamy wiele czołowych firm rynku woj. łódzkiego i mazowieckiego z różnej branży przemysłowej, tekstylnej czy handlowej. Świadczymy usługi z zakresu tworzenia i wdrażania systemów informatycznych oraz adaptacji istniejących już rozwiązań do potrzeb klientów. Nasze główne usługi:

- tworzenie oprogramowania na zamówienie klienta,
- administrowaniem serwerów i systemów serwerowych,
- sprzedaż oraz wdrożenie systemów i aplikacji,
- pomoc merytoryczna, szkolenia, instruktarze,
- konsultacje i doradztwo,
- tworzenie dodatkowej funkcjonalności, moduły dla Microsoft Dynamics,
- programowanie aplikacji w architekturze klient-serwer i .NET,
- projektowanie relacyjnych baz danych MS SQL,
- integracje systemów informatycznych.

Powyżej zaprezentowaliśmy przykładową ofertę firmy.

Porady praktyczne

Musisz pamiętać, że cokolwiek, z tego co tu przeczytałeś jest niejako „wierzchołkiem góry lodowej”, wbrew pozorom pomimo, iż te zagadnienia występują na samym początku większości kursów inżynierii oprogramowania, są bardzo złożone i trudne. Ich pierwszeństwo w świecie rzeczywistym wynika z faktu, iż wpierw musisz mieć zespół zanim przejdziemy do realizacji zadań dla klienta. Z kolei w trakcie wykładów omawia się je na początku by pokazać złożoność procesów wytwórczy i dopiero stopniowo uszczegóławia się każdy z elementów. Jednak, gdy zapoznasz się z całym kursem, wróć do tego modułu, przeczytaj literaturę dodatkową i dopiero wtedy w praktyce załóż firmę.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- rozumiesz jakie fazy składają się na cały cykl życia oprogramowania, jakie role można wyróżnić wśród członków zespołu projektowego, jaka jest rola środowiska pracy i narzędzi projektowych, rolę inżynierii oprogramowania jako źródła różnego rodzaju dobrych praktyk w zakresie tworzenia oprogramowania,
- umiesz zidentyfikować role członków zespołu, zorganizować zespół projektowy, stworzyć odpowiednie warunki pracy,

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. Andrzej Jaśkiewicz, *Inżynieria oprogramowania,* Helion 1997

   W książce Autor omawia podstawowe zagadnienia związane z procesem wytwarzania oprogramowania.

2. Roger S. Pressman (adapted by Darrel Ince), *Software Engineering – A Practitioner' Approach, European Adaptation*, McGraw-Hill Publishing Company 2000

   W książce Autor przedstawia zagadnienie tworzenia oprogramowania z punktu widzenia praktyka, co stanowi bardzo wartościowe ujęcie tematu.

3. Ian Sommerville, *Inżynieria oprogramowania*, WNT 2005

   W książce Autor omawia proces wytwarzania wielkich systemów informatycznych. Autor, sam będąc światowej sławy specjalista w tej dziedzinie, przedstawia kolejne etapy tego procesu.

4. Susan Snedaker, *Zarządzanie projektami IT w małym palcu*, Helion 2007

   Książka ta stanowi niejako kompendium wiedzy na temat zarządzania projektami IT.

5. Ścibór Sobieski, *Inżynieria oprogramowania*, skrypty w wersji elektronicznej http://www.scibor.org/wp-content/uploads/2007/01/si.pdf

   W tym skrypcie Autor omawia ogólne zagadnienia związane z inżynierią oprogramowania zwracając szczególną uwagę na aspekt praktyczny.

6. Ścibór Sobieski, *Zarządzanie projektem informatycznym*, skrypt w wersji elektronicznej http://www.scibor.org/wp-content/uploads/2007/03/skrypt_zpi.pdf

   W tym skrypcie Autor omawia ogólne zagadnienia związane z zarządzaniem projektami informatycznymi zwracając szczególną uwagę na aspekt praktyczny.

7. Robert K. Wysocki, Rudd McGary, *Efektywne zarządzanie projektami*, Helion 2005

   W książce Autorzy starali się przybliżyć nowoczesne metody zarządzania projektami, stosowane od kilku lat.

8. Zbigniew Szyjewski, Metodyki zarządzania projektami informatycznymi

   W książce Autor porusza problematykę zarządzania projektami informatycznymi.

8. http://en.wikipedia.org/wiki/Test-driven_development

   Na tej stronie znajduje się szkic metody Test Driven Development (TDD).

9. PMI – A Guide to the Project Management Body of Knowledge (PMBOK GUIDE)

Książka ta omawia metodykę PMI.

10. PRINCE2™ - Skuteczne Zarządzanie Projektami, Drugie wydanie polskie Crown Copyright 2010

Książka ta omawia metodykę PRINCE2™.

Laboratorium podstawowe

Jesteś jednym z właścicieli niewielkiej firmy o nazwie LENIWIEC SP. J., do niedawna Wasza spółka zajmowała się handlem urządzeniami komputerowymi i hostingiem stron WWW na niewielką skalę. Teraz okazała się, że jeden z Waszych stałych klientów chce by napisać dla niego pewną aplikację. Ze wstępnych rozmów wynika, że stworzenie tej aplikacji będzie wymagało zespołu złożonego z 4-5 osób o kompetencjach z zakresu: programowania, baz danych, tworzenia grafiki i prawdopodobnie tworzenia stron WWW – prawdopodobnie, gdyż klient nie podjął ostatecznej decyzji w tym zakresie. Sam projekt będzie trwał 3-4 miesiące. Żaden z pozostałych dwóch wspólników nie ma talentów organizacyjnych i do tego nie ma pojęcia o programowaniu, więc to właśnie Tobie powierzono zadanie organizacji zespołu projektowego. Twoim zadaniem zatem będzie:

- Przeprowadzenie rozmów kwalifikacyjnych z kandydatami do pracy.
- Identyfikacja mocnych i słabych cech oraz kompetencji każdego z kandydatów.
- Wybór 4-5 członków zespołu.
- Przypisanie im podstawowych ról.

Zadanie	**Tok postępowania**
1. Przygotowanie ankiety oceniającej kandydata	• Za pomocą dowolnego edytora tekstów przygotuj szablon ankiety oceny kandydata do pracy. • W szablonie umieść pytania, które będą odzwierciedlały potrzeby Twojej firmy. • Dodaj do ankiety pytania z zakresu zbliżającego się projektu. Zauważ, że projekt ten ma nie do końca sprecyzowane wymagania. • Ankietę przygotuj sam – bez pomocy koleżanek i kolegów z grupy.
2. Przeprowadzenie rozmowy kwalifikacyjnej	• Przeprowadź rozmowę z każdym z potencjalnych kandydatów (prowadzący powinien podzielić grupę laboratoryjną na podgrupy 4 -5 osobowe, Twoimi kandydatami są twoje koleżanki i koledzy z podgrupy). • Daj kandydatowi do wypełnienia ankietę. • Na podstawie ankiety i rozmowy stwórz tabelę mocnych i słabych cech kandydata.
3. Wybór członków zespołu i przypisanie im ról	• Ponieważ w warunkach laboratoryjnych jesteście już podzieleni na grupy 4-5 osobowe, to w tym punkcie nie będziecie dokonywali selekcji kandydatów, a jedynie przypiszcie im role, jakie uważacie za właściwe.
4. Weryfikacja ocen i tworzenie zespołu	• Każdy z Was powinien to ćwiczenie wykonać oddzielnie. • Po jego wykonaniu porównajcie Wasze wyniki, poddajcie je pod dyskusję. • Stwórzcie na tej podstawie wstępny podział ról.

Laboratorium rozszerzone

Twoim zadaniem będzie stworzenie przykładowych opisów stanowisk pracy oraz stworzenie testów kwalifikacyjnych na te stanowiska (stanowiska jak w poprzednim ćwiczeniu). Poniżej znajdują się przykładowe opis i test.

Opis stanowiska pracy

Właściwie opisane stanowisko pracy jest punktem wyjścia do realizacji strategii zarządzania zasobami ludzkimi w firmie. Stanowi odniesienie w trakcie projektów rekrutacyjnych, jest kluczowym elementem sporządzania okresowych ocen pracowniczych, standaryzuje procedury administracyjno – kadrowe.

Zakres obowiązków

- Kierowanie projektami informatycznymi o średniej wielkości / znaczeniu,
- Kontrola poszczególnych etapów przebiegu projektu i zapewnienie koordynacji zespołów w projekcie,
- Zarządzanie poszczególnymi elementami uzyskiwanymi w wyniku projektu oraz relacjami między nimi,
- Zarządzanie ryzykiem w projekcie,
- Ustalanie oraz nadzór nad prawidłowa realizacją budżetu projektu,
- Zarządzanie pracą zespołu projektowego,
- Raportuje status prac projektowych.

Wymagania ogóle

- odpowiedzialność,
- analityczny umysł, umiejętność logicznego myślenia,
- dbałość o szczegóły, rzetelność, dokładność,
- dobra organizacja pracy własnej i zespołu,
- komunikatywność, kreatywność, samodzielność, kultura osobista,
- umiejętność pracy samodzielnej jak i w zespole,
- doświadczenie w prowadzeniu projektów informatycznych,
- umiejętność interpretacji zapisów procedur,
- odporność na stres (umiejętność pracy pod presją czasu).,
- kreatywność i innowacyjność.

Wymagania specjalistyczne

- znajomość metodologii prowadzenia projektów informatycznych (PMI lub Prince – preferowane obie),
- udokumentowane doświadczenie w prowadzeniu projektów,
- znajomość narzędzia i metody zarządzania projektem informatycznym,
- praca w charakterze kierownika przy minimum jednym projekcie informatycznym, wielkość projektu: zespół min. 10 osób.

Test rekrutacyjny

Poniżej przedstawiamy przykładowe pytania, przy czym dla uproszczenia w jednej formie osobowej.

1. Czy masz jakieś doświadczenia związane z pracą w firmie z sektora IT?
    - Tak , pracuję w takiej.
    - Tak, pracowałem w takiej firmie studiując.

- Nie, dopiero zaczynam pracę w tym sektorze.

2. Na jakim poziomie znasz język angielski?

- Bardzo dobrze.
- Dobrze.
- Podstawy.

3. Czy pracowałeś kiedyś w projekcie?

- Tak, jako członek zespołu projektowego.
- Tak, kierowałem takim projektem.
- Nie.

4. Nawiązywanie kontaktów z innymi ludźmi to:

- Twoja słabsza strona.
- To, co najbardziej lubisz.
- Jak trzeba, to nawiązuję, ale wolę konkretną pracę.

5. Czy jesteś gotów na częste wyjazdy?

- Wolę tego uniknąć.
- Tak, ale nie dłużej niż na kilka dni.
- Tak, niezależnie od długości wyjazdu.

6. Czujesz się przede wszystkim?

- Pomysłodawcą, inicjatorem.
- Realizatorem, wykonawcą.
- Planistą, organizatorem.

7. Czy miałeś w dotychczasowej pracy kontakt z budżetem (firmy , działu, projektu)?

- Nie.
- Tak, odpowiadałem za jego realizację.
- Tak, odpowiadałem za jego zaplanowanie i realizację.

8. Czy jesteś gotowy na pracę w nienormowanym czasie?

- Jeśli będę robić coś pożytecznego, mogę nie patrzeć na zegarek i dni tygodnia.
- Rozumiem, że można raz czy dwa zostać po godzinach, czy wyjechać służbowo w sobotę, ale nie może stać się to regułą.
- Nie wyobrażam sobie poświęcania czasu wolnego na sprawy zawodowe.

9. Na pracę w sektorze IT decydujesz się, ponieważ:

- Już w nim pracuję.
- Czuję potrzebę zrobienia czegoś dla biznesu i firmy.
- Ważne jest dla mnie, żeby robić interesujące mnie rzeczy, nieważne w którym sektorze.

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 2

Wersja 1

Wstępna analiza wymagań

Spis treści

Informacje o module

Opis modułu

W tym module znajdziesz wprowadzenie do problematyki pozyskiwania wstępnych wymagań klienta, oraz ich analizy. Oba te procesy mają na celu określenie zakresu prac przyszłego projektu i umożliwienie oszacowania jego pracochłonności, kosztów i czasu wykonania. Znajduje się tu omówienie zagadnień inżynierii systemowej, inżynierii wymagań, analizy wymagań i ich wstępnej specyfikacji.

Cel modułu

Celem modułu jest wprowadzenie w zagadnienia wstępnej analizy wymagań klienta.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- wiedział, na co należy zwracać uwagę w trakcie pozyskiwania wymagań od klienta,
- wiedział, jakie czynności trzeba wykonać by pozyskać wymagania od klienta oraz by jest przeanalizować i sprecyzować,
- potrafił stworzyć wstępny zestaw pytań w celu pozyskania wymagań,
- potrafił przeanalizować odpowiedz klienta i doprecyzować niejasności,
- potrafił stworzyć spis wstępnych wymagań,
- rozumiał, jakie problemy leżą u podstaw inżynierii systemowej oraz inżynierii wymagań,
- rozumiał, z jakimi problemami spotykamy się w trakcie pozyskiwania wymagań i do jakich niejednoznaczności to może doprowadzić.

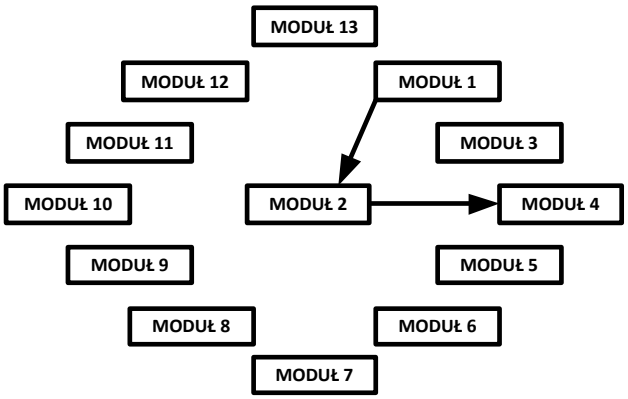
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawowe pojęcia inżynierii oprogramowania ze szczególnym uwzględnieniem inżynierii wymagań, ich analizy i specyfikacji,
- rozumieć problematykę w/w zagadnień.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w module



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Posiadamy niewielki zespół programistyczny – powiedzmy 3-10 osób – chcielibyśmy zrealizować projekt dla pewnej firmy. Zanim jednak przystąpimy do realizacji, chcemy przeprowadzić wstępny wywiad z klientem, w celu pozyskania ogólnych wymagań wobec systemu. Musimy również zapoznać się z posiadaną infrastukturą IT, by wiedzieć jakie rozwiązanie jest właściwe, lub mówiąc inaczej, by doradzić klientowi rozwiązanie, które dobrze wpasowuje się w jego zasoby, przez co uzyskujemy efekt maksymalizacji zysku przy minimalizacji kosztów.

Pamiętamy, że wstępna analiza wymagań klienta musi zawierać kluczowe informacje o funkcjonalności przyszłego systemu i to na tyle dokładne, byśmy na tej podstawie mogli możliwie dokładnie powiedzieć klientowi ile czasu zajmie nam stworznie systemu oraz jakie to pociągnie za sobą koszty. Stąd też przeprowadzenie wywiadu i opracowanie wstepnej analizy jest podstawą np. do umowy, jaką chcemy podpisać z klientem.

Wiemy, że na takie rozmowy musimy wydelegować dwie, góra trzy osoby, które z jednej strony dobrze orientują się w technologiach IT, a z drugiej łatwo nawiązują kontakt, umieją myśleć analitycznie i – co bardzo istotne – umieją sporządzać notatki. Oczywiście nie każda osoba z naszej delegacji musi posiadać wszystkie te cechy.

Pamiętamy, iż na studiach mówiono nam, że takie wstępne rozmowy (wyjątkowo rzadko jest to jedna rozmowa) powinny zakończyć się mniej lub bardziej formalną notatką ze spotkania. Jednak jak należy prowadzić taką rozmowę? O co pytać klienta? A może należałoby pójść do niego z pewnym planem i odpytać jego pracowników w sposób metodyczny? Przecież mieliśmy zajęcia z Inżynierii Oprogramowania, jak to się stało, że nie pamiętam tak podstawowej rzeczy jak pozyskiwanie wymagań od klienta? Może dlatego, że nigdy nie mieliśmy praktycznych zajeć, które pozowoliłyby na rzeczywiste uczestnictwo choćby w mini projekcie, a może dlatego, że znów uczyliśmy się tylko by zaliczyć? No cóż… teraz należy wrócić do tej wiedzy, podanej choćby teoretycznie i spróbować przygotować się do spotkania z klientem.

Podstawy teoretyczne

Inżynieria systemowa

Przypomnijmy, że w inżynierii oprogramowania mówiliśmy o różnych podejściach do procesu tworzenia, zwanych cyklami życia oprogramowania. W ramach tego kursu będziemy skłaniali się do tzw. metodyk lekkich. Nie zmienia to jednak faktu, iż musimy przypomnieć sobie jak wyglądają ogólne fazy wytwarzania oprogramowania, gdyż w mniej jawnej formie występują one zawsze – niezależnie od przyjętej metodyki – najwyżej są one inaczej nazwane, lub wymieszane pomiędzy sobą. Przypomnijmy, że w ramach modelu kaskadowego wyróżniamy następujące fazy:

1. Analiza – faza, w której próbujemy odpowiedziec „co” jest do zrobienia, to tutaj pozyskujemy wymagania.
2. Projekt – faza, w której odpowiadamy na pytanie „jak” zrealizować, to czego klient od nas oczekuje.
3. Kodowanie – zamiana projektu na konkretną realizację przy użyciu ustalonych technologii.
4. Testowanie – weryfikacja i walidacja wytworzonego oprogramowania.

W przypadku metody lekkich nie stosujemy takiego sztywnego podziału, co wiecej faza analizy i projektowania przenika się wzajemnie. Co ciekawe również w przypadku innych metody, z latami dostrzeżono, że właśnie faza analizy i projektowania stanowi do pewnego stopnia całość – zresztą w ramach klasycznego podziału również łączono je w tzw. inżynierię systemową. To właśnie te dwie fazy będą dla nas podstawą do dalszych rozważań, to na ich podstawie zaproponujemy najważniejsze punkty, jakie należy uwzględnić przy wstępnej analizie wymagań klienta. Przejdziemy

teraz do przypomnienia wiedzy o pozyskiwaniu wymagań (będącej elementem fazy analizy) zwanej też inżynierią wymagań.

Inżynieria wymagań

Wynikiem procesu inżynierii wymagań jest specyfikacja systemu zawierająca informację o różnym poziomie szczegółowości, na podstawie której można przejść do etapu projektowania. Niewątpliwie największym problemem dla inżyniera systemowego jest odpowiedź na pytanie, w jaki sposób można się upewnić, że to co napisał jest tym co klient oczekuje? W rzeczywistych dużych projektach kompletna odpowiedź na to pytanie jest trudna a często wręcz niemożliwa na etapie modelowania systemu, czy też pozyskiwania wymagań.

Inżynieria wymagań jest procesem, który ma zaradzić tym trudnością i pomóc w odpowiedzi na postawione pytanie. Zawiera ona procedury uzyskania i dokumentowania informacji o potrzebach, życzeniach i żądaniach klienta. Dokumentuje je wszystkie, zawiera ich analizę, oraz rejestruje procesy negocjacyjne oraz ich wynik. Dostarcza informacji o jednoznacznych rozwiązaniach, weryfikuje całą specyfikację, oraz na sam koniec zarządza wymaganiami, które są stawiane wobec systemu. Proces ten możemy zatem podzielić na sześć kolejnych etapów:

1. **wydobywanie wymagań**,
2. **analiza i negocjacja wymagań**,
3. **tworzenie specyfikacji wymagań**,
4. **modelowanie systemu**,
5. **weryfikacja wymagań**,
6. **zarządzanie wymaganiami**.

Omówimy teraz pokrótce te etapy.

Wydobywanie wymagań

W procesie tym staramy się pozyskać wymagania od klienta poprzez zadawanie coraz to bardziej szczegółowych pytań (stąd też określenie wydobywanie). Sommerville i Sawyer stworzyli szczegółowy przewodnik, który można wykorzystać do uzyskania wymagań od klienta, upraszczając można go streścić w następujących punktach:

1. należy oszacować biznesowe i techniczne możliwości dla tworzonego systemu,
2. należy zidentyfikować ludzi, którzy pomogą określić wymagania oraz zrozumieć organizacyjne podłoże tych wymagań,
3. należy określić środowisko techniczne, tj. architekturą, system operacyjny, wymagania sieciowe i teleinformatyczne, w którym ma docelowo pracować system,
4. należy zidentyfikować wzajemne więzy, które ograniczają funkcjonalność czy wydajność tworzonego systemu,
5. należy zdefiniować jedną lub więcej metod wydobywania wymagań, mogą być to spotkania, wywiady, grupowe narady, czy nawet wspólne wyjazdy,
6. należy domagać się udziału jak największej ilości osób, by wymagania były naświetlone z jak największej ilości punktów widzenia,
7. trzeba znaleźć niejednoznaczności w wymaganiach, staną się one kandydatami dla prototypowania,
8. należy stworzyć scenariusze użycia, które pozwolą klientowi lepiej określić kluczowe wymagania.

Choć powyższe punkty będą bardzo pomocne, to jednak musimy pamiętać o trudnościach realizacji powyższych uwag. Dostrzegli to również dwaj autorzy Christel i Kang i zgrupowali je w trzy poniższe punkty:

1. **Problemy zakresu** – zasięg systemu może być wadliwie zdefiniowany, klient może dodać zbędne techniczne szczegóły, które mogą powodować niepotrzebne zamieszanie i rozmydlać prawdziwy cel.
2. **Problemy zrozumienia** – klient nie jest do końca pewny czego tak naprawdę potrzebuje, zwykle ma nikłą znajomość ograniczeń środowiska, w jakim ma pracować system. Nie ma pełnego obrazu całości systemu, zwykle ma trudności komunikacji z informatykiem. Klient często pomija pewne informacje, które uznaje za „oczywiste". Wyspecyfikowane właściwości mogą pozostawać w konflikcie z potrzebami innych użytkowników systemu. Wyspecyfikowane wymagania są niejasne, dwuznaczne oraz trudne do przetestowania.
3. **Problemy zmienności** – wymagania zmieniają się w czasie.

Analiza i negocjacja wymagań

Analiza wymagań opiera się na informacji uzyskanej od klienta i składa się z kilku etapów, które można scharakteryzować następująco:

- **Podział** – wymagania są dzielone na kategorie w celu otrzymania kilku zbiorów abstrakcji.
- **Powiązania** – określane i badane są więzy i powiązania pomiędzy poszczególnymi wymaganiami.
- **Spójność** – sprawdzana jest spójność wymagań, identyfikowane są braki w ich określeniu czy dwuznaczności.
- **Ważność** – ustala się priorytet dla wszystkich wymagań w oparciu o potrzeby klienta.

Pressman w swojej publikacji wskazuje, iż w procesie analizy należy zadać następujące pytania i uzyskać na nie odpowiedzi:

1. Czy każde wymaganie jest zgodne z całością cech systemu?
2. Czy każde wymaganie jest wyspecyfikowane na właściwym poziomie abstrakcji? Jeżli tak to, czy wymaganie nie wprowadzają niepotrzebnych technicznych szczegółów?
3. Czy dane wymaganie jest naprawdę niezbędne, czy też jest tylko dodatk bez którego system może działać?
4. Czy każde wymaganie ma wyznaczone granice i jest jednoznaczne?
5. Czy każde wymaganie ma atrybuty? Jeżeli tak to, czy ich źródło zostało opisane?
6. Czy wymagania nie są w konflikcie z innymi?
7. Czy każde wymaganie jest wykonalne w środowisku w którym będzie pracował system?
8. Czy każde wymaganie będzie poddawało się testom i czy da się zaimplementować?

W rezultacie analizy powienien powstać uszczegółowiony spis wymagań funkcjonalnych i niefunkcjonalnych będący podstawą do finalnej specyfikacji wymagań, która uzupełniona o modele systemu, poddana weryfikacji będzie stanowić pełny opis przyszłego systemu.

Specyfikacja wymagań

Specyfikacja wymagań jest dokumentem, w który opisujemy szczegółowo docelowy system. Dokument ten zwany jest też czasem specyfikacją funkcjonalną, musi być bardzo precyzyjny, stąd też zwykle narzuca się pewną jego strukturę. Mówimy też o tym dokumencie jako o pomoście pomiędzy zamawiającym a wykonawcą. Forma, a raczej struktura dokumentu i sposób jego zapisu jest zależny od wykonawcy, w dużej mierze zależy to od podejścia, jakie firma preferuje do rozwiązania tego problemu (przyjęta metodologia wytwarzania oprogramowania). Pamiętajmy, że wielu autorów sugeruje, by do opisu tego dokumentu używać pewnego stadnardowego szablonu, który poprzez swoje uporządkowanie byłby czytelniejszy. W dokumencie tym powinno się określić pewne dodatkowe rzeczy, które we wcześniejszych punktach nie występowały, mianowicie:

1. Użytkownik powinien dostarczyć informacje potrzebne do zdefiniowania interfejsów (pliki, WebService, inne), które będą używane przez inne programy.
2. Każdy z typów takich interfejsów powinien mieć opisane narzędzie, które będzie go używać.

3. Użytkownik powinien określić zakres importu i eksportu danych do i z aplikacji.
4. Użytkownik powinien określić zasady współpracy z innymi programami, jak np. konieczność wykorzystania mechanizmu OLE.

Dokument ten, według Sommerville'a, powinien zawierać następujące rozdziały:

1. **Wprowadzenie** – zawiera cele systemu, krótko opisuje jego funkcje oraz to jak ma współpracować z resztą systemu.
2. **Słownik** – definicja wszystkich technicznych i specyficznych terminów użytych w dokumencie. Pamiętajmy, że nie powinniśmy czynić żadnych założeń o wiedzy czy doświadczeniu czytających ten dokument.
3. **Modele systemu** – opis modeli systemu, które ilustrują relacje pomiędzy jego komponentami i jego środowiskiem. Może zawierać modele obiektów, przepływu danych oraz modele semantyczne danych.
4. **Definicja wymagań funkcjonalnych** – opis w języku naturalnym wszystkich usług, która ma wykonywać program. Opis ten ma być zrozumiały dla klienta, powinniśmy posługiwać się diagramami, schematami i innymi sposobami wizualizacji funkcji mających na celu jak najczytelniejsze przedstawienie ich czytającemu.
5. **Definicja wymagań nie funkcjonalnych** – spis ograniczeń nałożonych na system oraz ich odniesienie do wymagań funkcjonalnych. Możemu tu zawarzeć opis konkretnej reprezentacji danych, określenie czasu odpowiedzi, wymagania pamięciowe, obciążenie procesora itp.
6. **Ewolucja systemu** – opis ogólnych założeń, co do sposobu reakcji systemu na zmiany środowiska sprzętowo-systemowego, oraz na rządanie zmiany od klienta.
7. **Specyfikacja wymagań** – dokładniejszy opis wymagań funkcjonalnych, jeżli istnieje interakcja z wymaganiami nie funkcjonalnymi, to również musi się znaleźć tu opis tego faktu. Jest to miejsce na wytyczne interfejsu użytkownika, oraz innych detali produktu.
8. **Wymagania dla bazy danych** – logiczna organizacja danych wykorzystywanych przez system oraz ich wzajemne oddziaływanie, opis technik modelowania encji i ich relacji.
9. **Sprzęt** – opis sprzętu, o ile system ma być zainstalowany na specyficznej maszynie. W przypadku produktu na popularne komputery, określa się wymagania minimalne i zalecane.
10. **Indeks** – indeks całego dokumentu, ma za zadanie ułatwić poruszanie się po nim.

Można dodać, że zwykle opis wymagań funkcjonalnych (funkcji) powinien zawierać następujące punkty:

- Opis – krótki opis funkcjonalności.
- Wejście – definicja danych wejściowych i ich ewentualnych ograniczeń.
- Wyjście – definicja zwracanych rezultatów.
- Efekty uboczne – określenie dodatkowych czynności, np. interakcji z innymi funkcjami.

Modelowanie systemu

Modelowanie systemu jest podstawowym narzędziem w inżynierii wymagąń, za jego pomocą tworzymy modele procesów, które:

- defniują procesy zaspokajające potrzeby użytkownika,
- opisują zachowania tych procesów oraz przyczyny tych zachowań,
- deniuje zewnętrzne i wewnętrzne wejścia dla fragmentów systemu (modułów),
- reprezentują wszelkie powiązania, zarówno wewnętrzne jak i zewnętrzne, by lepiej zobrazować całość systemu.

W literaturze można znaleźć następujące czynniki, na które musimy zwrócić uwagę, by prawidłowo zamodelować przyszły system:

1. Założenia – pozwolą nam ograniczyć ilość możliwości, co pozwoli by model był odzwierciedlenim rzeczywistości w takim wymiarze, w jakim jest to konieczne.
2. Uproszczenia – pozwolą zbudować model w rozsądnym czasie.
3. Ograniczenia – pozwalą wyznaczać granice dla systemu, w szczególności granice technologiczne czy też ramy czasowe.
4. Więzy – są dla nas, informują o zależnościach i środowisku, w którym powstaje system.
5. Preferencje – są dla nas źródłem informacji o upodobaniach dotyczących architektury, funkcji, technologii, czy też sposoby działania przyszłego systemu.

Mając w głowie powyższe czynniki, inżynier systemowy próbuje wynegocjować rozwiązanie obustronnie możliwe do zaakceptowania. Niestety bywa też często, że mimo najszczerszych chęci obu stron, system legnie w gruzach już przy próbie specyfikacji wymagań, gdyż np. okaże się on zbyt kosztowny (dlatego też modelowanie jest tak bardzo użyteczne w fazie wstępnej analizy, o czym będziemy pisać dalej).

Zawsze przy analizie wymagań należy brać pod uwagę charakter działań zamawiającego w zakresie przetwarzanych przez niego informacji. Często ta właśnie analiza nazywana jest inżynierią procesu biznesowego, ma ona na celu nie tylko zrozumienie tego, w jaki sposób jest przetwarzana informacja u klienta, ale też doradzenie, jak tę informację można przetwarzać efektywnie. W przypadku wielu dużych systemów, które już istnieją na rynku i są sprawdzone, dochodzi do sytuacji, w której firma wdrażająca system zmienia obieg dokumentów u klienta na lepiej dostosowany do jego charakteru pracy i jednocześnie dający się zaimplementować w ich systemie.

Weryfikacja wymagań

Weryfikacja wymagań sprowadza się do czynności mających na celu pokazanie, że wymagania stawiane wobec systemu są faktycznie tym, czego klient oczekuje. Należy podkreślić dużą rangę tych działań właśnie na tym etapie projektu, zauważmy, że jakikolwiek błąd popełniony na etapie ustalania wymagań będzie ciągnąc się aż do końca realizacji systemu. Im później zostanie stwierdzony i usunięty tym większe zostaną poniesione nakłady na jego usunięcie. Należy zdawać sobie sprawę, że koszt takiego błędu narasta zdecydowanie szybciej niż liniowo wraz z postępem prac. Usunięcie takiego problemu na etapie kodowania może być już nawet kilkadziesiąt razy droższe, niż na etapie specyfikacji wymagań. Ponadto najlepsze efekty uzyskuje się, gdy proces weryfikacji jest niejako prowadzony w tle pozostałych działań inżynierii systemowej, nie zaś jako działanie na jej samym końcu – tak właśnie jest w metodykach lekkich. Oczywiście powodem takiego postępowania jest, jak to zasygnalizowano wcześniej, unikanie niepotrzebnych kosztów.

W celu przeprowadzenia weryfikacji podejmuje się działąnia w następujących obszarach:

- **Zakres problemu** – użytkownik określił funkcjonalność wobec powstającego systemu, lecz na etapie analizy tej funkcjonalności może się okazać (i zwykle tak bywa), że potrzeba jest więcej funkcji niż tylko te określone.
- **Spójność** – w przypadku dużych systemów, należy sprawdzić współzależności pomiędzy poszczególnymi jego częściami, które często były specyfikowane przez różnych doradców (ekspertów, przyszłych użytkowników, itd.). Może się okazać, że musimy w pewnym zakresie wypracować kompromis pomiędzy niektórymi funkcjami. Oczywiście żadna z funkcji nie może naruszać integralności innej, czy też całości systemu.
- **Kompletność** – opis systemu powinien zawierać wszystkie funkcje oraz więzy określone przez użytkownika.
- **Realizm** – żadne z wymagań nie może być nie realizowalne, w szczególności nie można dopuszczać rozwiązań „na zapas”, co do których niebardzo wiadomo jak mają być wykonane.

Pamiętajmy, że weryfikacja wymagań wymaga:

1. Rozmów prowadzonych „na bieżąco” z klientem, w celu upewnienia się, że rozumienie materii zagadnienia jest prawidłowe.
2. Ci¡głego informowania klienta o naszym (technicznym) spojrzeniu na zagadnienie, ułatwi to klientowi zrozumienie dokumentacji.
3. Nieformalnych spotkań z przyszłymi użytkownikami, celem pozyskania pewnych informacji o przyzwyczajeniach czy też spostrzeżeniach na temat ich pracy.
4. Zapoznania się i przeczytaniu przez obie strony (zarówno wykonawcę jak i klienta) wszystkich dokumentów zawierających poczynione ustalenia.

Mimo, iż jak to wcześniej zauważyliśmy, dokumentacja wymagań użytkownika powinna być napisana w języku naturalnym z naciskiem położonym na taki sposób opisu, by klient był w stanie bez problemu ja przeczytać i zrozumieć, to jednak wymaga się od niej dużej precyzji w formułowania opisów. Celem takiej konstrukcji dokumentu ma być ułatwienie etapu weryfikacji, który powinien umożliwić odnalezienie wszelkich niespójności i niejednoznaczności zawartych w dokumentach.

Zarządzanie wymaganiami

Zarządzanie wymaganiami, często też postrzegane jako ewolucja wymagań, jest procesem mającym na celu kontrolę nad wszelkimi zmianami w oczekiwaniach klienta. Pamiętajmy, że chodź wydaje się, że użytkownik wie czego chce i nie będzie zmieniać, to jednak jest to pozorne i ulotne. Możemy wyróżnić kilka przykładowych powodów, dla których może nastąpić zmiana (ewolucja) wymagań, przy czym część z nich wynika z faktu, że analiza dużego projektu może trwać nawet kilka lat i w tym czasie zmieniają się pewne warunki. Wymieńmy kilka możliwych przyczyn zmiany:

1. Mogą ulec zmianie warunki zewnętrzne, np. zmiany legislacyjne w danym państwie.
2. Mogą ulec zmianie warunki wewnętrzne, np. zmieni się polityka jakości firmy, lub też firma właśnie taką politykę wdrożyła.
3. W czasie tworzenie aplikacji, pogłębia się zrozumienie wymagań klienta, co może być przyczyną zmian pierwotnych założeń.
4. Zmieniły się osoby decydujące po stronie klienta, zatem mogła też ulec zmianie wizja systemu.
5. Klient stwierdził błąd we wcześniejszych wymaganiach, który musi być naprawiony.
6. Zmieniło się otoczenie systemu, w tym system operacyjny, lub oprogramowanie stowarzyszone, jak baza danych czy inne produkty.
7. Zmienił się sprzęt, mogło to doprowadzić nawet do konieczności przeniesienia serwera aplikacji na inna platformę sprzętowo-systemową.

Przy czym zauważmy, że zmiany mogą następować zarówno na w zakresie wymagań funkcjonalnych (punkty 1-3) jak i nie funkcjonalnych (punkty 5-6), zaś punkt 4 będzie mógł generować zmiany w obu tych kategoriach zależnie od jego charakteru.

Wstępna analiza wymagań

Spróbujmy teraz zastanowić się, jak powyższe informacje możemy wykorzystać na etapie analizy wstępnej. Stąd też zacznijmy od uściślenia czego oczekujemy od procesu takiej wstępnej analizy, zauważmy, że:

1. Analiza wstępna musi być przeprowadzona na tyle precyzyjnie, byśmy na jej podstawie umieli oszacować wstępny harmonogram realizacji i koszty projektu.
2. Analiza wstępna musi być na tyle ogólna, by nie rodzić niepotrzebnej straty czasu i kosztów.
3. Analiza wstępna powinna pozwolić oszacować biznesowe i technologiczne możliwości realizacji przedsięwzięcia.

Zauważmy, że analiza wstępna poprzedza jakiekolwiek dalsze działania, co więcej jest przeprowadzana często przed podpisaniem umowy, a co za tym idzie bywa, że nie wyceniamy jej osobno. W szczególności, jeśli nie dojdzie do podpisania umowy, to czas na nią poświecony często

uznajemy za stratę (oczywiście wszystko zależy od uzgodnień z klientem, gdyż może ten proces być osobno fakturowany). Stąd też bardzo ważne jest znalezienie złotego środka pomiędzy precyzją i ogólnością tej analizy – punkty 1 i 2. Z kolei wymieniony powyżej punkt 3 jest podstawą do stwierdzenia, czy projekt ten jest realny do wykonania przy zadanych warunkach brzegowych, tak ze strony biznesowej (np. szacunkowy budżet), jak i technologicznej (np. musimy to wykonać w technologiach związanych z Mainframe, których zupełnie nie znamy).

Zastanówmy się zatem jak wykorzystać wiedzę, którą przypomnieliśmy w poprzednich punktach przy przeprowadzaniu wstępnej analizy. Korzystając z informacji zawartych w puncie „Wydobywanie wymagań", możemy zauważyć, że analiza powinna zawierać informacje:

1. pozwalające na oszacowanie biznesowych i technicznych możliwości dla tworzonego systemu,
2. o środowisku technicznym klienta, zasadniczo ważne jest środowisko docelowe,
3. o wszystkich aplikacjach i systemach, z którymi musimy zintegrować nowy system,
4. o wymaganiach funkcjonalnych wobec tworzonego systemu.(powinien być to w miarę kompletny spis wszystkich oczekiwań klienta).
5. o kluczowych użytkownikach systemu (w miarę możliwości należy krótko porozmawiać z każdą z tych osób),
6. o scenariuszach użycia dla ważniejszych i trudniejszych fragmentów systemu.

W powyższych punktach jedynie ostatni wymaga od nas pewnego komentarza, otóż w zasadzie w pełnej analizie (niezależnie od metodologii tworzenia oprogramowania) wymaga się jak najwiecej scenariuszy użycia, jednak ich tworzenie i spisywanie jest czasochłonne. Stąd we wstępnej analizie czasami rezygnuje się z nich w ogóle, my zalecamy by rozważyć ich utworzenie dla najważniejszych (najtrudniejszych) fragentów, gdyż pomogą one oszacować czas i koszt wytworzenia takiej funkcjonalności. Przeprowadzając analizę wstępną musimy wciąż pamietać o zagrożeniach wynikajacych z problemów: zakresu, zrozumienia i zmienności.

Nie mamy w procesie wstępnej analizy za dużo czasu na wszystkie te elementy, które są wymienione w punkcie „Analiza i negocjacja wymagań", stąd też jedynie pobieżnie musimy zwrócić uwagę na najważniejsze aspekty:

1. Czy każde wymaganie ma wyznaczone granice i jest na tyle jednoznaczne, że będziemy umieli oszacować koszt i czas jego wykonania?
2. Czy każde wymaganie jest wykonalne w środowisku w którym będzie pracował system?
3. Czy każde wymaganie będzie poddawało się testom i czy da się zaimplementować?

Spróbujmy szerzej omówić, czemu wybór padł na te punkty. Pierwszy punkt jest dosć oczywisty i nie wymaga szerszego omówienia. Drugi punkt jest ważny i również wydaje się oczywisty, jednak musimy pamiętać o tym, że osoba przeprowadzająca analizę, może nie posiadać dostatecznej wiedzy technicznej, by umieć na niego odpowiedzieć. Stąd, też warto w drugim punkcie sporządzić listę zagrożeń, wypływających z niewiedzy, lub z braku szczegółowych informacji na ten temat. Trzeci punkt jest bardzo ważny, gdyż coraz więcej metodyk tworzenia oprogramowania (np. Extreme Programming) kładzie szczególny nacisk na tworzenie testów (przypadków testowych) zanim w ogóle przystąpimy do tworzenia systemu. Jednak w tym punkcie nie chodzi o to, by od razu sporządzić testy dla każdej funkcjonalności, a jedynie by niejako „na wyczucie" określić, czy dana funkcjonalność podda się testom (jest to naprawdę nieoczywiste i więcej informacji na ten temat znajduje się w dalszych modułach). W przypadku watpliwości, lepiej od razu zapytajmy klienta, czy wie jak przetestować ten fragment systemu.

Na podstawie wcześniejszej wiedzy, w szczególności podpunkt o wydobywaniu wymagań, zastanówmy się teraz jak mógłby wyglądać dokument związany ze wstępną analizą wymagań, znów możemy odwołać się do wcześniejszej wiedzy. I tak możemy zaproponować następujące rozdziały, czy też sekcje:

1. **Wprowadzenie** – krótki opis celu systemu.
2. **Słownik** – definicja wszystkich technicznych i specyficznych terminów użytych w dokumencie. Choć mówimy tylko o wstępnej analizie, jest to sekcja bardzo ważna, gdyż jak to wcześniej pisaliśmy nie wolno czynić żadnych założeń co do wspólnej wiedzy pomiędzy nami a klientem. Dobrze zdefiniowany słownik pomoze uniknąć wielu problemów zarówno przy wycenie, jak i przy późniejszej realizacji.
3. **Modele systemu** – dość istotny fragment opisu, o ile uda się już na tym etapie taki model sporządzić.
4. **Definicja wymagań funkcjonalnych** – jak pisaliśmy już wcześniej, powinien się tu znaleźć możliwie pełny spis wymagań, przy czym nie muszą one jeszcze być opisywane bardzo szczegółowo. W tej sekcji można również zawrzeć informacje o danych, jaki system będzie przetwarzał.
5. **Definicja wymagań nie funkcjonalnych** – w tej seksji przede wszystkim należy wypisać ograniczenia wynikajace z oczekiwań klienta co do technologii, w której ma to być realizowane.

Oczywiście powyższy układ jest jedynie propozycją, każdy z nas może zaproponować własny, bardziej mu odpowiadajacy, co zresztą będzie zilustrowane w następnych przykładach. Powinniśmy pamiętać o tym by każdy tego typu dokument posiadał metryczkę historii zmian, jest ona potrzebna by śledzić, jakie zmiany i kto nanosił (patrz „Przykładowe rozwiązanie").

Doszliśmy do „Modelowania systemu", choć jak to powiedzieliśmy wcześniej jest to bardzo ważny element analizy, to jednak ze względu na jego pracochłonność jest często pomijany na tym etapie. Pamiętajmy o nim w sytuacjach dużych niejasności co do oczekiwań klienta, wtedy zastąpi on częściowo prototypowanie. W następnej kolejności w analizie mamy „Weryfikację wymagań" i jest to etap, którego w żaden sposób nie wolno ani pominąć, ani też zbagatelizować – jest on kluczowy z punktu widzenia przyszłości projektu. Dokument sporządzony na podstawie rozmów z klientem powinien zostać przekazany do weryfikacji i ostatecznie zatwierdzenia. Dopiero po ostatecznej akceptacji klienta możemy traktować go jako bazę do sporządzenia wstępnego harmonogramu i kosztorysu. Pamiętajmy, że rzadko możemy sporządzić wstępną analizę po jednej rozmowie z klientem, jednak osoba posiadajaca doświadczenie i odpowiednią wiedzę powinna to zrobić po 2-3 spotkaniach, zatem cykle spotkań i innych czynności w ramach wstępnej analizy powinien wyglądać następująco:

1. **Pierwsze spotkanie z klientem** – z naszej strony powinny w nim uczestniczyć 2-3 osoby, przy czym jedna prowadzi rozmowę i wypytuje o wszystko, sporzadza własne notatki, druga osoba notuje absolutnie wszystko, ewentualna trzecia osoba podsuwa pierwszej zagadnienia, które należy dodatkowo poruszyć.
2. **Opracowanie pierwszej wersji dokumentu** – dokument wstępnej analizy zostaje opracowany przez naszych pracowników. Może się zdarzyć, że potrzebują oni coś dodatkowo zapytać klienta, lub skonsultować się z konsultantami po naszej stronie. Jednocześnie z tym dokumentem powinien powstać spis pytań, jakie należy zadać klientowi podczas kolejnego spotkania. Dokument zostaje wysłany do przejrzenia do klienta.
3. **Drugie spotkanie z klientem** – na spotkanie udają się te same osoby, ewentualnie może zostać zaproszony jakiś konsultant, o ile jest to niezbędne. Możemy również poprosić klienta o dodatkowe osoby po jego stronie, które posiadają wiedzę dziedzinową. Na spotkaniu zadajemy wcześniej przygotowane pytania, a klient zgłasza wątpliwosci i uwago do przesłanego wcześniej dokumentu.
4. **Opracowanie drugiej wersji dokumentu** – dokument wstępnej analizy zostaje doprecyzowany i przesłany do ostatecznej akceptacji przez klienta.

Jak wcześniej pisaliśmy, o ile zajdzie taka potrzeba, możemy udać się na trzecie spotkanie i doprecyzować pozostałe wątpliwości. W praktyce unikamy wielu spotkań, ze względu na podnoszenie kosztu tego procesu.

Zauważmy ostatecznie, że pominęliśmy zupełnie „Ewolucję systemu", jednak nie ma w tym nic dziwnego, jest to zbyt wczesny etap by omawiać takie aspekty, wyjątkiem jest sytuacja, gdy od razu klient informuje, że oczekuje duzego systemu, który ma być dostarczany w etapach, ale to już zupełnie inna sytuacja i nic nie stoi na przeszkodzie, by zawrzeć taki rozdział w dokumencie wstępnej analizy.

Podsumowanie

W tym rozdziale przedstawione zostały główne fakty z zakresu inżynierii systemowej, inżynierii wymagań oraz analizy i specyfikacji wymagań klienta. Wszystkie podane informacje należy traktować jako sygnalizację zagadnień, których rozwinięcie można znaleźć we wskazanej literaturze.

Przykładowe rozwiązanie

Zlecenie od klienta

Klient – Pan Wacław – posiada firmę zajmująca się doradztwem inwestycyjnym w zakresie różnego rodzaju produktów – zarówno powiązanych z ubezpieczeniem, jak i typowo inwestycyjnych. Pan Wacław doskonale zdaje sobie sprawę, że klienta może zdobyć poprzez profesjonalną obsługę. Jednak kawa, herbata czy przytulne biuro już nie wystarczają, osoby do niego przychodzące nie mają czasu i same są zaganiane, zatem oczekują przedstawienia oferty czytelnie i szybko. Nasz Klient postanowił, że chciałby mieć program, który będzie umożliwiał porównanie ofert inwestycyjnych dla jego klientów. Porównanie powinno odbywać się za równo w postaci tabeli potencjalnych zysków w zadanym okresie czasowym jak i za pomocą wykresów graficznych.

Naszym zadaniem jest przeprowadzenie wywiadu z Panem Wacławem w celu pozyskania od niego wymagań, a następnie ich analiza.

Przygotowanie do wywiadu

Zanim pójdziemy przeprowadzić wywiad musimy do niego się przygotować. Zacznijmy od wydobywania wymagań. Zastanówmy się, jak na podstawie 6 punktów zawartych w punkcie Wstępna analiza wymagań, możemy przygotować się do rozmowy z klientem. Poniżej odnosimy się do każdego z tych 6-ciu punktów z zachowaniem ich kolejności:

1. Ponieważ mamy do czynienia z relatywnie prostym projektem, to nie ma specjalnie problemu z szacowaniem jego biznesowych i technicznych możliwości.
   a) W zakresie biznesowych możliwości projekt pozostawiamy w wyłącznej gestii Klienta i nie podejmujemy na ten temat dyskusji.
   b) W zakresie technicznych możliwości musimy jedynie dowiedzieć się na ilu komputerach system ma działać (czy ma być sieciowy) oraz jaki system operacyjny jest preferowany.
2. W zakresie środowiska technicznego i jego implikacji znów mamy uproszczoną sytuację, gdyż jak zauważyliśmy to będzie prosty system.
3. Zdecydowanie warto zapytać, czy nasza aplikacja ma współpracować z systemami zewnętrznymi.
4. Najważniejszym elementem wywiadu będzie pozyskanie informacji o wymaganiach funkcjonalnych wobec tworzonego systemu.
5. W przypadku omawianego projektu jedynym kluczowych użytkownikiem systemu, będzie Pan Wacław, zatem tu mamy prostą sytuację.
6. Do pierwszej rozmowy nie będziemy przygotowywali się w zakresie scenariuszy użycia.

Na podstawie tych założeń sporządzimy teraz listę pytań do naszego Klienta, przy czym ograniczymy się do najważniejszych i dla ułatwienia będziemy grupowali je punktami jak powyżej (poniżej pytań znajduje się dodatkowy komentarz):

1. Na jakim systemie operacyjnym chciałby Pan uruchamiać stworzoną aplikację? *[Tu warto pamiętać, że Klient może posiadać np. system Windows 98, my mamy prawo z nim dyskutować, czy nie warto dokonać modernizacji sprzętu. Nie zawsze dopasowanie do tego co klient posiada jest ekonomicznie właściwe. Zauważmy, że tworzenie oprogramowania na platformy już „nie modne" bywa trudniejsze i w efekcie kosztowniejsze. Zatem czasami warto zrobić podsumowanie kosztów rożnych wariantów i zastanowić się, co Klientowi w ostatecznym rachunku bardziej się opłaca – warto rozmawiać o ty m z Panem Wacławem i mu uświadomić te aspekty. Osobna sprawa to fakt, iż nowe technologie znamy, a starych nie, zatem wyceną będzie uwzględniała ten aspekt.]*
2. Proponowane pytania w tym punkcie to:
   a) Czy tworzona aplikacja ma pracować w środowisku sieciowym i tym samym czy planuje Pan zatrudnić pracowników?
      [Zawsze system sieciowy jest trudniejszy do tworzenia od aplikacji działającej na pojedynczym komputerze, zatem odpowiedź na to pytanie wpłynie zasadniczo na czas i koszt realizacji.]
   b) W konsekwencji powyższego pytania, czy aplikacja ma posiadać system użytkowników oraz ich uprawnień?
      [To znów jest bardzo ważne pytanie, budowa systemu uprawnień mocno zmienia sposób projektowania aplikacji i wpływa na wydłużenie czasu realizacji i tym samym podnosi koszty.]
   c) Czy posiada Pan licencję na jakąś bazę danych, czy też mamy wolną rękę za zakresie propozycji w tym zakresie?
      [Z pozoru może się wydawać, że jeśli klient posiada już bazę danych, to będzie taniej. Jednak wcale to nie jest takie proste, gdyż dla przykładu, technologia w której będziemy tworzyć aplikację może nie wspierać w dobry sposób bazy Klienta. Zatem odpowiedź na to pytanie niesie ważną informację, ale nie jest krytyczna. Istnieje wiele baz, które są darmowe lub relatywnie tanie dla niewielkich projektów.]
3. Tu warto zadać kilka pytań
   a) Czy aplikacja stworzona dla Pana ma współpracować z innymi systemami?
      [W pierwszej chwili nasz Klient może udzielić negatywnej odpowiedz na to pytanie, ale to dlatego, iż nie wie jakie korzyści mogą płynąć z automatyzacji pewnych procesów – spójrzmy na dalsze pytania, które zadamy.]
   b) Panie Wacławie, proszę zauważyć, że dużo obliczeń będzie korzystało z różnego rodzaju danych publikowanych np. WIBOR, LIBOR i inne. Czy chciałby Pan, by takie dane były wczytywane automatycznie?
      [Oczywiście pytanie to może być znacznie szersze w zależności od rozmowy, może okazać się, że jakieś firmy dostarczają danych do swoich produktów, można rozszerzyć ofertę o notowania giełdowe itd.]
   c) Panie Wacławie, z pewnością Pana Klienci to ludzie obeznani z nowymi technologiami, być może chcieliby otrzymać arkusz kalkulacyjny by w domu móc na spokojnie zastanowić się nad Pana propozycjami. Czy zatem rozważa Pan eksport wyliczeń z Pan systemu do arkusza kalkulacyjnego?
4. Wymagania funkcjonalne to najdłuższa część, z konieczności ograniczymy się do kilku pytań:
   a) Proszę spróbować scharakteryzować grupy produktów, jakie oferuje Pan swoim Klientom, pomoże nam to określić typy parametrów jakie będzie Pan musiał wprowadzać.
      [Tu należy podkreślić, ze to pytanie/polecenie wydaje się z pozoru proste, ale należy pamiętać, że grupowania/generalizacja nie jest procesem łatwym dla osób nie nawykłych do tego. Zatem pewnie nad samym tym pytaniem możemy spędzić trochę czasu dyskutując i wyszukując podobieństwa w produktach.]

b) Czy chciałby mieć Pan bazę Klientów z możliwością przechowywania dla nich historii ofert?
[Pytanie dość oczywiste i odpowiedź raczej będzie znana. Niemniej jednak należy pamiętać, że w tym momencie dotyczą nas zagadnienia związane z ustawą o ochronie danych osobowych.]
c) Jakie dane Pańskich Klientów są niezbędne do stworzenia symulacji inwestycji?
[Z pozoru wydaje się, że wystarczy nazwa do identyfikacji klienta, bo przecież jeśli ktoś inwestuje pieniądze to po co cokolwiek więcej. Jednak nie jest to prawda, bo jeśli Pan Wacław oferuje produkty z ubezpieczeniem to są potrzebne dane klienta do wyliczenia ryzyka śmierci.]
d) Jakie sposoby prezentacji wyników Pana interesują, czy mógłby Pan pokazać przykłady?
[Musimy pamiętać, że konkretne przykłady są zawsze lepsze od opowiadania „na sucho".]
e) Czy umie Pan na ten moment określić jakieś „nietypowe" funkcje systemu?
[To pytanie może wydawać się dziwne, ale jest ważne, gdyż ma Klienta uczulić, że powinien nam mówić prawie o wszystkim co przyjdzie mu do głowy.]

Pozostałe dwa punkty omijamy z oczywistych względów. Ostatecznie sformułowaliśmy wstępną listę pytań do Pana Wacława. Należy pamiętać, że pytania te zadawane w trakcie rozmowy będą w naturalny sposób rodziły kolejne pytania. Jednak należy pilnować się przed wchodzeniem w zbytnie szczegóły na początku jest to nie wskazane.

Pierwsza rozmowa

Na rozmowę udajemy się przygotowani i unikamy wizyt w pojedynkę, wynika to z faktu, iż jedna osoba powinna pytać i robić luźne notatki, a druga powinna notować wszystko. Co prawda obecnie można posiłkować dyktafonem, jednak notowane na żywo daje dużo lepsze efekty w zakresie rozumienia wymagań klienta. Co więcej, jeśli idziemy na rozmowę z kimś, to po wizycie mamy partnera do wymiany poglądów.

W trakcie rozmowy należy możliwie dużo notować oraz jeśli tylko się da rysować, daje to lepsze zobrazowanie przepływu danych, stwarza możliwość innego spojrzenia na problem. Szczególnie Klient, który prawdopodobnie pierwszy raz widzi takie diagramy może być pobudzany do spojrzenia na problem od innej strony.

Opracowanie wyników rozmowy

Po rozmowie z klientem należy jak najszybciej sporządzić wstępny spis jego oczekiwań – czyli rodzaj notatki ze spotkania. Taka notatka powinna powstać najpóźniej następnego dnia po spotkaniu, wynika to z faktu, iż nasza pamięć jest zawodna i im dłuższy czas upłynnie od spotkania, tym więcej zapomnimy.

Dobrze jest przygotować szablon notatki, taki by każda następna wyglądała podobnie, co więcej należy wprowadzić numerację takich dokumentów, by potem do nich móc się odwoływać. Dla przykładu wszystkie notatki możesz poprzedzać skrótem „NOT" a potem może występować sygnatura projektu/klienta dla którego notatka jest wykonywana, następnie datę i numer, taki sposób zapisu pomaga porządkować pliki. Zatem nasza pierwsza notatka może mieć następująca nazwę pliku NOT-WAC-2010_07_01-01.docx (patrz przykładowy plik).

Druga rozmowa

Na drugą rozmowę z Panem Wacławem udajemy się z opracowaną notatką, oraz przygotowanymi pytaniami uszczegóławiającymi lub wyjaśniającymi niejasności. Dobrze mieć również pewien zarys przyszłego systemu w postaci np. wypisane w punktach co powinna przyszła aplikacja robić.

Finalna specyfikacja

W przypadku naszego – niezbyt skomplikowanego – projektu, dwie rozmowy powinny być wystarczające, bo sporządzić dokument wstępnej analizy, oraz przedstawić Klientowi naszą propozycję. Należy unikać przy niewielkich i niezbyt skomplikowanych projektach poświęcania zbyt wiele czasu na fazę inicjacji tego projektu (w skład której wchodzi w naszym przypadku wstępna analiza), wynika to z prostej zależności, o której wspomnieliśmy już wcześniej – czas ten musi być „opłacony" poprzez realizację projektu. Stąd zdecydowanie dwa spotkania i nie więcej.

Przechodzimy do tworzenia ostatecznej propozycji dla Klienta, która powinna być podsumowaniem dwóch spotkań oraz naszych przemyśleń. Powinna mieć formę ustrukturyzowaną i napisaną możliwie zrozumiałem językiem (bez zbędnego żargonu informatycznego) – pamiętajmy, że Klient musi ten dokument przeczytać i zrozumieć. Dobrze jest, jak pisaliśmy wcześniej, zawrzeć rysunki i schematy, które ilustrują przepływ informacji. Tworząc ten dokument musimy mieć w świadomość, iż będzie on załącznikiem do umowy na wykonanie prac, stąd też należy unikać w nim przykładowo następujących stwierdzeń:

- „Aplikacja będzie umożliwiała wydruk raportów." – takie stwierdzenie otwiera przed Klientem możliwość żądania stworzenia dowolnego raportu w dowolnym momencie, w efekcie projekt nigdy się nie zakończy.
- „Aplikacja będzie pozwalała tworzyć wykresy dla Klienta." – znów, podobnie jak powyżej, brak ostrych kryteriów.
- „System będzie umożliwiał obliczenie dowolnego ubezpieczenia Klienta." – i znów brak ostrych kryteriów, z pozoru wydaje się, że przecież nie może być inaczej, bo jak Pan Wacław będzie wprowadzał ubezpieczenie, ale znów brak ostrych kryteriów może doprowadzić do problemów.

Przykład finalnego dokumenty znajduje się w pliku SPC-WAC-2010_07_15.docx.

Porady praktyczne

Bardzo trudno wskazać prosty i zamknięty zbiór porad praktycznych, wynika to z faktu, iż każdy projekt bywa inny, ale co najważniejsze, każdy Klient to inna osoba i może zupełnie inaczej formułować swoje potrzeby. Zdecydowania warto wspomnieć o dwóch rzeczach, które mają charakter uniwersalny – nie zależą praktycznie od rodzaju projektu:

- Zawsze przed pierwszą rozmową warto poczytać o dziedzinie, której projekt dotyczy, jeśli są to ubezpieczenia to o nich, jeśli finanse to o tym, nawet warto spotkać się z kimś, kto wyjaśni podstawowe zagadnienia. Dzięki temu łatwiej będzie przygotować zestaw pytań. Co więcej przy dużych projektach i ważnych kontraktach nawet warto zatrudnić zewnętrzną osobę, która zna daną dziedzinę.
- Zawsze przy analizie należy mieć otwarty umysł – słuchać co Klient mówi, nie zamykać się na żadne rozwiązanie, nie odrzucać żadnego pomysłu, dokonywać kalkulacji ekonomicznej, a nie emocjonalnej.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- rozumiesz jakie problemy leżą u podstaw inżynierii systemowej oraz inżynierii wymagań,
- rozumiesz, z jakimi problemami spotykamy się w trakcie pozyskiwania wymagań i do jakich niejednoznaczności to może doprowadzić,
- umiesz stworzyć wstępny zestaw pytań w celu pozyskania wymagań,
- umiesz przeanalizować odpowiedzi klienta i doprecyzować niejasności,
- umiesz stworzyć spis wstępnych wymagań,
- wiesz na co należy zwracać uwagę w trakcie pozyskiwania wymagań od klienta,

- wiesz, jakie czynności trzeba wykonać by pozyskać wymagania od klienta oraz by jest przeanalizować i sprecyzować,

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. S. Motamarri. *Systems modeling and description*. Software Enineering Notes, 17(2) pp. 57-63, 1992.

   W artykule tym Autor omawia zagadnienia związane z modelowaniem i opisem systemów.

2. Andrzej Jaśkiewicz, *Inżynieria oprogramowania,* Helion 1997

   W książce Autor omawia podstawowe zagadnienia związane z procesem wytwarzania oprogramowania.

3. Roger S. Pressman (adapted by Darrel Ince), *Software Engineering – A Practitioner' Approach, European Adaptation*, McGraw-Hill Publishing Company 2000

   W książce Autor przedstawia zagadnienie tworzenia oprogramowania z punktu widzenia praktyka, co stanowi bardzo wartościowe ujęcie tematu.

4. Ian Sommerville, *Inżynieria oprogramowania*, WNT 2005

   W książce Autor omawia proces wytwarzania wielkich systemów informatycznych. Autor, sam będąc światowej sławy specjalista w tej dziedzinie, przedstawia kolejne etapy tego procesu.

5. Ścibór Sobieski, *Inżynieria oprogramowania*, skrypty w wersji elektronicznej http://www.scibor.org/wp-content/uploads/2007/01/si.pdf

   W tym skrypcie Autor omawia ogólne zagadnienia związane z inżynierią oprogramowania zwracając szczególną uwagę na aspekt praktyczny.

6. http://en.wikipedia.org/wiki/Test-driven_development

   Na tej stronie znajduje się Szic metody Test Driven Development (TDD).

Laboratorium podstawowe

Jesteście podzieleni na zespoły 4-5 osobowe. W tym ćwiczeniu dwa zespoły będą pracowały razem. Przy takim podziale każdy zespół będzie pełnił rolę Klienta dla zespołu przeciwnego i odwrotnie. W przypadku nieparzystej ilości zespołów, jeden zespół musi pełnić rolę dwu Klientów (projekty powinny być inne i zaleca się podział grupy na dwie tak by byli to dwaj klienci), jednak nie można dopuścić do sytuacji, w której jeden zespół jest wykonawcą dla dwu Klientów – jest to czasowo nie wykonalne.

Zadanie	**Tok postępowania**
1. Klient wymyśla projekt	• Każda z grup wymyśla dla drugiej projekt do realizacji. • Projekt nie powinien być za trudny ani też za łatwy. Wtedy wywiad byłby zbyt długi, albo za krótki, by opis funkcjonalności nie zajął 4 linijek – takim bardzo banalnym projektem jest kalkulator. • Dobrze jest by drużyna spisała główne założenia projektu w formie choćby punktów, ale nie wolno ich pokazywać drużynie Wykonawcy. Mają one jedynie pomóc w omawianiu projektu.
2. Pierwsza rozmowa	• Grupy umawiają się wzajemnie na pierwsza rozmowę. • Najlepiej, jeśli uda się każdą z grup podzielić tak by połowa udawała Klienta, a druga połowa przeprowadzała wywiad. Jeśli z przyczyn podziału ról i kompetencji jest to nie możliwe, to wtedy wywiady musza następować po sobie.
3. Opracowanie wyników rozmowy	• Opracowanie wyników rozmowy, ze względów dydaktycznych, powinno odbywać się w ramach całej grupy. • Najpierw osoby przeprowadzające wywiad opracowują notatkę ze spotkania (można wykorzystać wzór podany wcześniej). • Następnie osoby przeprowadzające wywiad referują po krótce oczekiwania Klienta i wspólnie tworzony jest lista dodatkowych pytań.
4. Druga rozmowa	• Druga rozmowa przeprowadzana jest według analogicznego schematu jak pierwsza.
5. Finalna specyfikacja	• Opracowanie finalnej specyfikacji, znów ze względów dydaktycznych, powinno odbyć się w ramach całej grupy. • Notatkę ze spotkania drugiego opracowują osoby przeprowadzające rozmowę z Klientem. • Następnie ustalenia są przekazywane reszcie grupy i powstaje ostateczna propozycja dla Klienta.

Laboratorium rozszerzone

Zadanie 1

W prasie znalazło się następujące ogłoszenie:

Firma NT – zajmuje się prowadzeniem usług księgowych dla wielu firm z obszaru pewnego dużego miasta. Zakres działalności firmy to:

- *Obsługa księgowości w zakresie karty podatkowej, ksiąg podatkowych, ksiąg handlowych.*
- *Prowadzenie spraw kadrowych klientów, zatem przechowywanie akt pracowników etatowych, wysyłanie ich na badania, wymagane szkolenia, zgłoszenia do ZUS, naliczanie wynagrodzenia, rozliczenie z ZUS i urzędem skarbowym.*
- *Prowadzenie obsługi pracowników firm klientów, którzy zawierają umowę zlecenie lub dzieło.*

Firma NT jako cel działalności stawia sobie pełną obsługę klienta w obszarze kadrowo-płacowo-finansowym, tak by klient mógł skupić się na swojej podstawowej działalności. Drugim celem jest stopniowy rozwój firmy i obsługa coraz większej ilości klientów, którzy mogą być rozłożeni na dużym obszarze miasta i jego obrzeży. Stąd też, firma NT, chciałbym stworzyć wygodne narządzie komunikacji ze swoimi klientami oraz obiegu dokumentów.

Wstępne oferty prosimy przesyłać na adres firmy (tu adres).

Celem ćwiczenia podobnie jak poprzednio jest analiza i ustalenie wstępnych wymagań, jednak tu nie mamy Klienta z którym możemy porozmawiać, mamy tylko dość lakoniczne ogłoszenie, na podstawie którego mamy sporządzić ofertę. Zatem zadaniem każdej z grup jest próba postawienia się zarówno w roli Klienta jak i bycie jednocześnie Wykonawcą. Mając do dyspozycji Internet i jego zasoby należy sporządzić wstępną analizę wymagań Klienta.

Zadanie 2

Korzystając z dowolnej wyszukiwarki internetowej znajdź SIWZ (Specyfikację Istotnych Warunków Zamówienia) dla jakiegoś niewielkiego projektu informatycznego. Zwykle takie ogłoszenia zamieszczają gminy, czasami urzędy miasta. Zapoznaj się ogólnie z tą specyfikacją. Czy na jej podstawie umiesz przeprowadzić wstępną analizę dla takiego projektu? Spróbuj to zrobić.

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 3

Wersja 1

Środowisko pracy grupowej

Spis treści

Informacje o module

Opis modułu

W tym module znajdziesz omówienie problematyki stworzenia środowiska pracy zespołu projektowego i wyboru narzędzi ułatwiających pracę. Poprzez środowisko pracy będziemy rozumieli tu głównie oprogramowanie i jego właściwą konfigurację, tak by uzyskać optymalne środowisko pracy grupowej. Nie będziemy tu poruszać zagadnień związanych z otoczeniem, biurem, zapleczem itp. Nie będziemy tu również poruszać zagadnień związanych np. z elektroniczną ewidencją czasu pracy. Zatem znajdzie się tu omówienie typowych narzędzi wspomagających tworzenie oprogramowania.

Cel modułu

Celem niniejszego modułu jest zapoznanie czytelnika z ideą podstawowych narzędzi wykorzystywanych w projektach informatycznych.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- Wiedział, z jakimi narzędziami wspierającymi zespoły informatyczne można się spotkać w codziennej pracy, w dojrzałych zespołach programistycznych,
- Wiedział, jakie narzędzia w tym zakresie dostarcza firma Microsoft,
- Potrafił wykorzystać omówione narzędzia do zainicjowania pracy nad projektem,
- Rozumiał potrzebę i rolę zaawansowanych narzędzi wspierających pracę grupową.

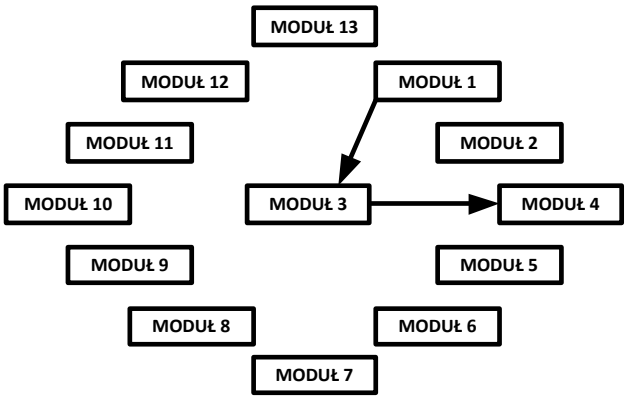
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawy tworzenia projektów w zespołach programistycznych (wymiana kodu, wspólna praca nad kodem).
- rozumieć dlaczego tworzenie oprogramowania w zespołach jest trudniejsze od strony organizacyjnej, niż w pojedynkę.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w module



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Posiadamy niewielki zespół programistyczny – powiedzmy 5-15 osób – chcielibyśmy zrealizować projekt dla pewnej firmy. Zanim jednak przystąpimy do realizacji, chcemy przygotować odpowiednie środowisko pracy, tak by później nie tracić czasu, a z drugiej strony, by środowisko to niejako ułatwiało pracę i wspierało standaryzowanie tworzenia. Przygotowując odpowiednie środowisko pracy ułatwimy sobie realizację projektu oraz unikniemy walki z „ginącymi piłkami", kilkoma wersjami tego samego dokumentu oraz innymi trudnościami wynikającymi ze specyfiki projektu.

Pamiętamy, że uczono nas niektórych narzędzi wspierających prace grupową. Gdzieś na ćwiczeniach przewinęły się takie pojęcia jak: zintegrowane środowisko programistyczne, serwer kontroli wersji, bugzilla i inne. Ale jak te narzędzie zainstalować? Tu okazało się, iż na początku nie było dużego problemu, bo system kontroli wersji, jakim jest SVN, łatwo zainstalowaliśmy z „paczki" na serwerze pod kontrolą systemu Linux, który i tak już wcześniej stał u nas w firmie do innych celów. Z instalacją Bugzilli już tak łatwo nie poszło. Waldek – nasz admin – męczył się 2 dni. No dobrze, a środowisko programistyczne? Właściwie to jeszcze nie wiemy, w jakim języku będziemy pisać, ani dla jakiego systemu operacyjnego, choć pewnie Klient ma Windows…

Najgorsze, że Marysia – nasz kierownik – chce by te narzędzia zintegrować w taki sposób, by łatwo propagować pomiędzy nimi wszelkie zmiany. To nas wykańcza, bo już 3 tydzień siedzimy i próbujemy to wszystko połączyć…

Podstawy teoretyczne

Zanim przejdziemy do części poświęconej temu jak dokonać wyboru w zakresie narzędzi tworzących środowisko pracy zespołu, to wpierw przypomnijmy po krótce, z jakimi głównymi typami narzędzi możemy mieć tu do czynienia.

Zintegrowane środowiska programistyczne

Zdecydowanie zintegrowane środowiska programistyczne (ang. IDE – Integrated Development Environment) są najbardziej znanym typem spośród omawianych. Spotykamy je już na samym początku rozwoju programisty i towarzyszą nam do końca kariery. Co prawda, wciąż tworzy się oprogramowanie poprzez pisanie kodu w edytorze, kompiluje przy użyciu make, a na sam koniec debuguje za pomocą gdb. Jednak IDE ułatwiają pracę poprzez integracje narzędzi i nie ma co do tego dyskusji. Wymieńmy zatem główne cechy, jakie będą charakteryzowały dzisiejsze środowiska tego typu:

1. Edycja kodu tworzonego programu wraz z podświetlaniem składni, coraz częściej udostępniane jest również autouzupełnianie.
2. Kompilacja/budowania aplikacji oraz tworzenie profili budowa np. Release (wersja dla klienta) i Debug (wersja z informacja dla debugera).
3. Zintegrowany debuger wraz z pracą krokową i podglądem zmiennych, rejestrów i często obliczaniem wyrażeń.
4. Wizualne programowanie np. tworzenie czy graficzne projektowanie okien dialogowych menu itp.
5. Tworzenie kontrolek i elementy programowania komponentowego.
6. Projektowanie schematu bazy danych i generowanie skryptów generujących bazę.

Istnieje bardzo dużo takich narzędzi, ale często są one specjalizowane do jednej platformy systemowej, a czasem tylko do jednego języka. Jako przykłady można podać: NetBeans, Eclipse (wieloplatformowe IDE, obsługujące wiele języków), Visual Studio (tylko Windows, obsługuje wiele

języków), Dev-C++ (tylko Windows, C++), Anjuta, KDevelop (tylko Linux, tylko C++), Eric (tylko Linux, tylko Python).

Systemy kontroli wersji

Systemy kontroli wersji (ang. version/revision control system) służą do śledzenia zmian w kodzie źródłowym (tak naprawdę łatwo wersjonuje się dowolne pliki tekstowe). Systemy te ułatwiają pracę w grupie, gdyż pomagają w łączeniu zmian dokonywanych przez kilka osób, wtedy pamiętana jest data zmiany oraz kto ją dokonał. Dzięki zachowaniu pełnej historii modyfikacji, łatwo jest sięgnąć do konkretnej wersji, czy też określonego momentu czasowego. Dzięki temu, przy zachowaniu odpowiedniej kultury tworzenia oprogramowania, istnieje zawsze możliwość powrotu do konkretnej wersji pliku, czy też całej aplikacji i stwierdzenie np. czemu wystąpił jakiś błąd – jest to cecha szalenie istotna w przypadku utrzymywania oprogramowania przez długi czas, kiedy dostarczamy Klientowi kolejne wersje. Dla przykładu my już pracujemy nad nową, ale Klient użytkuje wersję 1.0 i do niej raportuje błędy, w takiej sytuacji możliwość „cofnięcia" się do wersji 1.0 pozwala na stwierdzenie gdzie leży problem, co więcej pozwala na stwierdzenie, czy w obecnej wersji będzie ona nadal występował i jeśli tak to usunięcie go zawczasu.

Systemy kontroli wersji można podzielić na scentralizowane, oparte na architekturze klient-serwer (np. CVS, Subversion) i rozproszone, oparte na architekturze P2P (np. BitKeeper, svn, Git). W przypadku pierwszych istnieje jeden centralny serwer, a w przypadku drugich mogą istnieć niezależne równoprawne gałęzi, które można dowolnie synchronizować ze sobą nawzajem.

Systemy śledzenia błędów

Systemy śledzenia błędów (ang. Bug Tracking System - BTS), pozwalają na zamieszczanie błędów zgłaszanych przez użytkowników, rozwijających czy testerów. Każdy błąd otrzymuje numer i jest przechowywany w repozytorium wraz z całą historią „życia" błędu. Warto w tym miejscu omówić typowy „cykl życia":

1. osoba posiadająca uprawnienia do raportowania wprowadza informację o błędzie, czasami też oznacza jego ważność i priorytet,
2. osoba lub system określa osobę, której dotyczy błąd i generowane jest powiadomienie (jabber, email i inne kanały komunikacji),
3. osoba, która otrzymała powiadomienie, akceptuje go do poprawy lub deleguje ten błąd do innej osoby,
4. osoba odpowiedzialna kończy pracę nad błędem na jeden z kilku sposobów – informując, że błąd został poprawiony, odłożony w czasie, nie jest możliwy do poprawy czy, że zgłoszenie błędu było niepoprawne,
5. jeśli błąd usunięto, osoba odpowiednio uprawiona osoba weryfikuje, czy błąd faktycznie został usunięty i zwykle zgłoszenie błędu jest zamykane.
6. po ostatecznym rozwiązaniu problemu (np. po opublikowaniu albo instalacji u klienta wersji pozbawionej błędu) błąd jest zamykany.

Tak naprawdę dość często rozszerza się te systemy o ewidencje nie tylko błędów, ale również innych rzeczy takich jak np. rozszerzenia czy zmiany – ale ich idea pozostaje nie zmienna. Podstawowe cechy takiego systemu to:

1. Przechowywanie informacji o wszystkich zgłoszeniach wraz z ich historią.
2. Dostępność dla wszystkich członków zespołu, ewentualnie Klienta.
3. Umożliwianie przeszukiwania bazy pod kontem różnych kryteriów.
4. Umożliwianie sporządzania różnego rodzaju raportów np. skuteczności konkretnego programisty.

Istnieje wiele takich narzędzi, zarówno płatnych jak i darmowych, mniej lub bardziej zaawansowanych. Warto tu wymienić następujące: Bugzilla (bardzo zaawansowany system,

używany przez wiele aplikacji), Mantis, Trac, FlySpray (bardzo przyjemny i czytelny system), JIRA (zaawansowany i niestety nie tani system)

Narzędzia biurowe

Ta część aplikacji jest dobrze znana osobom pracującym na komputerach i nie będziemy jej rozwijać. Powiemy jedynie, że mowa jest tu o: edytorach tekstu, arkuszach kalkulacyjnych, czy edytorach schematów.

Narzędzia wspierające zarządzanie projektem

W tej kategorii istnieje z jednej strony wiele narzędzi, ale z drugiej tylko kilka jest naprawdę rozbudowanych i posiadających cechy, które są użyteczne przy dużych projektach, cechy takich narzędzi podajemy poniżej:

1. Definiowanie podstawowych informacji o projekcie.
2. Opracowanie struktury projektu (WBS lub SPP).
3. Tworzenie zależności pomiędzy zadaniami, ograniczenia sztywne i elastyczne zadań.
4. Tworzenie wykresów Gantta, PERT innych (wyznaczanie ścieżki krytycznej)
5. Definiowanie zasobów i ich zarządzanie (koszty, kalendarze).
6. Przypisywanie zasobów do zadań, balansowanie, rozwiązywanie konfliktów.
7. Szacowanie kosztów projektu.
8. Zapisywanie projektu bazowego, analiza odchyleń i zmian.
9. Raportowanie.
10. Zarządzania zasobów w różnych projektach.

W tej kategorii mamy na przykład narzędzie Microsoft Project (wraz z Project Server), Ganntt Projekt.

Podsumowanie

Kluczowym pytaniem, jakie się pojawia, jako tło do tego modułu jest „Jakie narzędzia są najlepsze dla zespołów programistycznych?”. Niestety nie ma jednoznacznej i łatwej odpowiedzi na tak postawione pytanie, wynika to z bardzo wielu przyczyn, wymieńmy dwie z nich:

1. Większość narzędzi, o których mowa, są przeznaczone na konkretną platformę, zatem jeśli chcemy tworzyć dla środowiska Linux, to nie bardzo możemy użyć np. Visual Studio.
2. Spora część narzędzi jest dostosowana do konkretnego języka lub ich grupy, zatem i tu nie mamy wolnej ręki.

Zatem zawsze wybór narzędzie będzie do jakiegoś stopnia określany poprzez realizowany projekt, z drugiej jednak strony, jeśli jako zespół świetnie tworzymy w C# w ramach ASP.NET, to nie należy spodziewać się poważnego zlecenia w Ruby on Rails.

Przykładowe rozwiązanie

Jako przykładowe rozwiązanie dla zespołu programistycznego zaprezentujemy i omówimy po krótce środowisko Visual Studio 2010 wraz z serwerem pracy grupowej Team Foundation Server 2010, oba produkty firmy Microsoft.

Wprowadzenie do Visual Studio 2010 i Team Foundation Studio 2010

Visual Studio 2010

Visual Studio 2010 (VS 2010) jest obecnie najnowszą wersją znanego od wielu lat rozbudowanego zintegrowanego środowiska programistycznego (IDE – Integrated Development Environment) firmy Microsoft. Jest to kompletne środowisko zawierające zarówno typowe narzędzia deweloperskie (np. kompilatory), jak i te bardziej zaawansowane, jak debuger, profiler, oraz narzędzia do

projektowania interfejsów, testów jednostkowych i wiele innych. Kod może być pisany w jednym z kilku języków: C#, Visual Basic .NET, C++, F#, a za pomocą tego środowiska możliwe jest tworzenie:

- **Aplikacji desktopowych** – zarówno w oparciu o bibliotekę WinForms, jak i wykorzystujących WPF.
- **Aplikacji dla przeglądarek** – w technologii ASP.NET i Silverlight.
- **Aplikacji dla urządzeń mobilnych** – aplikacje te mogą pracować pod kontrolą Windows Mobile i Windows Phone
- **Rozszerzeń aplikacji** – daje to możliwość tworzenia dodatków dla wielu aplikacji i systemów Microsoft, takich jak: Office czy SharePoint.

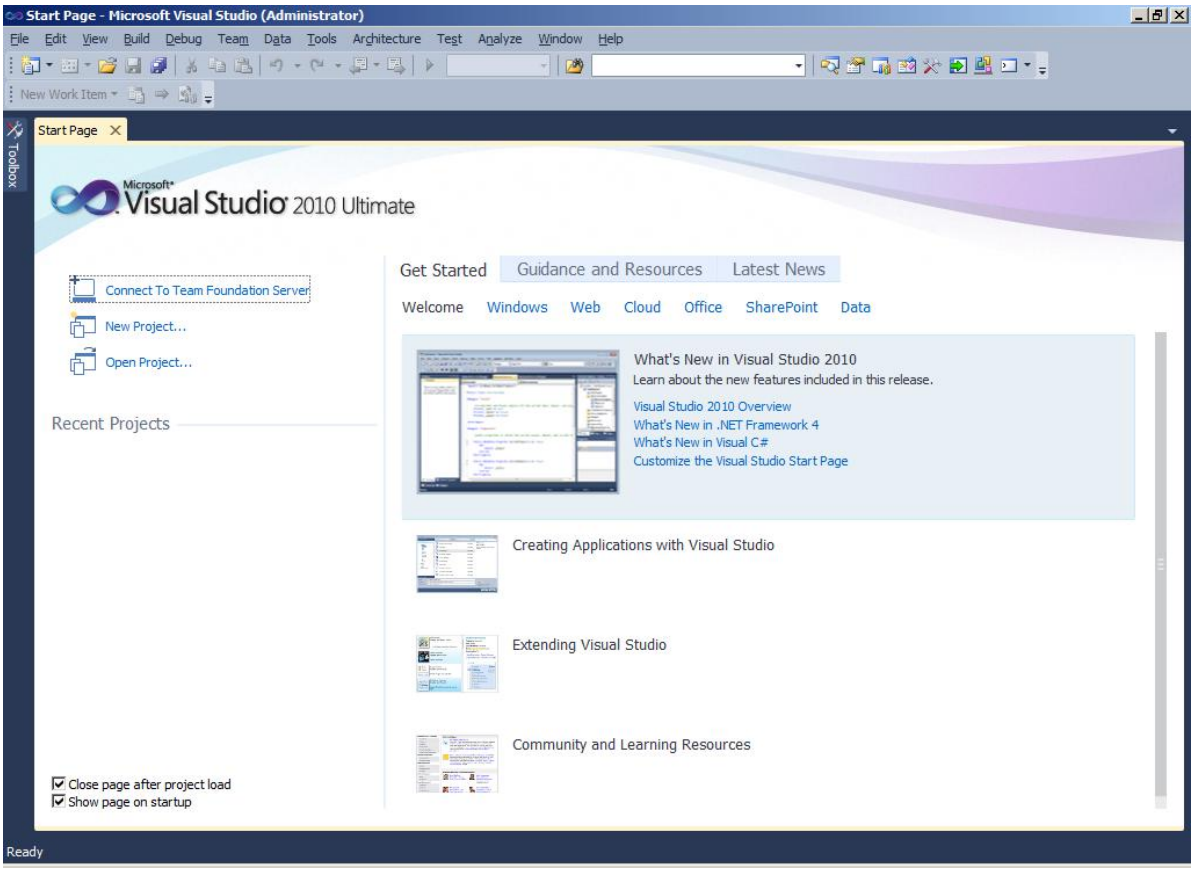
VS 2010 jest wypuszczany w kilku wersjach (Visual Studio Test Proffesional, Visual Studio Proffesional, Visual Studio Premium, Visual Studio Ultimate), z których Ultimate jest najbogatszą odmianą i tą wersją będziemy się posługiwać w dalszych opisach i ćwiczeniach. Osoby zainteresowane dokładnym porównaniem cech poszczególnych produktów powinny spojrzeć na stronę [W1]. Wersja Ultimate zawiera zaawansowane mechanizmy służące do testowania i debuggowania aplikacji, m.in.:

3. IntelliTrace – debugger, który pamięta historię wykonania aplikacji i umożliwia cofanie się, by prześledzić kroki wykonania.
4. Microsoft Test Manager – nowe środowisko do wykonywania testów zawierające: moduły testów obciążeniowych aplikacji, zarządzanie wirtualnym środowiskiem testowym.
5. Generator diagramów zgodnych z UML 2.0.

Dla osób, które miały do czynienia z wcześniejszymi wersjami tego środowiska może być interesujący skrót z wprowadzonych nowości:

- Całkowicie przebudowany interfejs, w tym dodano obsługa wielu monitorów.
- Dodana obsługę ASP.NET MVC, biblioteki dla aplikacji na Windows Phone 7 oraz Silverlight 4.
- Dodana obsługę tworzenia aplikacji z multitouch i wstążki.
- Umożliwiona budowę aplikacji dedykowanych dla SharePoint i dla środowiska Windows Azure.

VS 2010 po uruchomieniu powita nas oknem zaprezentowanym na Rysunku 1. Widać tu odnośniki zarówno do zasobów pomocy: „Get Started”, „Guidance and Resources” jak i linki do zewnętrznych zasobów, oraz po lewej stronie linki do podłączenia do serwera TFS (o czym dalej), tworzenia i otwierania projektu. Tutaj od razu uwaga, „New project” tworzy nowy projekt bez podłączania do serwera TFS.



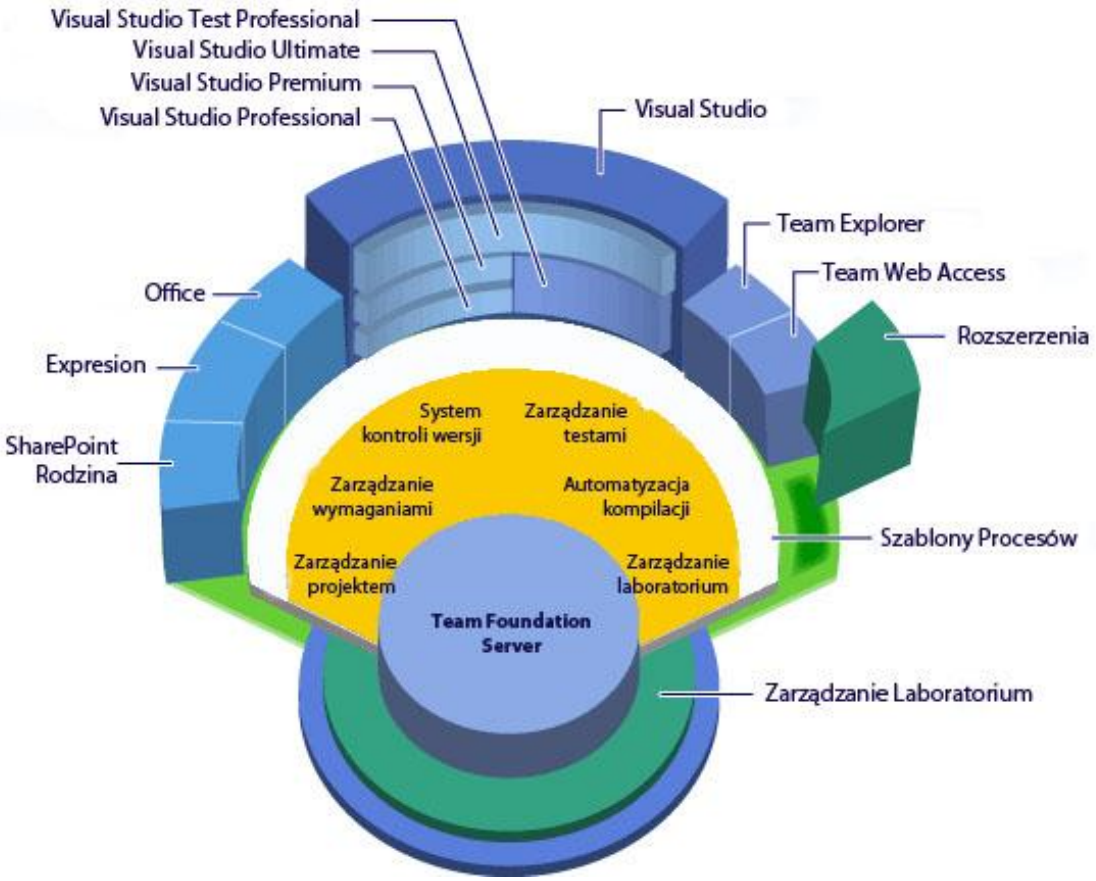
Rysunek 1. Okno powitalne Visual Studio 2010.

Team Foundation Studio 2010

O ile środowisko Visual Studio (niezależnie od wersji) z pewnością większości studentów jest choć trochę znane, to Team Foundation Server 2010 (czy jego poprzednik) raczej nie są powszechnie używanymi systemami na uczelniach. Jako główną przyczyną można podać dwa powody: pierwszy, to mały nacisk w procesie dydaktycznym kładziony na tak złożone środowiska pracy grupowej, a druga to niestety fakt, iż wersja TFS 2008 była bardzo trudna w konfiguracji i mało osób przez ten proces przeszło. Zacznijmy od ogólnej charakterystyki tego serwera.

Upraszczając można powiedzieć, iż TFS 2010 jest systemem do zarządzania projektami programistycznymi o bardzo rozbudowanych możliwościach, skupia on w sobie: serwer kontroli wersji, wspiera zarządzanie pracą w zespole, wspiera koordynowanie budowania aplikacji na różne platformy, posiada rozbudowany system raportów, wspiera budowę na serwerze oraz wykonywanie testów i wiele innych, np. dla każdego nowego projektu tworzona jest witryna SharePoint, która daje podstawę do budowy portalu projektu. Warto w tym miejscu powiedzieć, że TFS jest narzędziem wspierającym zarządzanie projektami w znaczeniu programistycznym, tzn. nie jest odpowiednikiem MS Project, choć daje możliwość podłączenia się tej aplikacji.

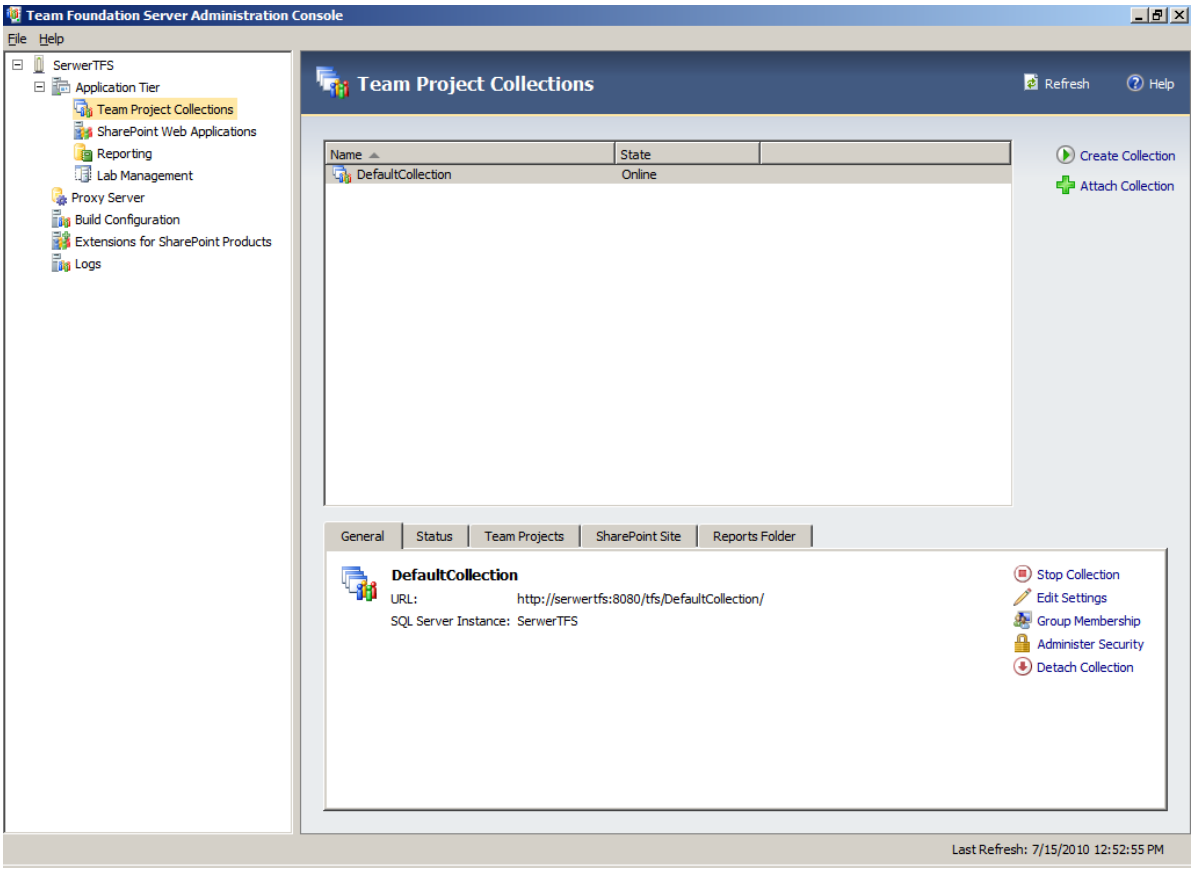
Firma Microsoft określa TFS jako rdzeń systemu zarządzania cyklem życia aplikacji (application lifecycle management – ALM) i faktycznie należy przyznać, iż pełne możliwości tworzenia aplikacji w zespołach programistycznych przy użyciu VS2010 uzyskuje się w oparciu o TFS 2010 – rysunek 2. pokazuje powiązania pomiędzy różnymi produktami rodziny VS 2010+TFS 2010 (więcej na temat ALM znajdziesz w module 5).



Rysunek 2. Rodzina produktów VS 2010+TFS 2010 wraz z ich funkcjami (opracowani własne na podstawie rysunku Microsoft).

Podstawowa instalacja TFS 2010 nie jest trudna i przebiega tak, jak to opisuje dokumentacja, po instalacji możemy otworzyć Team Foundation Server Administration Console (Rysunek 2.). W TFS 2010 projekty, którymi zarządzamy są umieszczane w kolekcjach, dla których możemy nadawać określone uprawnienia, takie podejście daje dużą swobodę wykorzystania tego serwera w dowolnej firmie, gdzie na przykład odbywa się praca na wielu projektach i w wielu zespołach i nie każdy ma prawo wszystko oglądać. Co prawda system uprawnień TFS 2010 już na poziomie uprawnień dla projektów jest dostatecznie rozbudowany, by dany programista nie miał dostępu do innego projektu niż jego, ale kolekcje dają więcej możliwości porządkowania.

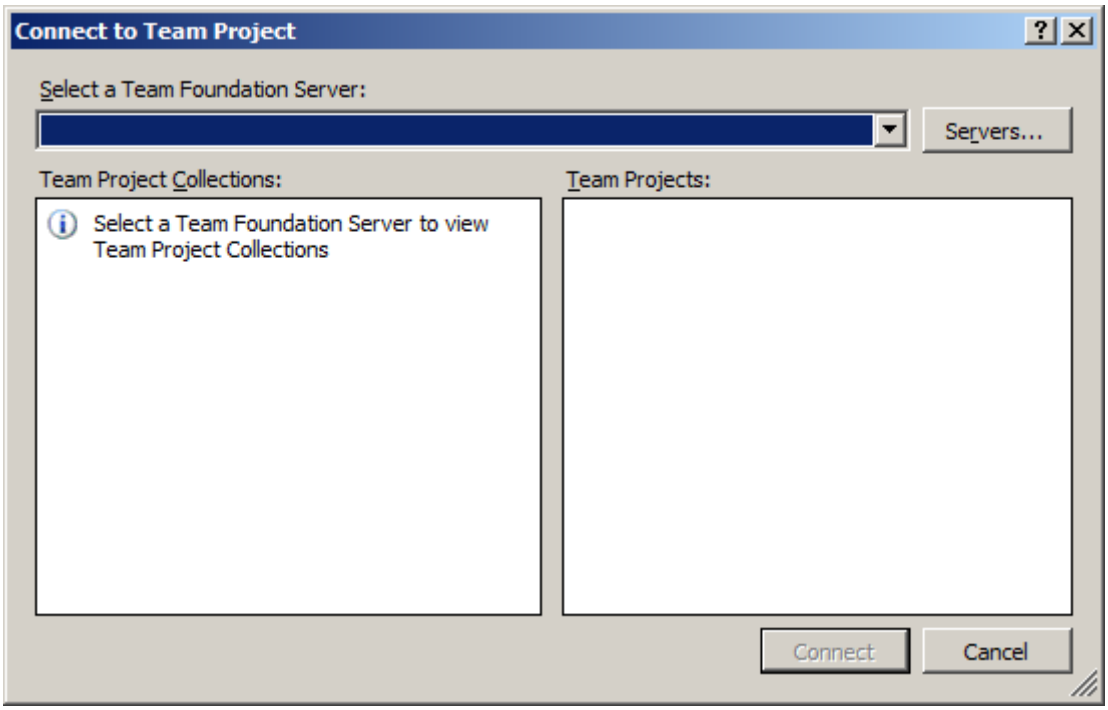
Pozostałe elementy, które widać na Rysunku 3., to: zarządzanie serwisami witryn SharePoint dla projektów, systemem raportowania, serwerem proxy (o którym nic nie powiemy), zarzadzaniem budowaniem aplikacji. W dalszej części opiszemy te funkcje, które są potrzebne zespołowi na samym początku.



Rysunek 3. Konsola administracyjna TFS 2010.

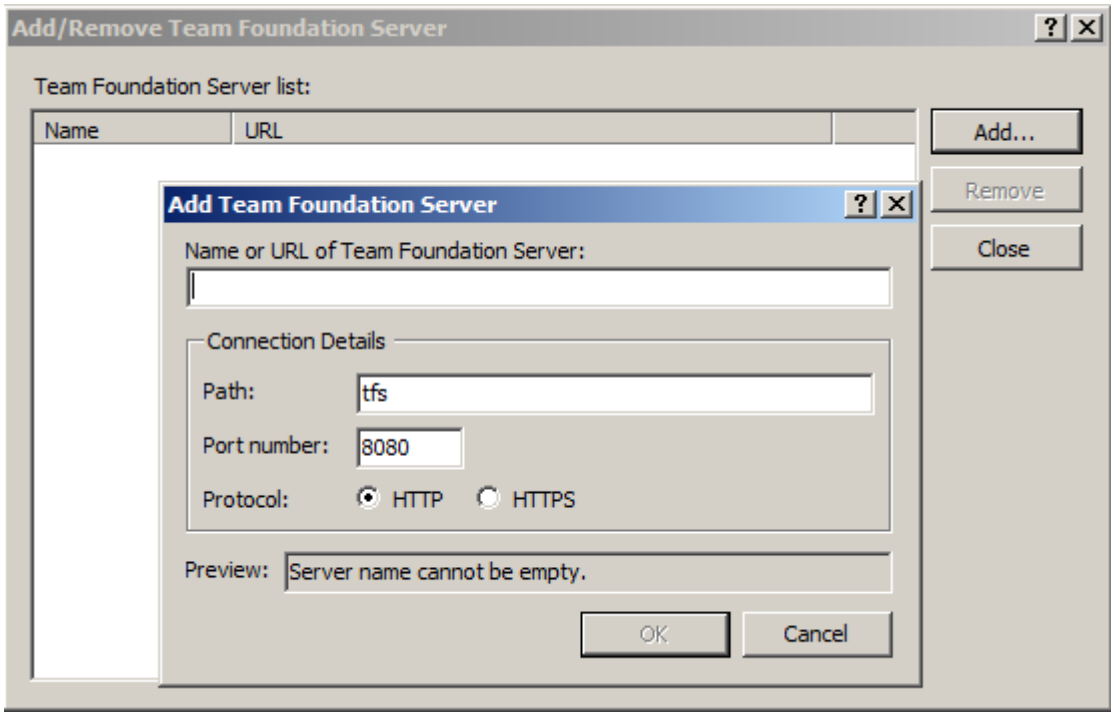
Tworzenie nowego projektu w VS 2010+TFS 2010

Jeśli powstaje nowy projekt, nad którym będzie miało pracować kilka osób, najwygodniej jest zacząć od jego utworzenie od razu w oparciu o serwer TFS 2010. W tym celu uruchamiamy VS 2010, a następnie musimy podłączyć się do jakiegoś serwera TFS 2010. Załóżmy, że jesteśmy zalogowani, jako Administrator, na pierwszym powitalnym oknie VS 2010 (Rysunek 1.) klikamy „Connect to Team Foundation Server" i otwiera się okno pokazane na rysunku 3., lista serwerów jest pusta, gdyż na początku – zaraz po instalacji – nie mamy skonfigurowanego dostępu do żadnego serwera.



Rysunek 4. Okno podłączenia do serwera TFS (z pustą listą serwerów).

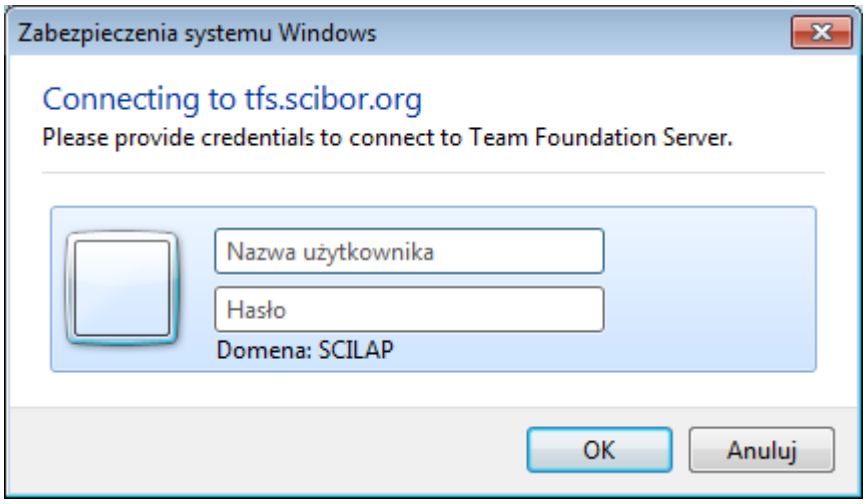
W celu dodania serwera TFS, klikamy w oknie Connect to Team Project (Rysunek 4.) przycisk Servers… i pojawia się nowe okno dialogowe (Rysunek 4.), tutaj znów lista serwerów jest pusta, zatem naciskamy przycisk Add… i ostatecznie pojawią się okno dialogowe Add Team Foundation Server (Rysunek 5.), w którym możemy wpisać parametry sieciowe dla tego serwera. W naszym przypadku wpisujemy jako nazwę localhost i resztę pozostawiamy bez zmian.



Rysunek 5. Okno pozwalające dodać do listy nowy serwer TFS.

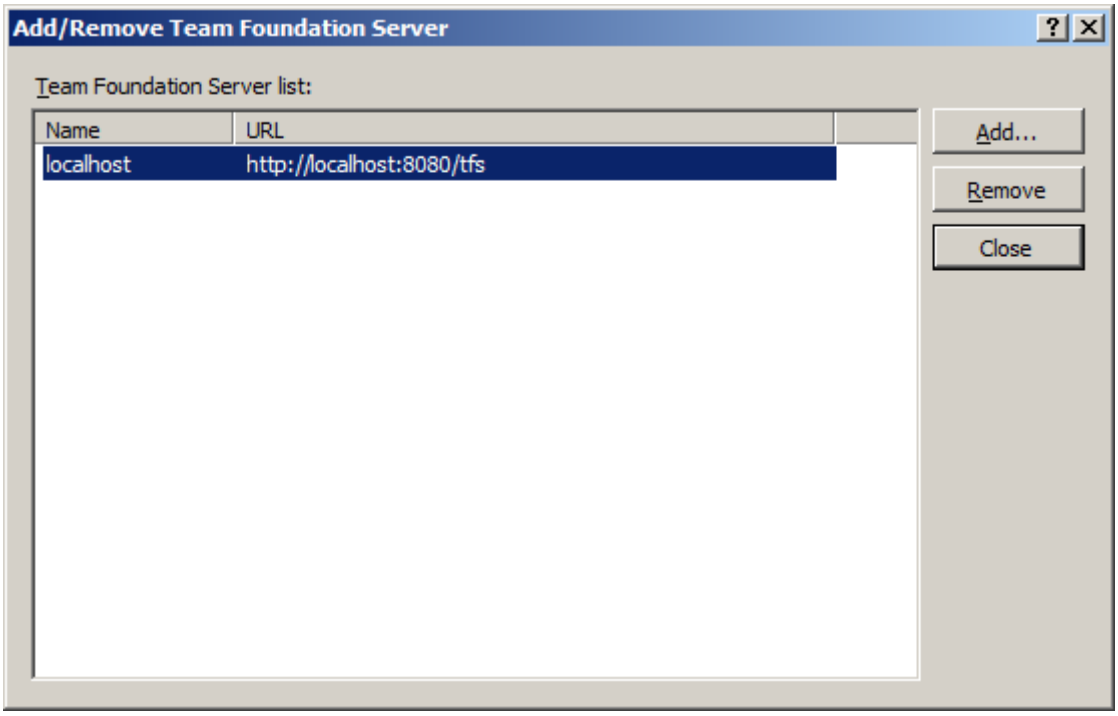
Uwaga, w przypadku korzystania z serwera na serwerze udostępnianym w sieci, po kliknięciu przycisku OK., pojawi się okno autoryzacji (np. takie jak na rysunku 5.), w którym należy wpisać

prawidłową nazwę i hasło użytkownika, który posiada odpowiednie uprawnienia do TFS. Jeśli działamy na serwerze lokalnym, to okno się nie pojawi.



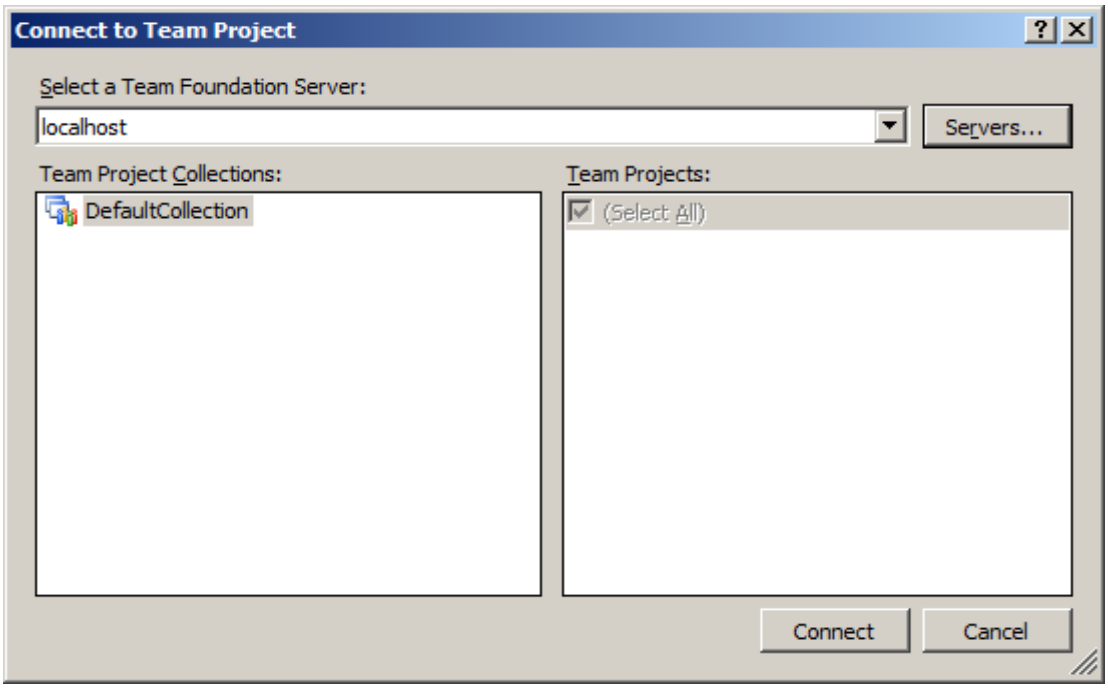
Rysunek 6. Przykładowe okno umożliwiające wpisanie hasła i użytkownika do serwera TFS.

Ostatecznie po dodaniu serwera pojawi on się na liście (Rysunek 7.). Okno dialogowe Add/Remove Team Foundation Server służy do zarządzania serwerami, zatem możemy w nim dodawać czy usuwać serwery, na których mamy projekty.

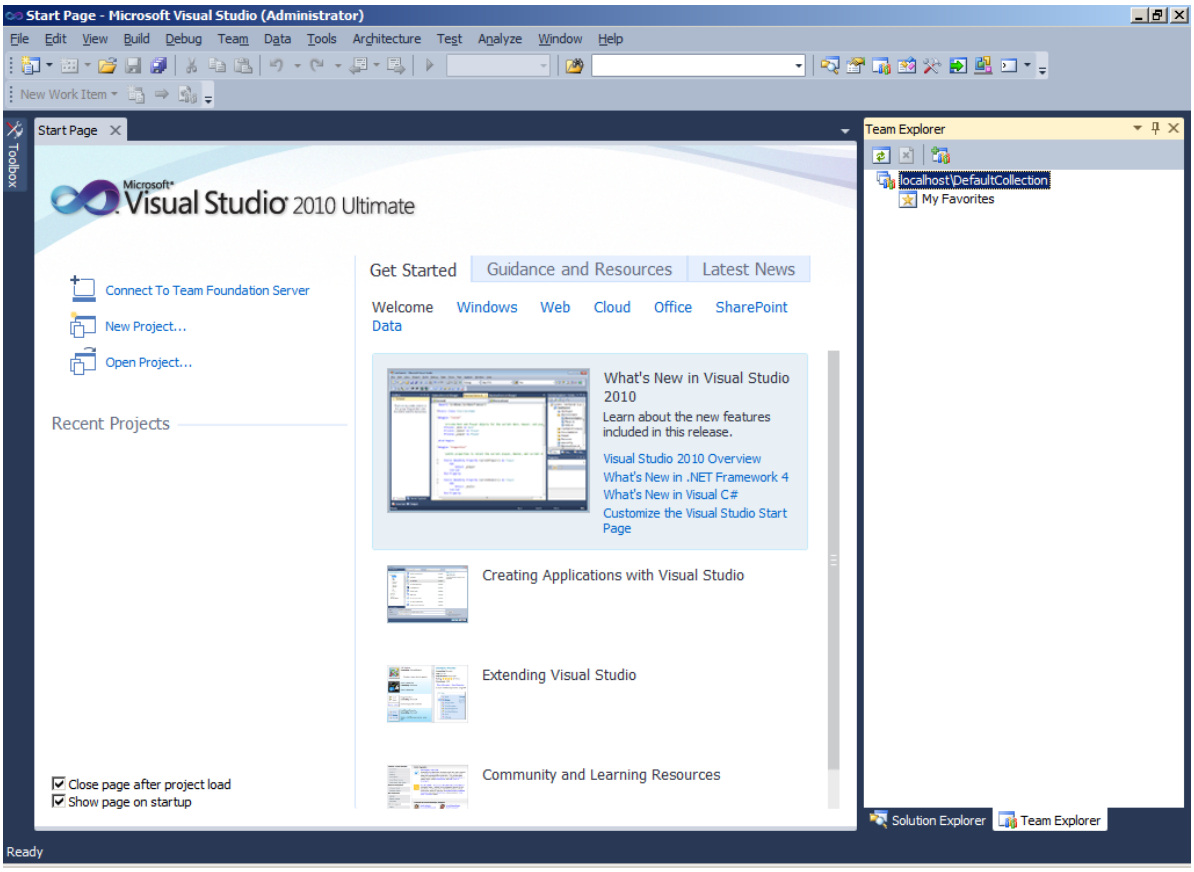


Rysunek 7. Okno zawierające listę serwerów (po dodaniu serwera na adresie lokalnym).

Po zamknięciu tego okna dialogowego wracamy do okna podłączenia do serwera TFS (Rysunek 8.), z listy serwerów wybieramy interesujący nas – w tym konkretnym przypadku localhost – jeśli połączenie z serwerem będzie prawidłowe to na liście Team Project Collections pojawią się dostępne dla nas kolekcje projektów. Po kliknięciu Connect podłączamy się do wybranej kolekcji i ostatecznie pojawia nam się początkowe okno VS 2010, ale już z oknem Team Explorer (Rysunek 8.)

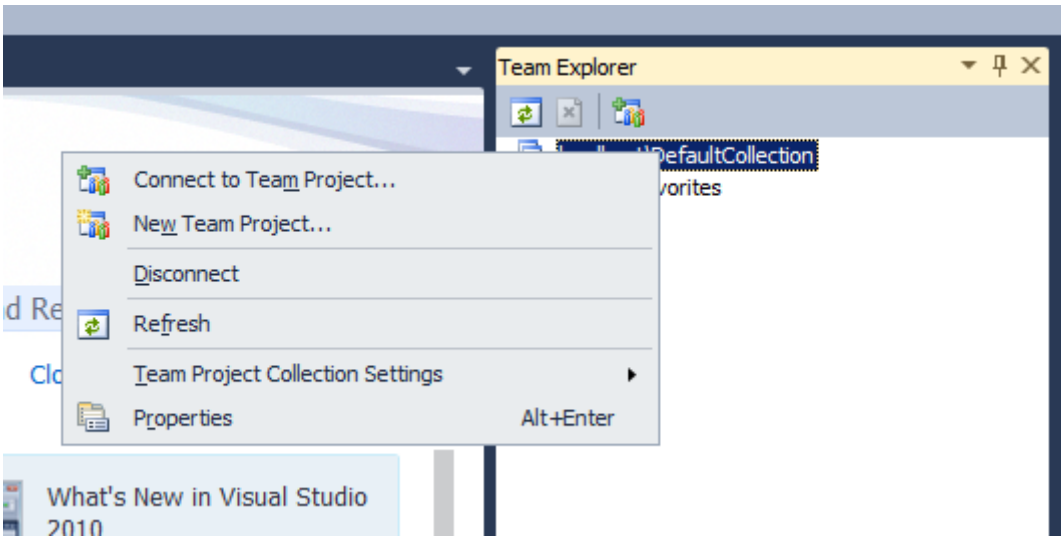


Rysunek 8. Okno podłączania do serwera TFS po wybraniu serwera, zawierające listę kolekcji na tym serwerze.



Rysunek 9. Okno początkowe VS 2010 wraz z oknem Team Explorer i podłączeniem do serwera TFS na serwerze lokalnym.

Chcąc dodać nowy projekt do kolekcji, klikamy prawym przyciskiem myszy, w oknie Team Explorer, gdy kursor myszy jest nad localhost\DefaultCollection, w rezultacie pojawia się menu jak na rysunku 10.



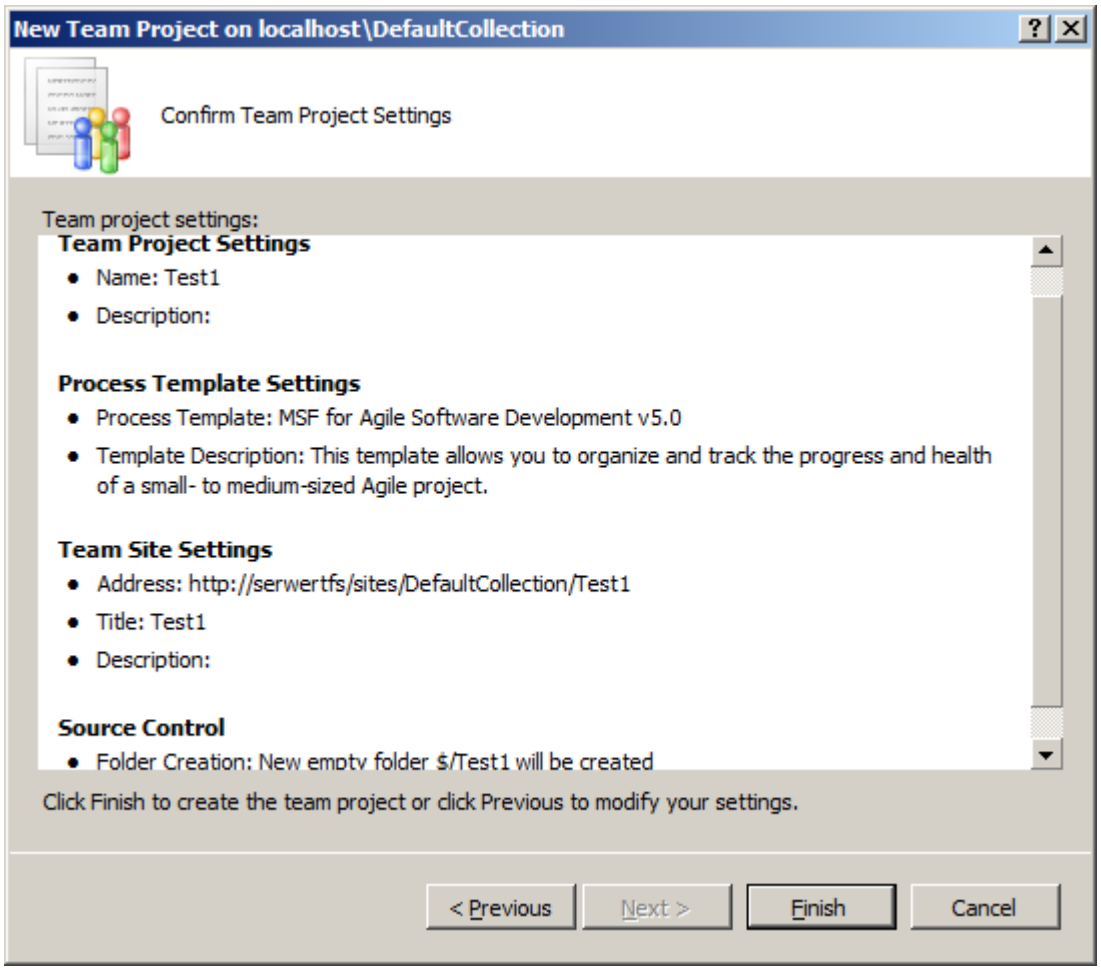
Rysunek 10. Menu pozwalające wykonywać operacje na kolekcji.

Kliknięcie pozycji menu New Team Project inicjuje proces zakładania nowego projektu podłączonego do TFS. Ponieważ proces ten jest prosty i sprowadza się do wypełnienia kolejnych pól, to nie będziemy pokazywać kolejnych ekranów a jedynie wskażemy, co należy w nich wpisać lub wybrać. Zakładanie projektu składa się z następujących kroków:

1. W pierwszym oknie (Specify the Team Project Settings) podajemy nazwę nowego projektu i jego opis – nie jest on obowiązkowy – np. Test1.
2. W drugim oknie (Select a Process Template) musimy wybrać szablon, według którego stworzony zostanie projekt. Szablony te umożliwiają tworzenie, a następnie prowadzenie projektu według określonego schematu, posiadają predefiniowane nazwy, narzędzia, raporty i wiele innych zgodne z danym standardem. Po zainstalowaniu TFS 2010 posiada zdefiniowane następujące dwa szablony:

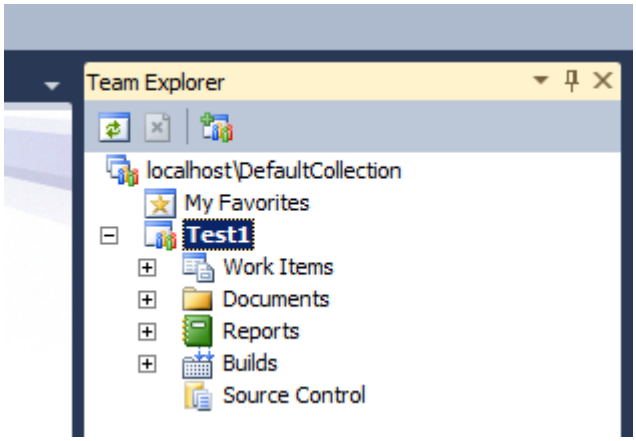
   a) **MSF for Agile Software Development v5.0** – szablon pozwalający tworzyć i nadzorować projekty zgodnie z metodyką Scrum – w oparciu o tę metodykę będziemy realizować większość modułów.
   b) **MSF for CMMI Process Improvement v5.0** – szablon pomagający tworzyć oprogramowanie według zasad standardu CMMI (Capability Maturity Model Integration), który został zdefiniowany przez organizację SEI (Software Engineering Institute).

   Dodatkową zaletą jest możliwość definiowania własnych szablonów, oraz wgrywania zdefiniowanych przez innych. Takie podejście daje ogromną elastyczność dostosowania tego środowiska do potrzeb konkretnej firmy.
   My wybieramy szablon Agile i przechodzimy do następnego kroku.
3. W następnym oknie (Team Site Sttings) mamy możliwość określenia, czy dla naszego projektu chcemy utworzyć witrynę SharePoint czy też nie. Wybieramy opcję utworzenia.
4. Kolejne okno (Specify Source Control Settings) pozwala na utworzenie nowego repozytorium plików dla potrzeb naszego projektu lub utworzenia go, jako kopi już istniejącego w repozytorium będącego odgałęzieniem (brench). Wybieramy utworzenie nowego.
5. Ostatnie okno kreatora (Confirm Team Project Settings) podsumowuje nasze wcześniejsze wybory, co pozwala się upewnić, że wybraliśmy pożądane ustawienia (w naszym przypadku prezentuje to rysunek 11.), ostatecznie naciskamy przycisk Finish i czekamy kilka minut na utworzenie projektu.
6. Jeśli projekt zostanie utworzony prawidłowo, to na koniec pokazuje się okno informacyjne (Team Project Created) i możemy wymusić, poprzez zaznaczenie checkboxa, otwarcie przewodnika o projekcie, a raczej o szablonie w oparciu, o który został utworzony.



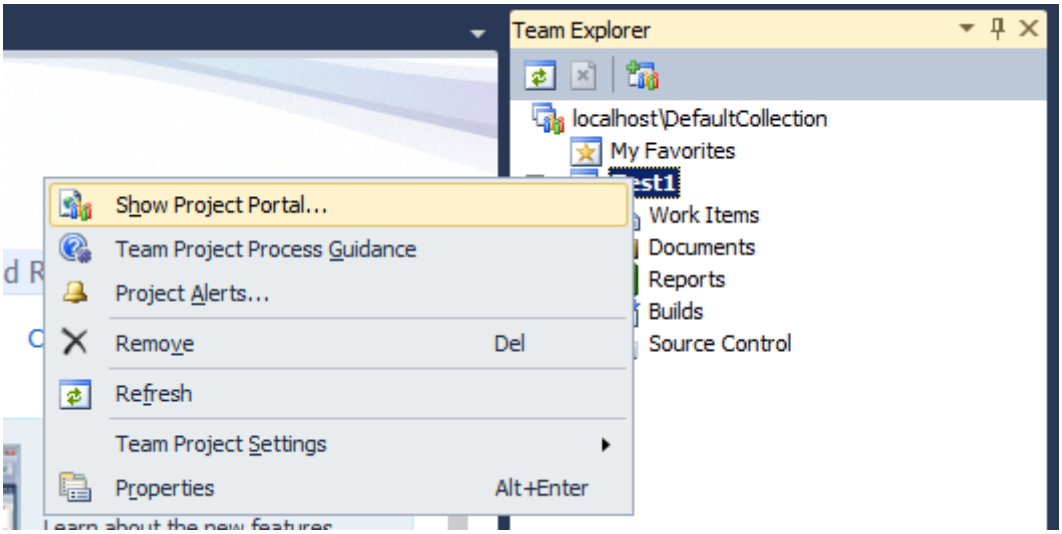
Rysunek 11. Okno podsumowujące ustawienia dla nowo tworzonego projektu.

Po utworzeniu projektu pojawia on się na drzewie w oknie Team Explorer (Rysunek 12.). Widać, że utworzone zostały gałęzie dla prac (Work Item), dokumentów, raportów oraz kolejnych kompilacji (Builds). W tym miejscu może pojawić się naturalne pytanie, czemu w trakcie tworzenia projektu nie zostaliśmy zapytani o typ aplikacji jaki chcemy utworzyć (np. konsola, WinForms, WPF, …). Otóż, przypomnijmy, że TFS zarządza cyklem życia aplikacji (ALM) i nie jest ona zależy od technologii, czy języka, w którym chcielibyśmy wytworzyć aplikację. Dodawanie konkretnego projektu i jego źródeł do TFS wykonamy za chwilę.

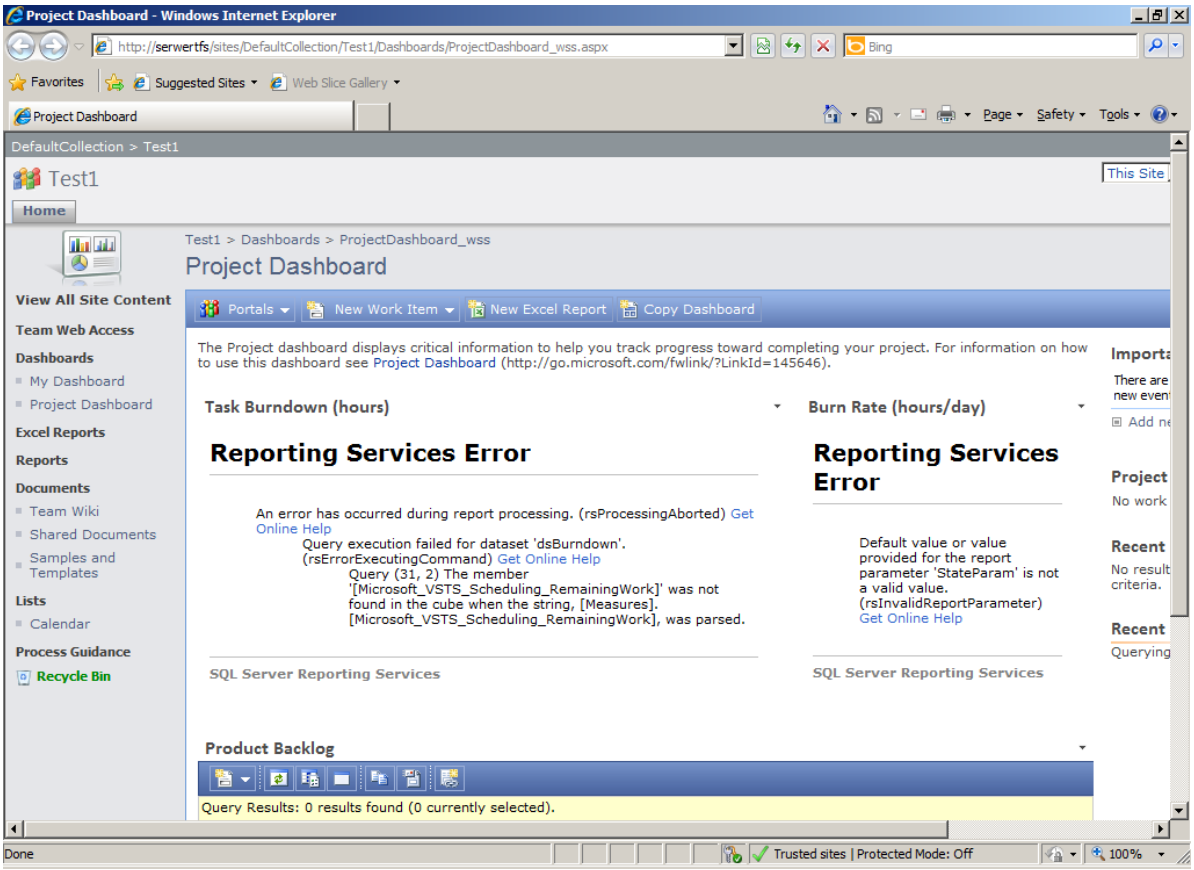


Rysunek 12. Team Explorer wraz z dodanym projektem Test1.

Jak pamiętamy wybraliśmy opcję utworzenia witryny SharePoint dla tego projektu, by do niej się dostać, wystarczy najechać kursorem myszy na nazwę projektu (Test1) i nacisnąć prawy klawisz myszy – pokaże się menu (Rysunek 13.) w którym możemy kliknąć w pozycję Show Project Portal, co w efekcie spowoduje otwarcie okna przeglądarki zwierające witrynę SharePoint (Rysunek 14.), która stanowi podsumowanie informacji o tym projekcie (Dashboard). Na samym początku możemy, jak w przykładzie, zobaczyć błędy raportów, jednak wynika to z faktu, iż projekt nie zawiera po prostu żadnego raportu (jednak stan tego ona zależy od rodzaju szablonu jakiego użyliśmy (więcej w module 5).

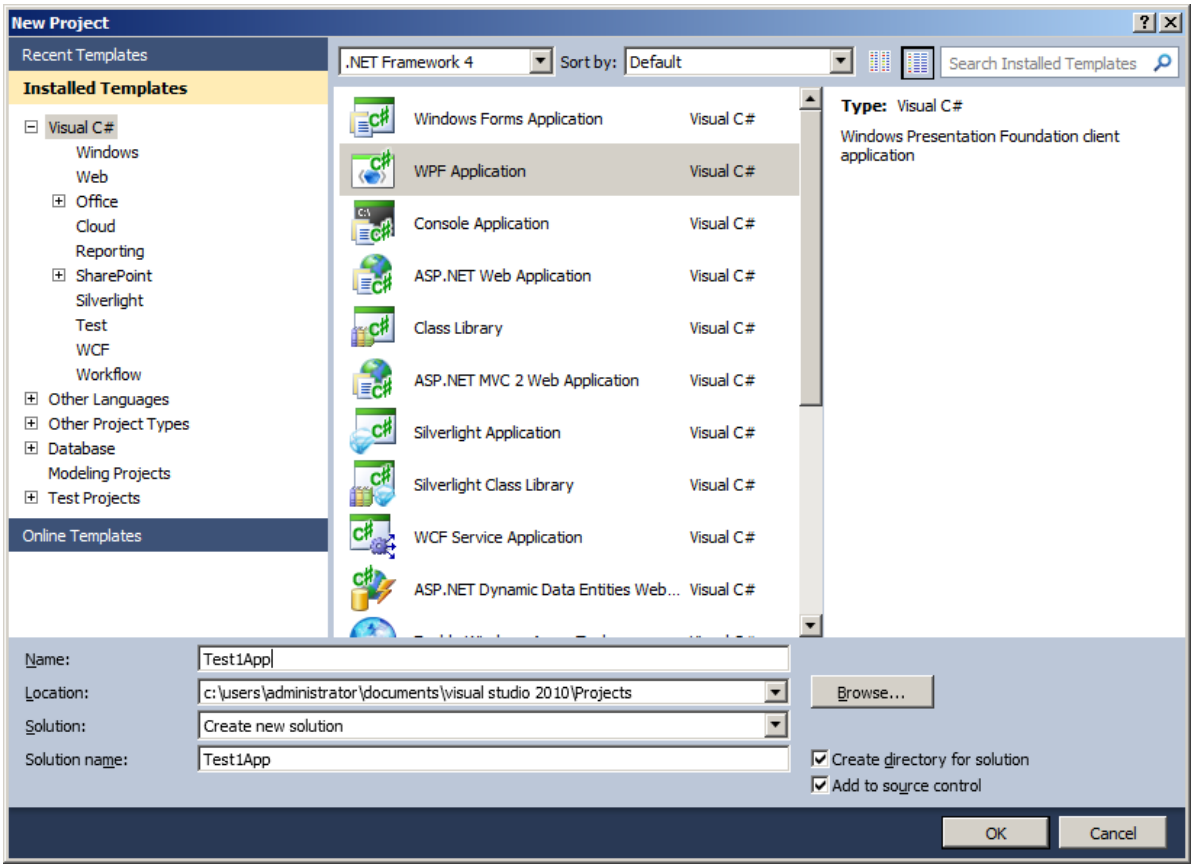


Rysunek 13. Menu dla projektu.



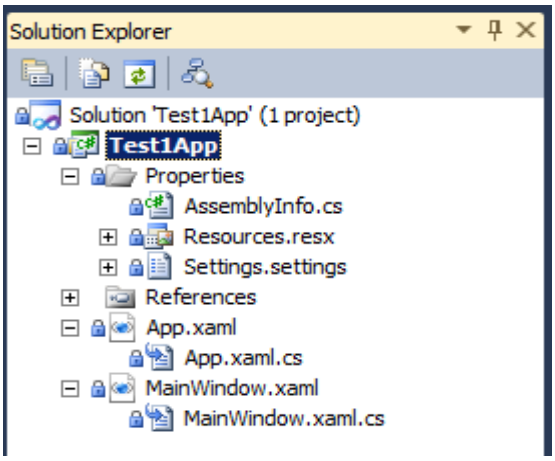
Rysunek 14. Okno podsumowania projektu na witrynie SharePoint.

Nadszedł czas, by wreszcie stworzyć nową aplikację, w tym celu z menu File, wybieramy pozycję New->Project, co spowoduje otwarcie okna dialogowego pozwalającego na wybór rodzaju aplikacji, języka, w której będzie tworzona, jej nazwy, lokalizacji oraz co dla nas jest ważne możliwości dodania kodu tworzonej aplikacji do repozytorium (Rysunek 15.). Jeśli, ktoś do tej pory już używał Visual Studio, okno to nie będzie dla niego nowością. Istotne jest zaznaczenia opcji Add to source control. Po wpisaniu pozostałych parametrów (nazwa: Test1App, rodzaj: WPF Application, .NET Framework 4, język C#) kończymy działanie tego okna przyciskiem OK.

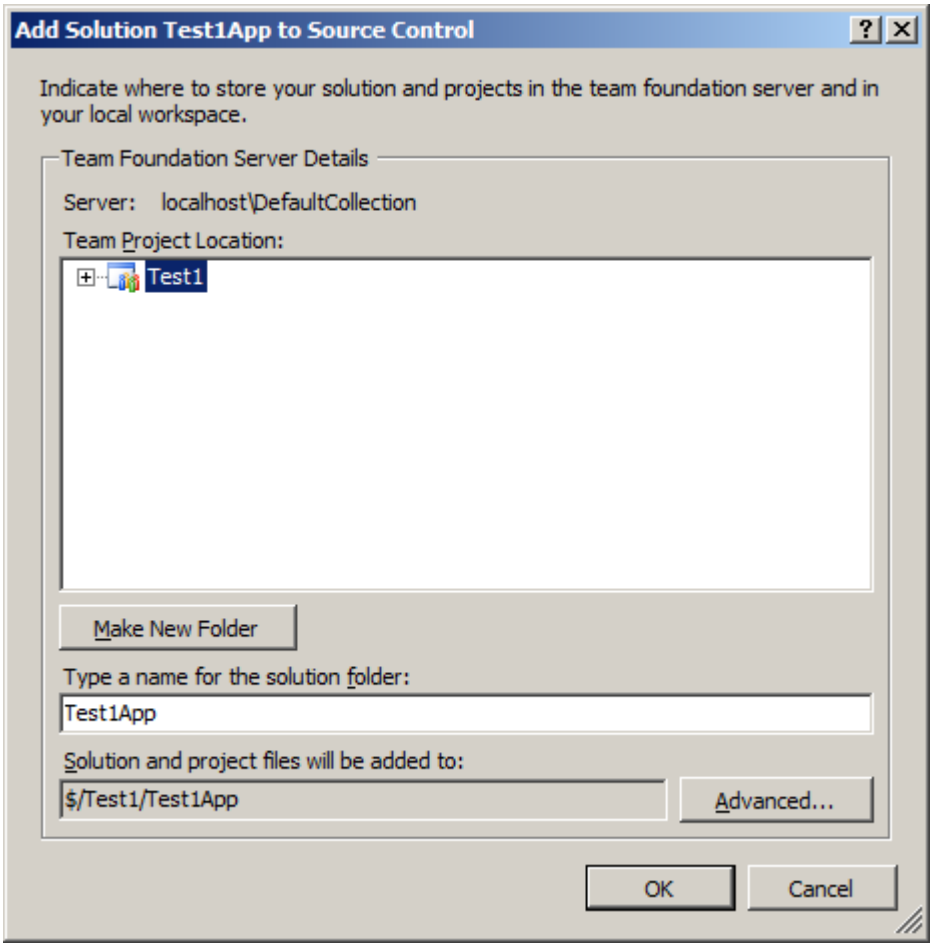


Rysunek 15. Okno pozwalające utworzyć nowy projekt.

W efekcie VS 2010 otworzy kolejne okno (Rysunek 17.) pozwalające doprecyzować parametry dodawania kodu do repozytorium. W oknie tym mamy informację o nazwę serwera i kolekcji, do której zostanie dopisana nasza aplikacja, dokładne jej położenie, możliwość określenia folderu, który będzie zawierał nasze pliki, a właściwie całe drzewo aplikacji i przycisku do zaawansowanych ustawień. W tym oknie nic nie zmieniamy, a jedynie zatwierdzamy je przyciskiem OK. Środowisko VS 2010 wygeneruje teraz aplikację wraz z wszystkimi potrzebnymi plikami, aby obejrzeć te pliki musimy przełączyć się do okna Solution Explorer (zwykle jest ona widoczne jako zakładka wraz z Team Explorer) i w tym okienku widać drzewo plików naszej aplikacji (Rysunek 16.).

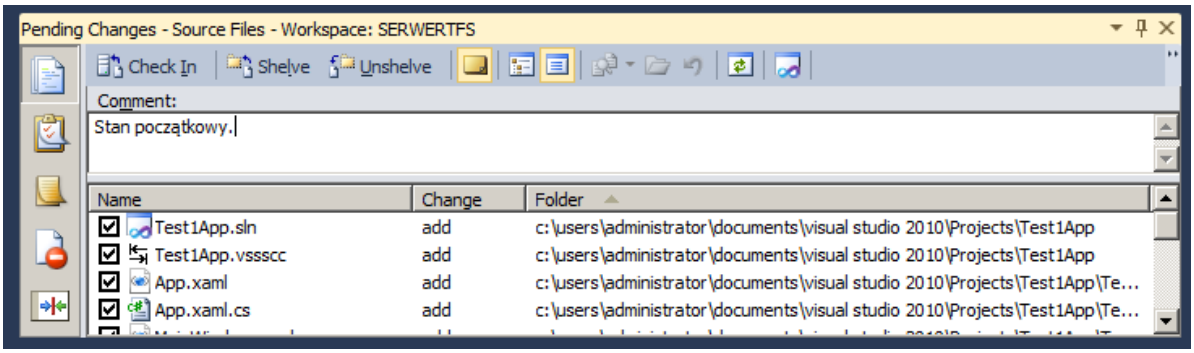


Rysunek 16. Okno Solution Explorer wraz z przykładowym drzewem aplikacji.



Rysunek 17. Okno opcji dodawania aplikacji do repozytorium.

Po wygenerowaniu wszystkich plików, w oknie Pending Changes (Rysunek 18.), można zobaczyć wszystkie zmiany, jakie mają zostać wykonane na repozytorium i oczekują na zatwierdzenie (check-in). W celu umieszczenia zmian lokalnych na serwerze repozytorium plików w oknie Pending Changes wpisujemy komentarz (np. Stan początkowy), opcjonalnie możemy wrzucić nie wszystkie pliki, jednak przy pierwszym umieszczaniu plików w repozytorium nie ma to sensu., zatem zostawiamy domyślne zaznaczenie wszystkich plików i naciskamy Check-In. Pojawi się okno weryfikacji, czy jesteś pewny, że chcesz umieścić (konkretna liczba) plików na serwerze, odpowiadamy tak i pliki zostają umieszczone w repozytorium.



Rysunek 18. Okno oczekujących zmian, które można zatwierdzić i wysłać do repozytorium.

W tym miejscu warto wspomnieć jeszcze o jednej funkcjonalności, która nie występuję w sposób bezpośredni w innych serwerach kontroli wersji, a mianowicie o działaniach Shelve i Unshelve (wydać to na rysunku 18.). Funkcje te są komplementarne, pierwsza z nich umieszcza zmiany na prywatnej „półce", a druga opcja je z tej „półki" zdejmuje. Przypomnijmy, że Check-In właściwie w każdym serwerze kontroli wersji działa w taki sposób, że umieszcza zmiany w głównym drzewie plików, zatem są one widoczne dla wszystkich. Shelve umieszcza te zmiany w prywatnej przestrzeni dostępnej tylko dla Autora tych zmian. Takie podejście daje dużą elastyczność, gdyż pozwala śledzić własne zmiany i dopracować je zanim zobaczą je inni. Warto o tej funkcji pamiętać, jednak wrzucenie początkowego drzewa aplikacji musi się odbyć oczywiście poprzez Check-In, gdyż muszą zobaczyć je pozostali użytkownicy. Jeśli ta operacja się powiedzieć, to lista plików w tym oknie powinna zostać wyczyszczona. Teraz nadszedł odpowiedni moment by dodać uprawnienia do tego projektu pozostałym członkom zespołu.

Użytkownicy w TFS 2010

Domyślnie serwer TFS jest instalowany z jednym użytkownikiem, jakim jest Administrator, oczywiście praca na jednym użytkowniku, w przypadku zespołu programistów jest rozwiązaniem bezsensownym, stąd też należy założyć użytkowników dla pozostałych członków zespołu (i przemyśleć, czy ktokolwiek powinien pracować jako Administrator). Proces zakładania nowego użytkownika zaczynamy od założenia standardowego użytkownika systemu Windows 2008.

Musimy pamiętać, że TFS to tak naprawdę trzy produkty: TFS jako taki, SharePoint oraz SQL Server Reporting Services, zatem będziemy musieli nadać uprawnienia do każdego z tych produktów na odpowiednim poziomie. Zacznijmy od przywołania nazewnictwa dla każdego z tych produktów, gdyż poszczególne bywają inaczej nazywane, nazwy te zawiera poniższa tabela.

Produkt	Czytelnicy	Współpracownicy	Kierownicy/Administratorzy
Team Foundation Serwer	Readers	Contributors	Project Administrators
SharePoint Products	Visitors	Members	Owners
SQL Server Reporting Services	Browser	Browser	Team Foundation Content Manager

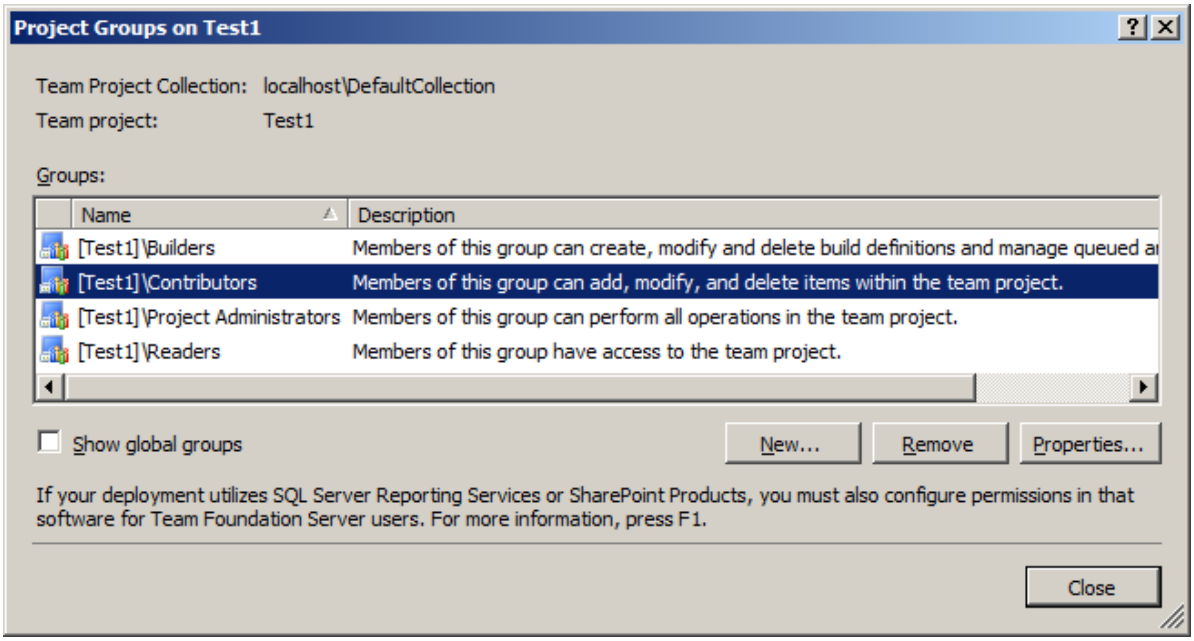
Należy również pamiętać, że zakładanie użytkowników jest możliwe, jeśli mamy poniższe uprawnienia:

- jeśli jesteśmy członkami grupy Project Administrators lub nasze uprawniania Edit Server-Level Information są ustawione na poziom Allow in Team Foundation Server.
- Jeśli jesteśmy członkami grupy Site Administrators lub Site Collections Administrators w ramach SharePoint.
- Jeśli jesteśmy członkami grupy Content Managers lub Project Content Managers w SQL Server Reporting Services.

Dodawanie użytkowników w TFS

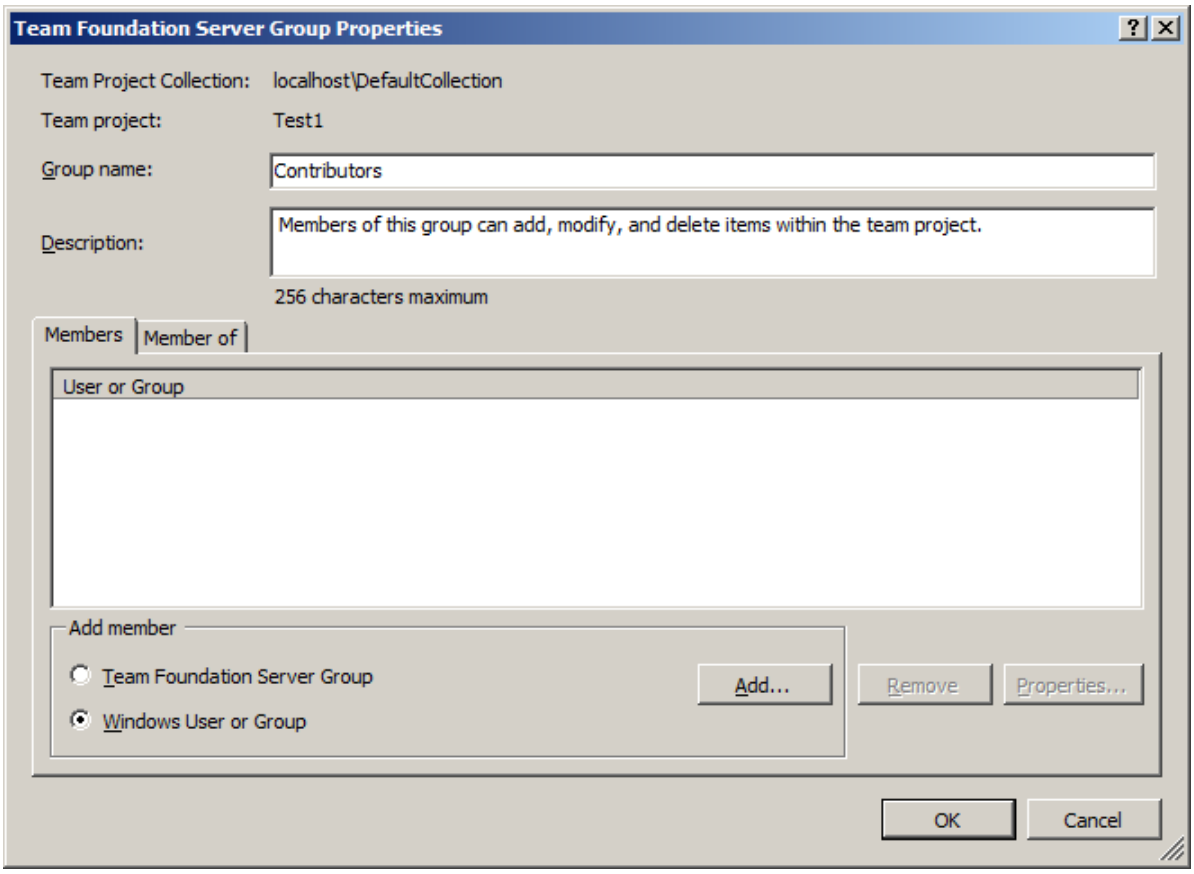
1. W oknie Team Explorer, prawym klawiszem myszy, klikamy na projekcie Test1 (do niego chcemy dodać użytkowników) – pojawi się okno z menu (Rysunek 13.). Wskazujemy pozycję Team Project Settings i klikamy Group Membership – pojawi się okno dialogowe Project Groups on Test1 (Rysunek 19.).
2. Wybieramy grupę, do której chcemy dodać użytkownika, przy czym:

   a) Test1\Readers – to grupa użytkowników o najmniejszych uprawnieniach.
   b) Test1\Contributors – to grupa użytkowników będących współpracownikami.
   c) Test1\Builders – to grupa użytkowników mogących zarządzać budowaniem aplikacji.
   d) Test1\Project Administratos – to grupa użytkowników o największych uprawnieniach.



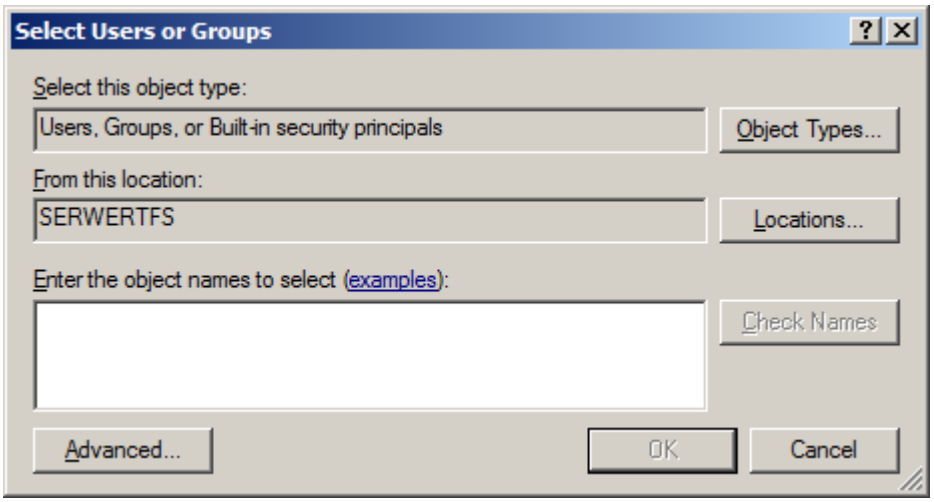
Rysunek 19. Grupy uprawnień w ramach projektu Test1.

3. W naszym przypadku możemy dodać kolejnych użytkowników do grupy Contributors, wybieramy ją i naciskamy na przycisk Properties, pojawi się kolejne okno Team Foundation Server Group Properties (Rysunek 20.), które pozwala zarządzać członkami grupy i innymi własnościami.



Rysunek 20. Okno właściwości grupy serwera TFS.

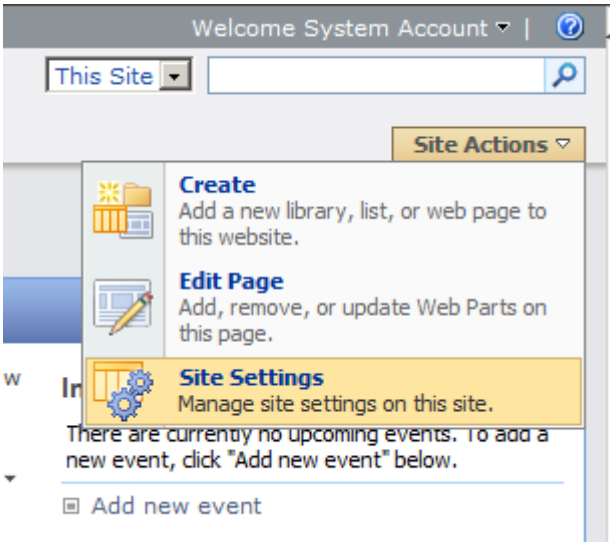
4. W ramach podgrupy Add Member, wybieramy Windows User or Group, a następnie klikamy Add, pojawia się standardowe okno Select Users or Groups (Rysunek 21.).
5. W ramach okienka Enter the object name to select wpisujemy nazwy użytkowników oddzielonych średnikami – możemy na koniec nacisnąć Check Names, co spowoduje, że system sprawdzi, czy są to poprawni użytkownicy.
6. Naciskamy przycisk OK, by zamknąć pierwsze okno dialogowe i kolejny raz OK, by zamknąć drugie okno dialogowe.
7. Ostatecznie przyciskiem Close zamykamy okno Project Groups.



Rysunek 21. Standardowe okno wyboru użytkowników lub grup.

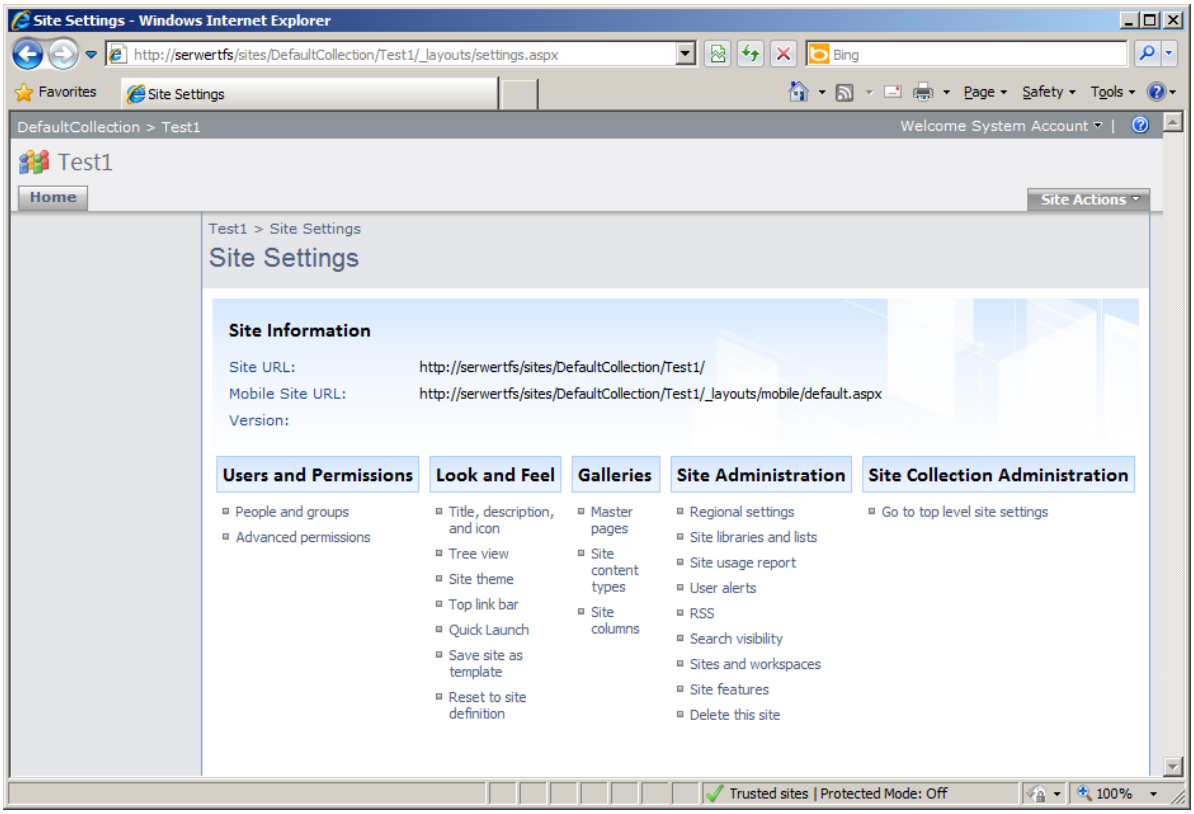
Dodawanie użytkownika w SharePoint

1. W oknie Team Explorer, prawym klawiszem myszy, klikamy na projekcie Test1 – pojawi się okno z menu (Rysunek 13.). Klikamy pozycję Show Project Portal – pojawi się okno przeglądarki z witryną SharePoint dla projektu Test1 (Rysunek 14.).
2. W prawym górnym rogu strony znajduje się przycisk Site Actions – klikamy go, a następnie Site Settings (Rysunek 22.).

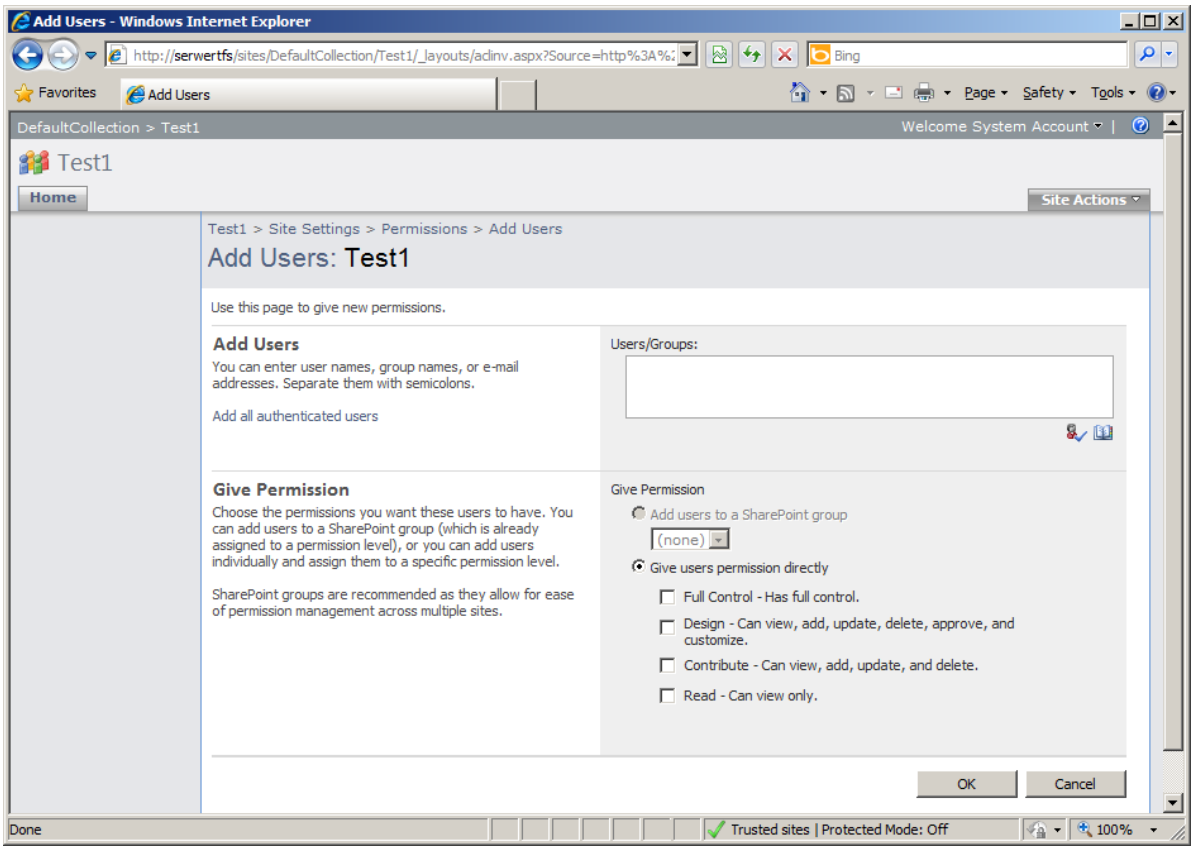


Rysunek 22. Menu Site Actions na witrynie SharePoint projektu.

3. Na stronie ustawień (Rysunek 23.) klikamy People and Groups, a następnie w menu New klikamy Add Users, ostatecznie pojawia się okno dodawania użytkowników i określania ich uprawnień (Rysunek 24.) Wpisujemy tutaj nazwę użytkownika oraz nadajemy mu pożądane uprawnienia – w naszym przypadku, co najmniej Contribute.



Rysunek 23. Strona ustawień serwisu SharePoint dla projektu Test1.

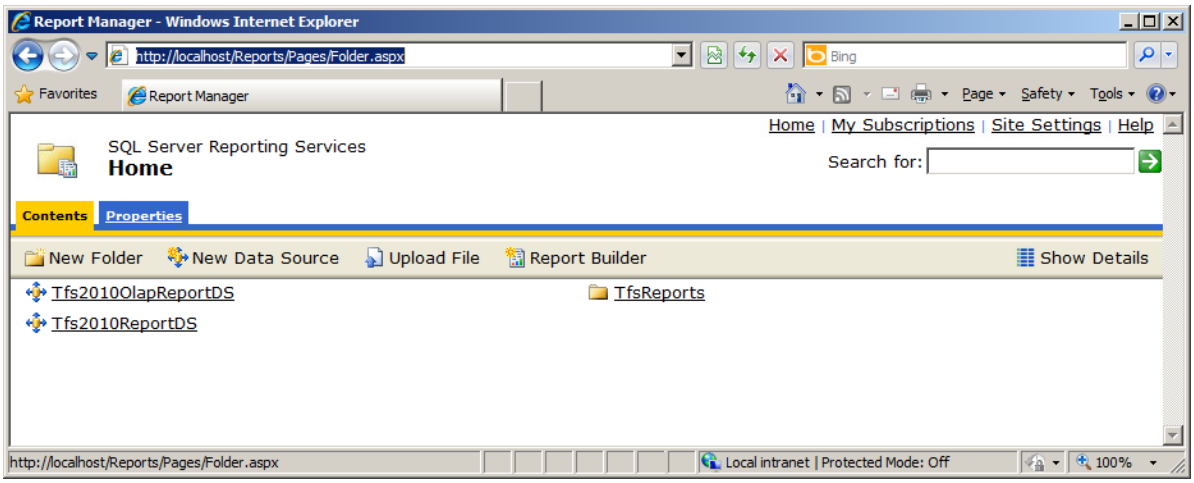


Rysunek 24. Podstrona serwisu SharePoint dla projektu Test1 pozwalająca dodać użytkownika i określić jego uprawnienia.

4. Naciskamy OK i kończymy nadawanie uprawnień dla witryny projektu.

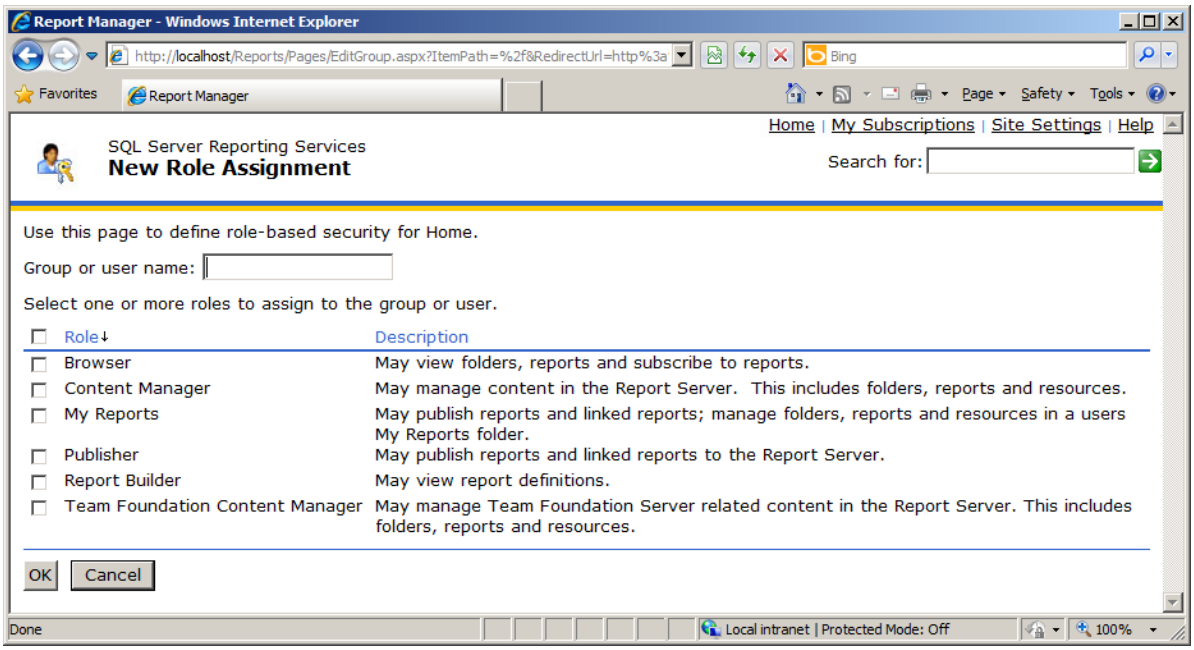
Dodawanie użytkownika w Reporting Services

1. Otwieramy przeglądarkę i wpisujemy adres http://localhost/Reports/Pages/Folder.aspx, lub zamiast localhost pełną nazwę serwera (jeśli nie pracujemy lokalnie), pojawi się strona serwera raportów (Rysunek 25.).



Rysunek 25. Strona serwera raportów.

2. Naciskamy zakładkę Properties, a następnie w menu wybieramy New Role Assignment, pojawia się strona, pozwalająca określić nazwę użytkownika oraz jego uprawnienia (Rysunek 26.).

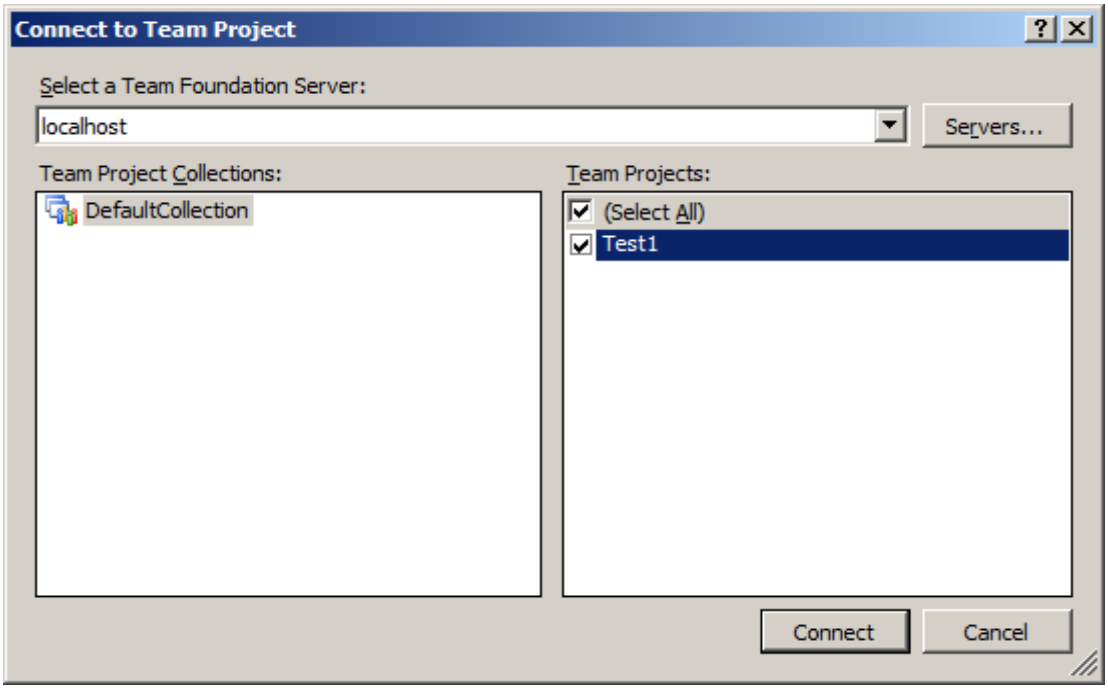


Rysunek 26. Strona serwisu raportów pozwalająca dodać użytkownika i nadań mu uprawnienia.

3. Dla naszych potrzeb wystarczy nadać uprawnienia Browser, a następnie zatwierdzić przyciskiem OK.

Elementy pracy grupowej

W celu przetestowania poprawności uprawnień i stwierdzenia, że środowisko jest gotowe do pracy, przełączmy się na użytkownika, dla którego nadawaliśmy uprawnienia w poprzedniej sekcji. Po zalogowaniu, uruchamiamy VS 2010, oraz podłączamy się do TFS 2010 – kroki analogiczne, jak na początku poprzedniej sekcji. Z tą różnicą, że po wybraniu serwera lokalnego pojawi się okno, w którym będzie dodatkowo lista projektów w ramach kolekcji domyślnej i na tej liście będzie znajdował się projekt Test1, zaznaczmy iż do niego chcemy się podłączyć i wciśnijmy przycisk Connect (Rysunek 27.).



Rysunek 27. Okno podłączania do serwera TFS wraz z listą projektów na nim umieszczonych.

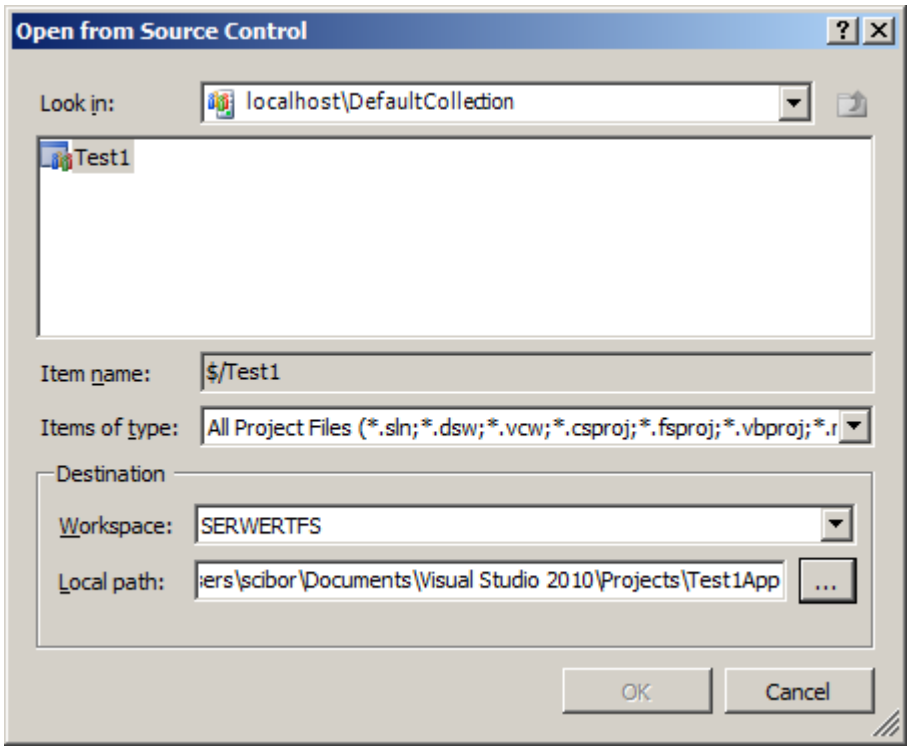
Jeśli w poprzedniej sekcji nadaliśmy prawidłowo uprawnienia użytkownikowi, na którym obecnie pracujemy to nastąpi podłączenie do TFS i drzewo projektu Test1 zostanie wyświetlone w oknie Team Explorer. Krok ten zapewnił, że mamy połączenie z projektem i widzimy np. prace, czy raporty, jednak jeśli spojrzymy do zakładki Solution Explorer to jest pusto – kod aplikacji nie został automatycznie wczytany.

W celu zaczytania drzewa plików aplikacji Test1App musimy w menu File wejść w podmenu Source Control, a następnie wybrać Open from Source Control – otworzy się okno dialogowe (Rysunek 28.). W oknie tym możemy wybrać kolekcję, z której możemy otwierać projekty – w naszym przypadku pozostawiamy bez zmian – następnie poniżej kolekcji widzimy projekt Test1, jest to katalog, do którego musimy zejść na tyle głęboko, by wreszcie zobaczyć plik Test1App.sln. Plik Test1App.sln jest plikiem zawierającym opis projektu z punktu widzenia Visual Studio, zaznaczamy go i teraz musimy wpisać, lub wybrać, lokalną ścieżkę gdzie zostanie umieszczona kopia plików z serwera. Jeśli wszystko wypełniliśmy prawidłowo przycisk OK będzie aktywny i możemy go nacisnąć. Gdy to uczynimy wszystkie niezbędne pliki zostaną wczytane z repozytorium i w oknie Solution Explorer pojawi się drzewo plików projektu (podobne jak na rysunku 16). Zauważmy, że na drzewie obok plików pojawiają się niebieskie kłódki, oznacza to, że elementy tak oznaczone znajdują się w stanie Checked-in, czyli są zatwierdzone i są zgodne z tym, co jest na serwerze.

Może pojawić się teraz naturalne pytanie, jak zacząć pracę na naszej kopii plików, otóż możemy całe drzewo pobrać dla siebie do edycji (prawy klawisz na nazwie projektu a następnie z menu należy wybrać Check-Out for Edit), ale zwykle nie modyfikujemy wszystkich plików, zatem nie ma to sensu. Zamiast tego możemy wykonać analogiczną operację dla konkretnego pliku, jaki chcemy edytować. W obu przypadkach mamy możliwość wybrania trybu, w jakim następuje blokada pliku:

- **Unchanged** – bez zmian, czyli pozostawiamy obecną blokadę.
- **Check Out** – najsilniejsza blokada, uniemożliwia innym użytkownikom wykonania operacji zarówno check out jak i check in.
- **Check In** – umożliwia pozostałym użytkownikom wykonanie operacji check out, ale nie mogą wykonać check in.

Jeśli nie jesteśmy pewni, co zrobić i nie rozumiemy do końca tego mechanizmu, zdecydowanie wybierzmy opcje Unchanged. Natomiast nie musimy ręcznie wykonywać Check Out, wystarczy, bowiem w Visual Studio zacząć edytować plik, by automatycznie został ona dla nas udostępniony do edycji.



Rysunek 28. Okno pozwalające na otwarcie projektu z repozytorium kodu.

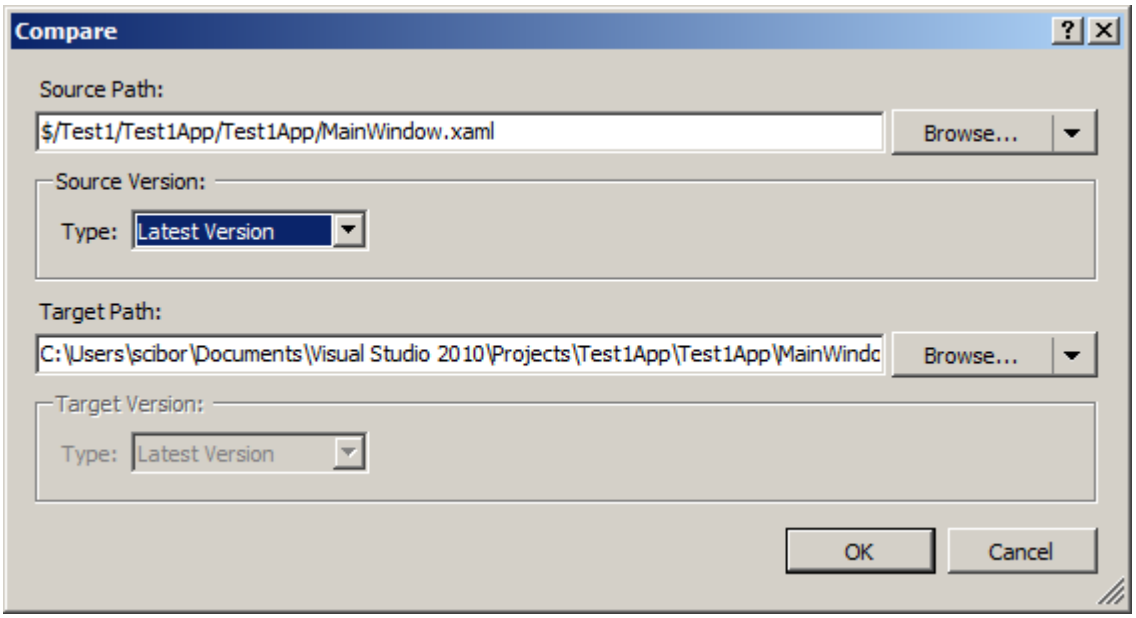
W tym celu w oknie XAML (Rysunek 29.) zamieńmy tytuł głównego okna z domyślnego MainWindow na Test1App.

Rysunek 29. Okno edycji pliku XAML.

Zauważmy, że po zmianie tytułu – tak naprawdę dowolnej edycji wybranego pliku – w oknie Solution Explorer nastąpiła zmiana i obok pliku MainWindow.xaml, zamiast kłódki pojawił się czerwony ptaszek, co oznacza, że plik ma status Checked Out. Jednocześnie w oknie Pending Changes plik ten pojawił się, jako oczekująca zmiana.

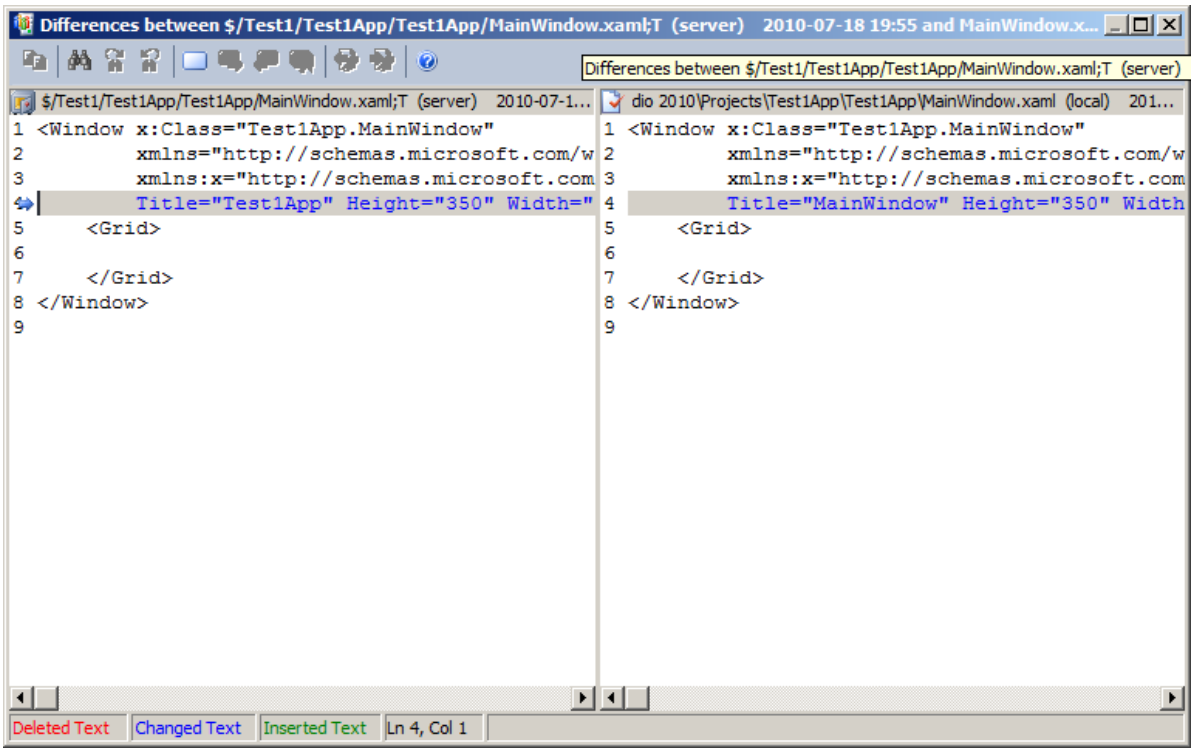
Warto zobaczyć jak ten plik będzie widoczny dla innego członka zespołu, w tym celu przełączmy się na Administratora i otwórzmy projekt w Visual Studio ponownie ten projekt (na oknie powitalnym będzie on w sekcji Recent Projects). W oknie Solution Explorer przy tym pliku pojawi się ikona przedstawiająca człowieka, co oznacza, że plik ten został pobrany do edycji przez inną osobę.

Na sam koniec, tego wprowadzenia do pracy grupowej, warto zwrócić uwagę jeszcze na dwie funkcje, pierwsza to możliwość porównania lokalnej wersji pliku z np. z ostatnią umieszoną na serwerze (Rysunek 30.) – by wywołać to okno najeżdżamy kursorem myszy na interesujący nas plik, naciskamy prawy klawisz i z menu wybieramy Compare.



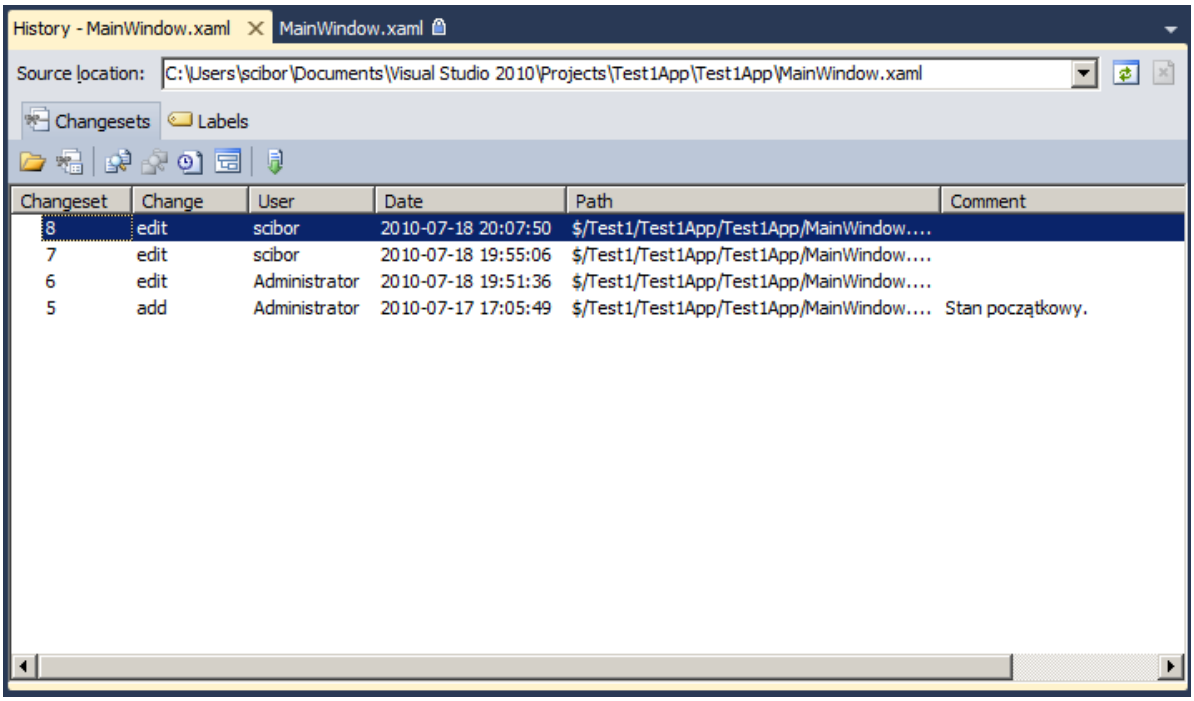
Rysunek 30. Okno dialogowe pozwalające określić parametry porównywania dla pliku.

Po określeniu naszych preferencji i naciśnięciu przycisku OK, otrzymujemy okno pozwalające porównywać wszystkie zmiany pomiędzy plikami (Rysunek 31.).



Rysunek 31. Okno zmian w pliku.

Druga bardzo przydatna funkcja to możliwość podglądu historii danego pliku, w tym celu znów najeżdżamy kursorem myszy na interesujący nas plik, naciskamy prawy klawisz myszy i z menu wybieramy View History, co powoduje wyświetlenie okna historii (Rysunek 32.). Widać na niej numer wersji (Changeset), typ zmiany, użytkownika dokonującego zmianę, datę wraz z czasem, ścieżkę na serwerze i komentarz.



Rysunek 32. Przykładowa historia zmian jednego z plików.

Pozostałe narzędzia pracy grupowej

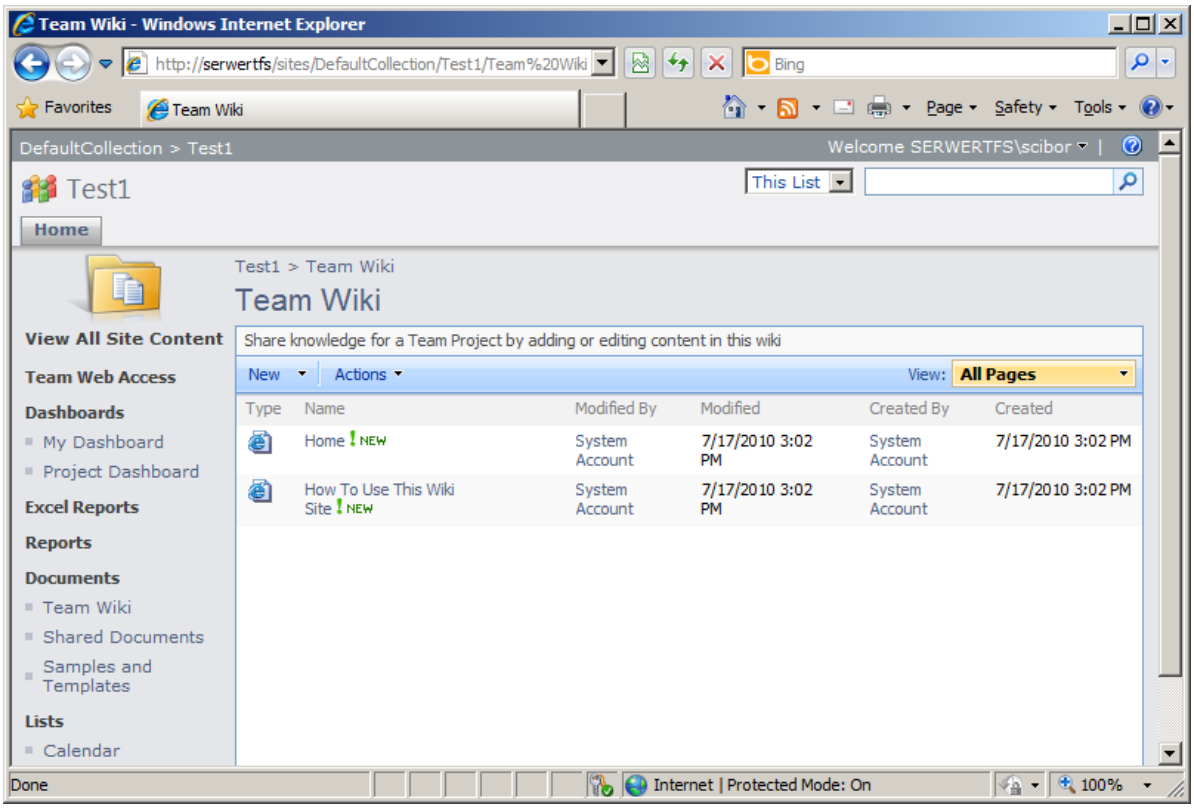
TFS 2010 wraz z usługami SharePoint daje dodatkowe możliwości dla zespołu projektowego, które często u innych dostawców występują oddzielnie, a tu są dobrze ze sobą zintegrowane. Jak pisaliśmy wcześniej, oprócz serwera kontroli wersji należy zespół wyposażyć co najmniej w dwa dodatkowe narzędzia. Przypomnijmy, iż było to repozytorium plików projektowych (nie będących plikami kodu) – zatem zespół powinien mieć możliwość korzystania np. z portalu umożliwiającego współdzielenie plików, oraz miejsce wymiany wiedzy i doświadczeń, również najlepiej w postaci wspólnego portalu.

Jak już zaznaczyliśmy TFS 2010 wraz z usługami stowarzyszonymi dostarcza tych dwu rozwiązań, a nawet znacznie więcej w jednym miejscu – portal projektu. Jeśli przywołamy jeszcze raz główne okno portalu projektu (Rysunek 14.) to widzimy z lewej strony linki do dwu interesujących nas zasobów: Team Wiki (Rysunek 33.) oraz Shared Documents (Rysunek 34.). Obie te strony są podstronami w ramach portalu, zatem są widoczne dla uprawnionych członków zespołu i umożliwiają wygodną pracę nad nimi.

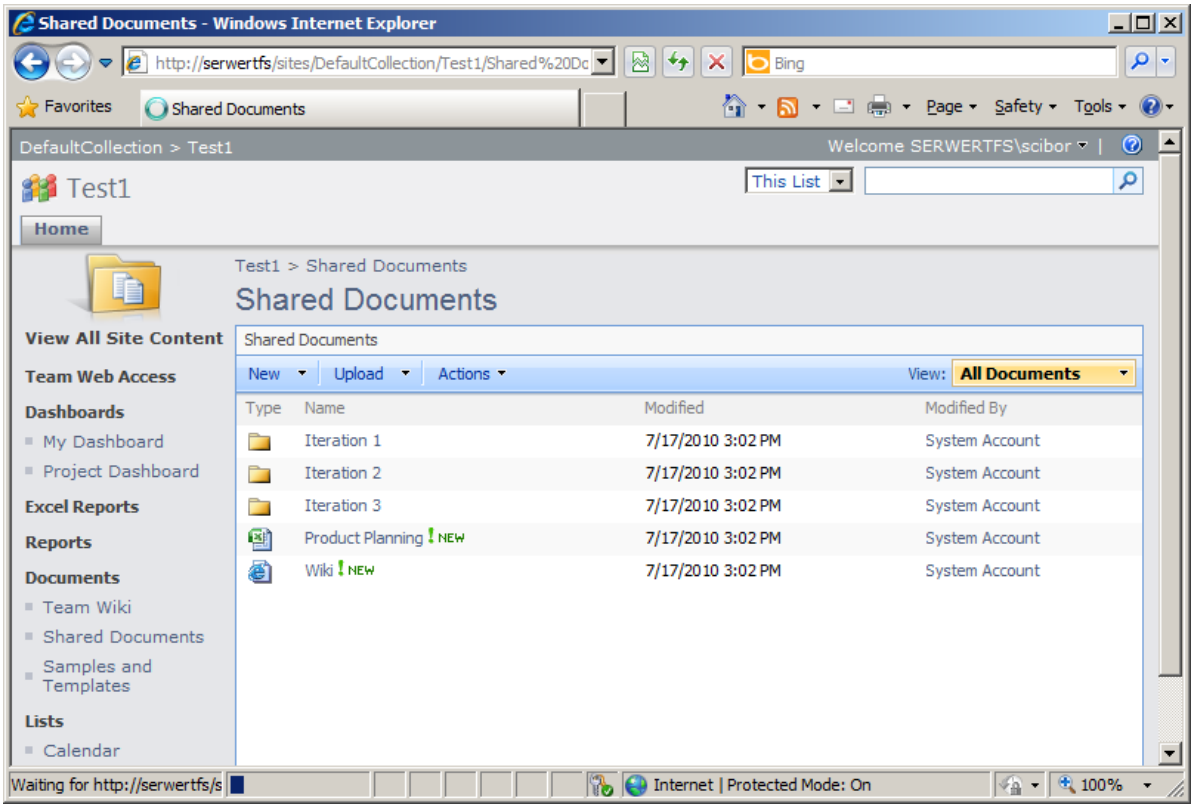
Pozostałe elementy, na które warto zwrócić uwagę, to:

- Strona My Dashboard, która grupuje: zadania, błędy i przypadki testowe dla danego użytkownika.
- Calendar, gdzie pojawiają się wydarzenia.
- Reports – grupująca raporty.
- Excel reports – grupująca raporty, które możemy obejrzeć za pomocą programu Excel.

Część z tych elementów będziemy omawiali w miarę potrzeb w przyszłych modułach.



Rysunek 33. Strona Wiki zespołu.



Rysunek 34. Strona współdzielonych dokumentów.

Porady praktyczne

Przy pracy w oparciu o serwerze kontroli wersji, warto pamiętać o kilku rzeczach:

1. Zawsze nadawać adekwatny komentarz do zmiany umieszczanej na serwerze – pomoże to później w historii zmian wyszukać interesujące nas wersje. Proszę zobaczyć, że zmiany bez komentarza nic nie mówią w oknie historii (Rysunek 32.).
2. Nie wprowadzać zmian, które są niekompletne w tym znaczeniu, że na przykład nie kompiluje się program, lub działa błędnie i o tym wiemy. Jeśli z jakiś przyczyn chcemy umieścić takie zmiany na serwerze, to połóżmy ją na własną półkę (Shelve).
3. Unikać pracy kilku programistów na tym samym pliku na raz, gdyż może to prowadzić do niemożności automatycznego połączenia zmian (o czym pisaliśmy wcześniej), a nawet w przypadku ręcznego łączenia do błędów.
4. W przypadku wprowadzania zmian należy unikać zatwierdzania kilku zmian w trakcie jednej operacji Check-In, gdyż prowadzi to do trudności w późniejszym odnajdywaniu różnic. Przykładowo, mamy do poprawy dwa błędy, które wymagają zmian w pliku1 dla pierwszego błędu i pliku2 dla drugiego oraz pliku3 w przypadku obu błędów. W tej sytuacji najlepiej dokonać poprawek związanych z pierwszym błędem, umieścić je na serwerze (Check In) z właściwym opisem i dopiero dokonać zmian związanych z drugim błędem.

Kilka uwag odnośnie pracy grupowej:

1. Żadne narzędzie, choćby najdoskonalsze, nie zastąpi zdrowego rozsądku i myślenia grupowego, stąd też osoby będące członkami zespołu muszą przestawić się z myślenia indywidualnego/egoistycznego na zespołowe, tylko wtedy uzyskają dobre efekty.
2. Osoby w zespole wymagają wsparcia i rozsądnego przewodnika, taka osoba powinna wskazywać sposób pracy i narzucać kulturę zespołu.
3. Należy pamiętać, że narzędzia bywają zawodne, zatem nie zapominajmy np. o kopiach bezpieczeństwa.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- Wiesz, z jakimi narzędziami wspierającymi zespoły informatyczne można się spotkać w codziennej pracy, w dojrzałych zespołach programistycznych.
- Wiesz, jakie narzędzia w tym zakresie dostarcza firma Microsoft i rozumiesz ich ogólne przeznaczenie.
- Potrafił wykorzystać omówione narzędzia do zainicjowania pracy nad projektem.
- Rozumiał potrzebę i rolę zaawansowanych narzędzi wspierających pracę grupową.

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. http://msdn.microsoft.com/en-us/library/ee789810.aspx

   Strona ta zawiera informację o cechach cyklu życia aplikacji, które są wspierane w poszczególnych produktach Visual Studio 2010.

Laboratorium podstawowe

Zadanie	Tok postępowania
1.	•

Laboratorium rozszerzone

Czas wykonania: od 2 godzin do kilkunastu w zależności od wydajności komputera.

Zadaniem jest stworzenie, od podstaw, środowiska z którym miałeś do czynienia w tym module.

Zadanie	Tok postępowania
1. Sprawdzenie wymagań sprzętowych i systemowych	• Na komputerze na których chcemy dokonać instalacji musimy posiadać system operacyjny, na którym możemy uruchomić narzędzie do wirtualizacji (np. Windows 2008 – narzędzie Hyper-V, Windows 7 – narzędzie Virtual PC). • Komputer na którym będziemy instalowali całe środowisko musi posiadać co najmniej 4 GB RAM, i około 50 GB wolnego miejsca na dysku.
2. Instalacja Windows Server 2008	• Pobieramy z sieci obraz płyty instalacyjnej systemu Windows 2008 Server Enterprise 32-bit i zapisujemy lokalnie (adres obrazu: http://www.microsoft.com/downloads/details.aspx?FamilyId=13C7300E-935C-415A-A79C-538E933D5424&displaylang=en). • W ramach wybranego środowiska wirtualizacyjnego tworzymy nowy system operacyjny, jako nośnik wskazujemy ściągnięty obraz płyty instalacyjnej, rozmiar dysku możemy pozostawiać jako zmieniany dynamicznie, koniecznie określamy co najmniej 2GB RAM dla systemu gościa. • Wykonujemy standardową instalację systemu Windows Server 2008 (przykładowe źródło dokumentacji procesu instalacji: http://www.microsoft.com/downloads/details.aspx?FamilyId=1087A498-40AD-46BA-9ADA-F32A58A94A85&displaylang=en).
3. Konfiguracja Windows Server 2008	• Ustawiamy hasło administratora na takie, by było znane i wygodne (to tylko maszyna testowa) i jednocześnie spełniające reguły poprawności np. P@ssw0rd (trzeci znak od końca to zero). • Wykonujemy aktualizację systemu (wszystkie zalecane i obowiązkowe poprawki).
4. Dodanie roli Web Server	•
5. Instalacja Microsoft SQL Server 2008	• Pobieramy z sieci obraz płyty instalacyjnej Microsoft SQL Server 2008 Enterprise 32-bit i zapisujemy lokalnie (adres obrazu: http://www.microsoft.com/downloads/details.aspx?FamilyID=265f08bc-1874-4c81-83d8-0d48dbce6297&displaylang=en). •

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 4

Wersja 1

Wstęp do Scrum

Spis treści

Informacje o module

Opis modułu

W tym module znajdziesz omówienie jednej z najpopularniejszych metodyk lekkich zarządzania projektami – Scrum. Metodyka ta powstała w 1999 roku i od tamtej pory zyskała ogromna popularność. Charakteryzuje ją prostota oraz iteracyjny charakter, który powoduje iż dobrze wpasowuje się w obecne potrzeby świata informatycznego. szczególnie, kiedy Klient nie umie od razu określić wszelkich potrzeb związanych z przyszłym systemem i są one pozyskiwane etapowo. W module tym znajduje się jedynie skrócony opis tej metodyki, jednak wybrano te elementy, które są najważniejsze i wystarczające do rozpoczęcia pracy. Metodyka ta obecnie jest jedną z preferowanych w środowisku Visual Studio 2010, w którym to zostanie zaprezentowany przykład jej użycia.

Cel modułu

Celem niniejszego modułu jest zapoznanie czytelnika z podstawowymi informacjami o metodyce Scrum.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- Wiedział, jakie są podstawowe pojęcia metodyki Scrum i umiał wykorzystać wiedzę o tej metodyce w trakcie tworzenie projektu w Visual Studio 2010,
- Wiedział, jak utworzyć projekt w oparciu o szablon Scrum w środowisku Visual Studio 2010 i TFS 2010,
- potrafił zainicjować projekt przy użyciu metodyki Scrum i określić role Scrum oraz potrafił przeprowadzić początkowe prace,
- rozumiał znaczenie ról, iteracji, Sprintu i pozostałych pojęć metodyki Scrum.

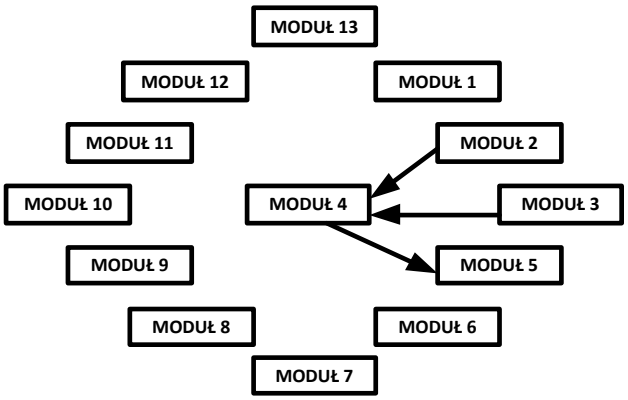
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawową obsługę programu Visual Studio 2010,
- znać sposób utworzenia projektu w oparciu o TFS 2010,
- znać podstawy zarządzania użytkownikami projektu w Visual Studio 2010, TFS 2010 oraz SharePoint Services,
- znać podstawy pracy w środowisku Visual Studio 2010,
- rozumieć pojęcia ról w zespole programistycznym,
- rozumieć podstawowe pojęcia inżynierii oprogramowania np. cykl życia oprogramowania.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w modułach 2 i 3



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

W ciągu ostatnich 3 tygodni Ty i twój kolega z zespołu, próbowaliście uzgodnić z klientem zakres prac. Nauczono Was, że wstępna analiza wymagań jest najważniejsza i bez niej nie wolno zacząć prac, co więcej nie można podpisać umowy, stąd też próbujecie tak dokładnie jak to możliwe dowiedzieć się od klienta, czego on tak naprawdę oczekuje, co więcej, ponieważ nie chcecie się pomylić w wyliczeniu wartości umowy, to dopytujecie o najdrobniejsze szczegóły – jak mają wyglądać poszczególne okna dialogowe, formularze a nawet raporty.

Ostatecznie wspólnie z klientem zagoniliście się w ślepy zaułek, gdyż Wy, jako zespół wykonawczy, chcecie wszystko precyzyjnie wycenić, z drugiej strony klient na tym etapie umowy (której de facto jeszcze nie ma) ma już Was serdecznie dosyć, gdyż w przeciągu 3 tygodni spotkaliście się z nim 8 razy i łącznie przegadaliście 24 godziny, co Klient zaczyna powoli traktować jako stratę czasu.

Najgorsze jest to, że minęło 3 tygodnie, Klient nie widział nic poza stosem papierów, a Wy zmarnowaliście ponad tydzień pracy dwu osób (32 godziny na spotkaniach, dojazdy i czas poświęcony na analizę we własnym gronie). Zaczyna do Was docierać, że jeśli czegoś nie zrobicie, to może okazać się, że klient zrezygnuje, a Wy stracicie kontakt.

Ponieważ wydaje Wam się, że cała praca nie powinna zająć więcej niż 3-4 miesiące i spokojnie wykonacie ją niewielkim, powiedzmy, 4 osobowym zespołem, więc być może jednak przesadziliście z przeprowadzaniem tej wstępnej analizy zamieniając ją w szczegółową analizę, która zwykle ma miejsce w modelu kaskadowym lub przy dużych projektach o szerokim zakresie prac. Zadajecie sobie pytanie czy to jest właściwe podejście? Przecież klient Was zna, deklaruje chęć współpracy, nie oczekuje, że wycenicie wszystko z dokładnością do grosza, co więcej sam kilkukrotnie mówił, że chętnie pracowałby jakimiś etapami.

W tym momencie przypominacie sobie, że kiedyś Wam mówiono o tzw. metodykach lekkich, co prawda głównie dotyczyło to metodyk lekkich wytwarzania oprogramowania jak np. XP, ale ktoś kiedyś wspomniał o takich metodykach kierowania projektami informatycznymi, w tym o Scrum, ale jak to było?

Podstawy teoretyczne

Podstawowe informacje o Scrum

Scrum jest jedną z najpopularniejszych metodyk lekkich (zwanych też zwinnymi – ang. Agile) zarządzania projektami, która została opracowana około 1993 roku przez Kena Schwabera i Jeffa Sutherlanda. Jak większość metodyk tego typu posiada iteracyjny charakter, oznacza to w efekcie, iż docelowy projekt powstaje metodą kolejnych przybliżeń. Jak każda metodyka posiada swoje zalety jak i wady, spróbujmy wymienić wpierw zalety:

1. W założeniu iteracyjny charakter metodyki pozwala na uściślanie założeń w trakcie realizacji, w efekcie klient może określić jego główne wymagania, a w miarę rozwoju doprecyzowywać to co konieczne. W efekcie w założeniu pozwala na wczesne uzyskanie istotnych elementów - oczywiście najwcześniej po pierwszej iteracji.
2. Często podkreślane jest, iż dzięki tej metodyce zarówno klient jak i wykonawca mają pełną kontrolę nad projektem - gdyż na końcu każdej iteracji można zrezygnować z części funkcjonalności jak i dodać nowe, a nawet w przypadku stwierdzenia, że projekt stracił uzasadnienie zrezygnować z niego w ogóle.
3. Według źródeł Scrum sprawdza się zarówno w małych zespołach jak i dużych organizacjach. Należy przy tym pamiętać, że metodyka ta podobnie jak wiele innych (szczególnie zwinnych) bazuje na pracy zespołowej, komunikacji, zaufaniu w zespole i wzajemnym więzom.

4. Metodyka ta dostarcza kierownikowi zespołu wielu technik do śledzenia postępu prac i stopnia zaawansowania projektu.

Jeśli zapytamy o wady Scrum, to właściwie wszystkie zalety mogą stać się również wadami, jeśli spojrzymy na nie niejako od drugiej strony. I tak odpowiednio:

1. Iteracyjny charakter tej metodyki powoduje, że jest bardzo trudno oszacować całkowity czas realizacji, ilość osób potrzebnych do realizacji (w każdej iteracji mogą występować np. eksperci czy konsultanci) i co za tym idzie ostateczny koszt przedsięwzięcia. Zatem to, co stanowi największą siłę tej metodyki, często jest również pokazywane jako jej największa wada. Niestety większość Klientów chciałaby znać budżet przedsięwzięcia, a tego w przypadku tej metodyki nie da się wyliczyć wprost.
2. Fakt możności odstąpienia od umowy po każdej iteracji jest niewątpliwie ważnym aspektem, jednak musi być właściwie skonstruowana umowa, by ani klient ani wykonawca w tym momencie nie poniósł nieuzasadnionych strat.
3. Dużo przeciwników Scrum jako wadę wymienia silną zależność od ludzi w zespole, jednak należy zauważyć, że to jest ogólnie cecha projektów informatycznych i właściwie niezależnie od stosowanej metodyki zarządczej będzie występować.
4. Jest to jedyny punkt, w którym trudno znaleźć krytykę w literaturze i praktyce.

Jak każda metodyka, tak i Scrum definiuje własne pojęcia, można je podzielić na trzy zasadnicze grupy: role, procesy i narzędzia. W ramach tej metodyki wyróżnione są jedynie trzy role (zaleca się, by osoba nie podejmowała się więcej niż jednej roli jednocześnie):

1. **Właściciel Produktu** (Product Owner) - reprezentuje on interesy wszystkich ludzi zaangażowanych w projekt oraz zainteresowanych finalnym produktem. Jego odpowiedzialność jest bardzo szeroka i sprowadza się do kluczowych decyzji związanych z produktem (np. docelowa funkcjonalność), ustalania priorytetów. Zatem mówiąc inaczej jest on odpowiedzialny zarówno za finalny kształt produktu, jak i terminy implementacji poszczególnych funkcji.
2. **Szef Scrum**, czasami zwany też Mistrz Scruma (Scrum Master) - jeśli znamy którąś z metodyk "klasycznych", to tu można nas spotkać rozczarowanie, gdyż Szef Scruma jest odpowiedzialny jedynie za dwie rzeczy: egzekwowanie praktyk i zasad Scruma, oraz za zapewnienie dobrych warunków pracy zespołu i usuwanie przeszkód. Zatem jego rola jest zdecydowanie mniejsza niż "klasycznego" Kierownika Zespołu.
3. **Członek Zespołu** (Team Member) - członkiem zespołu jest każda osoba należąca do grupy wykonującej czynności związane bezpośrednio z wytwarzaniem finalnego produktu. Jak pamiętamy z pierwszego modułu mówiliśmy o programistach, administratorach, testerach itd. - tutaj wszyscy oni są po prostu członkami zespołu. Co więcej w Scrum unika się etykietowania ludzi, zespół ma być elastyczny, lub jak to się określa samo-organizujący (do tego pojęcia jeszcze wrócimy).

W dalszej części zdefiniujmy główne procesy i narzędzia (pozostałe pojęcia zostaną wprowadzone w następnym punkcie), i tak:

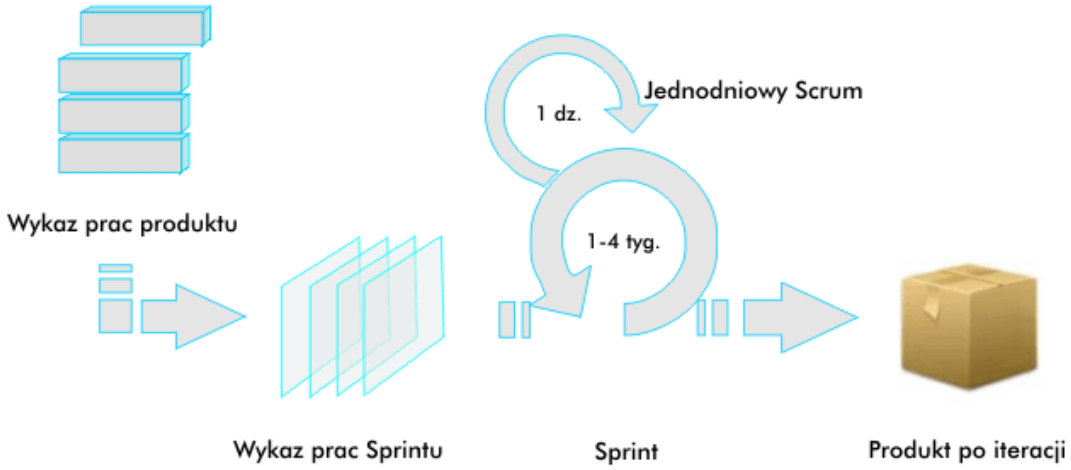
- **Wykaz prac produktu** (ang. Backlog) - jest listą prac, jakie należy wykonać, przy czym zakłada się, że każda praca powinna być wykonana w czasie 4-6 godzin. Lista tych prac nie musi być od razu kompletna i może być modyfikowana zarówno przez rezygnację z pewnych prac, jak i dokładanie nowych. Modyfikacji może dokonać Właściciel Produktu.
- **Sprint** - jest inną nazwą na pojedynczą iterację, zaleca się by pojedynczy sprint trwał 30 dni, ale zespół może ustalić również krótszy czas trwania. Należy dbać jedynie o to, by czas trwania pierwszego sprintu był utrzymany dla pozostałych.
- **Wykaz prac sprintu** - jest to lista prac (wybranych z wykazu prac produktu), które Właściciel Produktu wybrał jako istotne do realizacji w danej iteracji.

- **Cel sprintu** - określa biznesowe uzasadnienie dla tego konkretnego sprintu. To pojęcie jest bardzo ważne i jednocześnie początkowo trudne do zrozumienia. Weźmy prosty przykład, niech uzasadnieniem biznesowym będzie "Optymalizacja ergonomii formularza do wprowadzania faktur", by osiągnąć tę korzyść trzeba wykonać być może kilka, a nawet kilkanaście prac (zgromadzone w ramach Wykazu prac sprintu). Jednak, jeśli członkowie zespołu dostali by wykaz prac bez Celu sprintu mogłoby to utrudnić pracę, gdyż zawsze wykonanie pracy bez "wizji" jest trudne.

Przebieg prac w ramach Scrum

Początkowy etap prac w ramach metodyki Scrum można opisać następującymi krokami (patrz Rysunek 1.):

1. Właściciel Produktu sporządza początkowy wykaz prac produktu oraz budżet.
2. Właściciel Produktu, Szef Scruma oraz pozostali członkowie zespołu podejmują decyzję o czasie trwania pierwszej iteracji (i tym samym wszystkich pozostałych).
3. Planowana jest pierwsza iteracja, zatem kolejno:
   a) Właściciel Produkty określa Wykaz prac sprintu, cel sprintu.
   b) Na podstawie tych informacji członkowie zespołu określają zadania do wykonania, po czym rozchodzą się do swoich zadań.



Rysunek 1. Schemat prac w Scrum.

Powyższe czynności wydają się dość łatwe, jednak w tym miejscu należy zaznaczyć, że Scrum jest bardzo restrykcyjny, jeśli chodzi reżim czasu. Stąd też musimy doprecyzować planowanie iteracji, gdyż jest to bardziej złożony proces, co więcej będzie on powtarzany zawsze na początku każdego sprintu, więc warto go dobrze poznać. Zatem na początku każdego sprintu organizowane jest **spotkanie projektowe**, którego czas nie może przekroczyć ośmiu godzin i podzielone jest na dwie części:

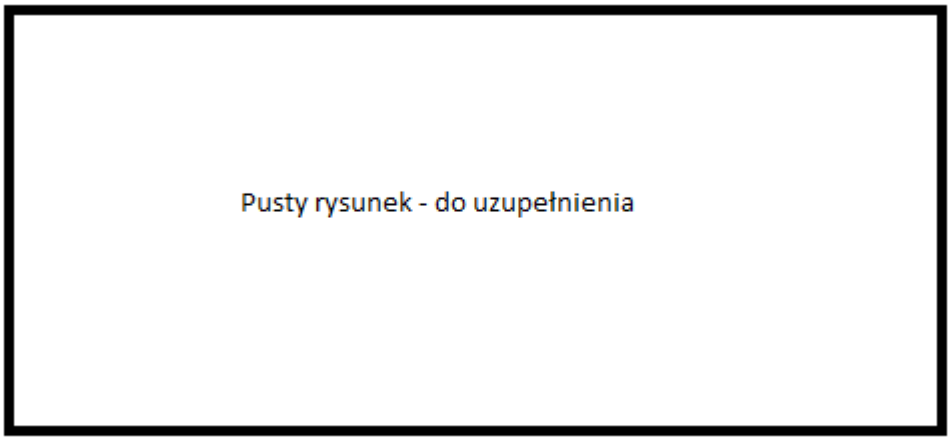
- Pierwsza część spotkania to czterogodzinna sesja planowania, w trakcie której Właściciel Produktu tworzy Wykaz prac sprintu.
- Druga część jest poświęcona podziałowi prac na zadania. W tej części spotkania nie uczestniczy Właściciel Produktu, jednak można zadawać mu pytania.

Jak wcześniej napisaliśmy Scrum jest restrykcyjny, jeśli chodzi o czas, zatem w sytuacji gdy pierwsza część spotkania miałaby się przeciągnąć powyżej 4 godzin ma ona zostać przerwana – takie

podejście ma zapewnić wprowadzenie odpowiedniego rytmu pracy oraz wymuszenie skupiania się jedynie na rzeczach najważniejszych w projekcie, dzięki czemu zespół programistów jak i Klient nie ma tendencji do „dryfowania” z wyznaczonego kursu.

Gdy sprint jest w toku, zespół obowiązkowo musi codziennie, najlepiej przed rozpoczęciem pracy programistycznych, odbyć spotkanie "rozruchowe" (ang. stand-up meeting) zwane również scrumem powszednim, jego czas trwania to maksymalnie 50 minut. W trakcie tego spotkania każdy członek zespołu musi odpowiedzieć na trzy pytania: co robiłeś wczoraj? co będziesz robić dzisiaj? czy coś Ci przeszkadza w osiągnięciu celu? Spotkanie to służy jedynie wymianie informacji, nie zaś rozwiązywaniu problemów technicznych. Kolejną ciekawą praktyką Scruma jest to, iż w tym spotkaniu może uczestniczyć każdy, kto jest zainteresowany projektem. Tu pojawią się znów specyficzne nazewnictwo Scruma, otóż programiści w trakcie tych spotkań nazywani są świnkami, podczas gdy pozostałe osoby - kurczakami. Tylko świnki mają prawo mówić, kurczaki jedynie się przyglądają.

Na koniec każdego dnia pracy, każdy z członków zespołu ma obowiązek zaktualizować **wykres malejący** (ang. burndown chart), który obrazuje ilość prac, jaka pozostała do wykonania w tym sprincie oraz szacować ilość pracy dla zadań, nad którymi członek zespołu pracował w danym dniu (Rysunek 2.).



Rysunek 2. Przykładowy wykres malejący.

Każdy sprint musi zakończyć się czterogodzinnym spotkaniem przeglądu sprintu (ang. spring review meeting), którego celem jest podsumowanie prac nad projektem. Zespół prezentuje Właścicielowi Produktu osiągnięcia tego sprintu, na podstawie tej prezentacji właściciel określa czy Cel sprintu został osiągnięty. Po tym spotkaniu odbywa się trzygodzinne wewnętrzne spotkanie zespołu (bez Właściciela Produktu) nazywane spotkaniem retrospektywnym sprintu (ang. sprint retrospective meeting). W trakcie spotkania odbywa się wewnętrzna konstruktywna krytyka, czyli co zostało źle zrobione, co można poprawić w przyszłości.

Podsumowanie

W tym rozdziale przedstawione zostały przedstawione podstawowe pojęcia metody Scrum. Poznałeś trzy role Scrum: właściciela produktu, szefa Scrum i członka zespołu oraz wiesz, jakie są ich zadania w zespole. Wiesz, jaka jest różnica pomiędzy wykazem prac produktu oraz wykazem prac sprint, co więcej wiesz co to jest sprint i jego cel. Następnie przeczytałeś o przebiegu prac w metodyce Scrum i już wiesz, w jaki sposób zacząć pracę w projekcie zarządzanym tą metodyką. Poznałeś dyscyplinę czasową Scrum i wiesz już jak przeprowadzić spotkanie projektowe oraz jaka jest kolejność dalszych prac.

Przykładowe rozwiązanie

Pierwsze zlecenie

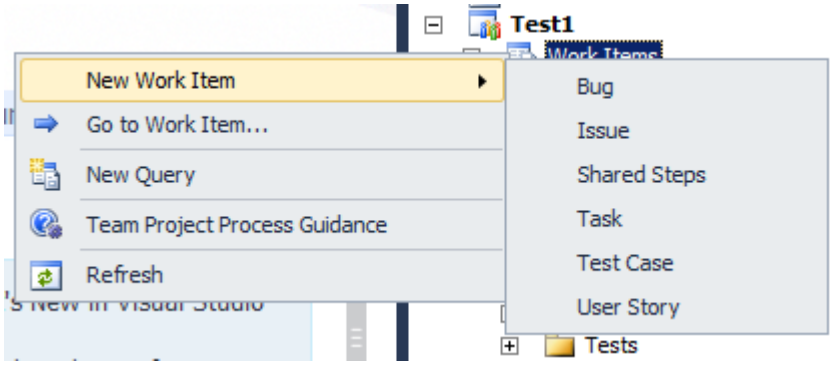
Jak to mówią „nie od razu Rzym zbudowano", zatem zanim zrealizujecie zadanie Klienta, z którym do Was przyszedł, chcecie wypróbować podstawowe zasady Scrum oraz nauczyć się go stosować na czymś prostszym. Tu nadarzyła się okazja, gdyż kolega z młodszych lat studiów nie potrafi poradzić sobie z napisaniem kalkulatora na ćwiczenia z programowania. Co prawda zadanie jest dla Was banalne i chętnie mu wytłumaczycie jak to zrobić, ale przy okazji pomyśleliście, że warto by potraktować ten projekt jako pilotowy i namówiliście by kolega był Waszym Klientem. Niech traktuje Was jako wykonawców i niech precyzyjnie powie o co mu chodzi. Otrzymaliście zatem pierwsze spójne wymaganie „Kalkulator ma posiadać graficzny interfejs użytkownika i klawiaturę przypominająca rzeczywistą, na początek ma on tylko pozwalać wprowadzać cyfry i wyświetlać je na wyświetlaczu" – nazwaliście to wymaganie „Szkielet kalkulatora".

Założenie projektu

Tworzymy projekt według schematu podanego w Module 3, tu podajemy kroki w skrócie:

1. Uruchamiamy Visual Studio 2010 i podłączamy do naszego TFS 2010.
2. W oknie Team Explorer (VS 2010) naprowadzamy kursor myszy na kolekcję, w której chcemy utworzyć nowy projekt (w naszym przypadku będzie to DefaultCollection), naciskamy prawy klawisz myszy. Następnie w menu, które się ukazało, klikamy „New Team Project…".
3. Pojawia się okno dialogowe zawierające kolejne kroki pozwalające utworzyć nowy projekt. Poprzednio (Moduł 3) tworzyliśmy projekt w oparciu o szablon MSF for Agile Software Development v5.0, i w kontekście wcześniejszych ćwiczeń, wydaje się to właściwa droga. Jednak zrobimy inaczej i przerwiemy tworzenie projektu, by rozwinąć ten wątek.

Szablon dostarczany przez Microsoft o nazwie MSF for Agile Software Development v5.0 jest tylko częściowo dostosowany do metodyki Scrum, dla przykładu nazewnictwo nie jest zgodne z tym, które proponuje ta metodyka. Jeśli chcemy się o tym przekonać, to wystarczy utworzyć dowolny projekt w oparciu o szablon MSF for Agile Software Development v5.0 a następnie w drzewie projektu w oknie Team Explorer (nadal w ramach VS 2010), należy najechać kursorem myszy najechać na pozycję „Work Item" a następnie nacisnąć prawy klawisz myszy i najechać na pozycję „New Work Item" pojawi się kolejne menu (Rysunek 3.).



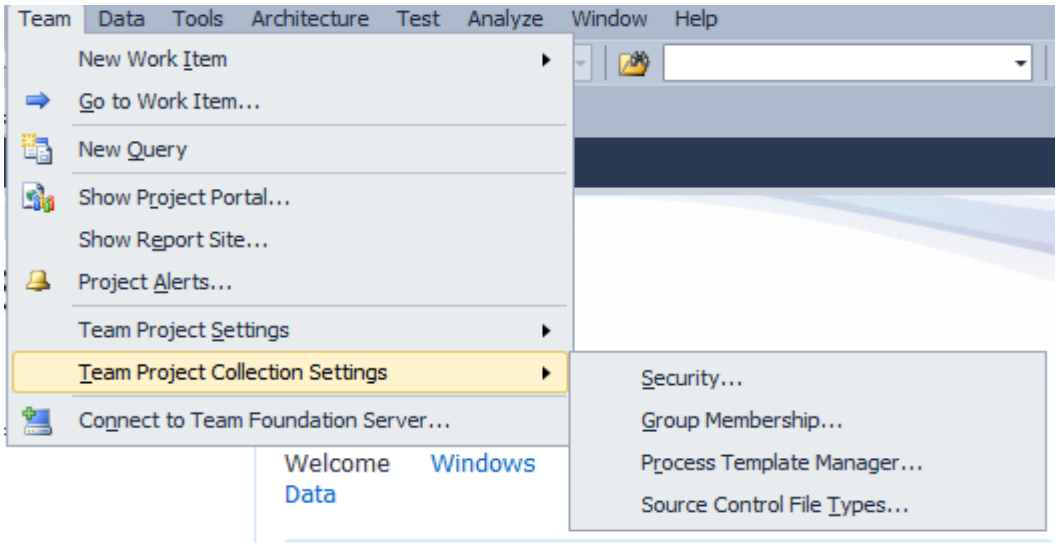
Rysunek 3. Menu kontekstowe dodawania nowego Work Item w ramach szablonu MSF for Agile Software Development v5.0.

Widać wyraźnie, iż nazwy które się pojawiają nie są znane z metodyki Scrum. Tę niedogodność niweluje szablon o nazwie Microsoft Visual Studio Scrum 1.0, który można pobrać ze strony http://visualstudiogallery.msdn.microsoft.com/en-us/59ac03e3-df99-4776-be39-1917cbfc5d8e.
Ten szablon zawiera następujące elementy:

- **Work Item Types** w : Sprint, Product Backlog Item, Bug, Task, Impediment, Test Case

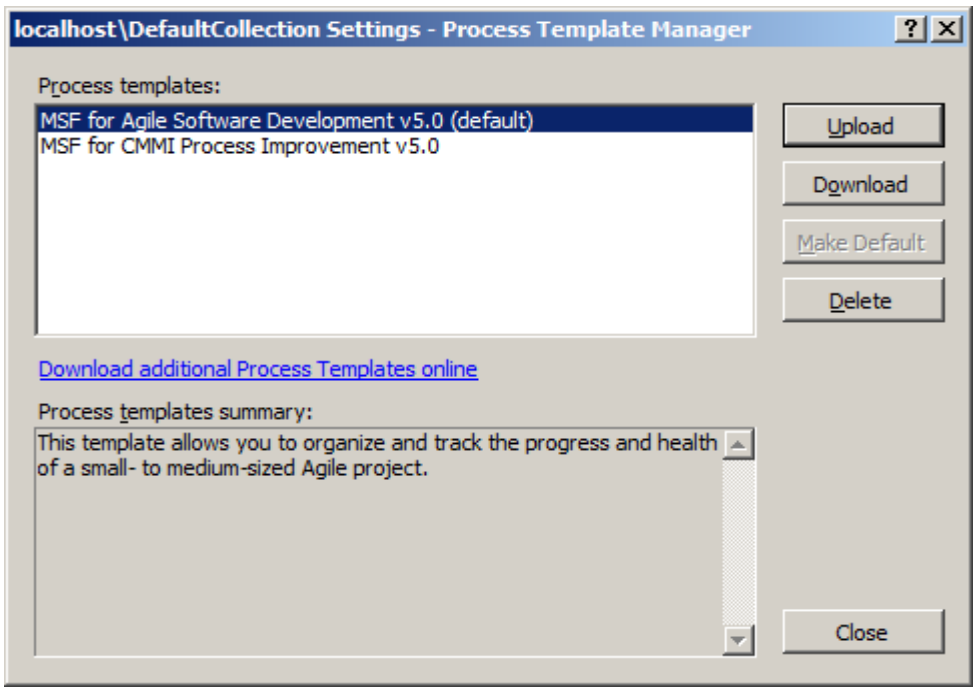
- **Reports**: Release Burndown, Velocity, Sprint Burndown, Build Success Over Time, Build Summary, Test Case Readiness, Test Plan Progress
- **SharePoint Project Portal**

Widać wyraźnie, iż ten szablon przygotowany na potrzeby Scrum, stąd też zanim dalej będziemy inicjować projekt musimy go zainstalować, abyśmy mogli z niego skorzystać. Wykorzystamy w tym celu podany powyżej link do ściągnięcia tego szablonu. Następnie klikamy Download i potwierdzamy chęć uruchomienia pliku. Pojawi się pierwsze okno, postępujemy według poleceń na ekranie: potwierdzamy licencję, wskazujemy lokalizację docelową oraz zaznaczamy, że ma być dostępna dla wszystkich.



Rysunek 4. Widok menu Team z rozwiniętym podmenu ustawień dla kolekcji.

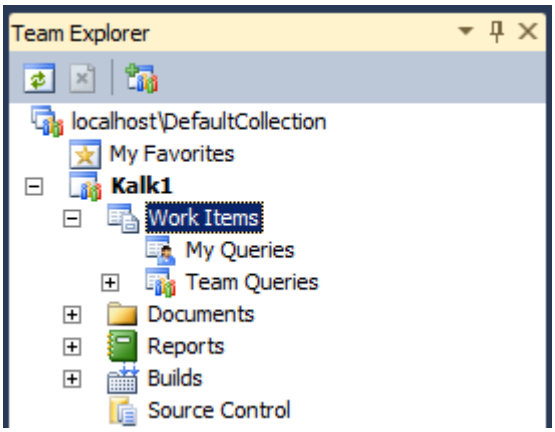
Następnie w menu Team najeżdżamy kursorem mysz na pozycję Team Project Collection Settings – pojawi się kolejne menu jak na Rysunku 4. Ostatecznie klikamy na pozycję Proces Template Manager, to spowoduje otwarcie okna dialogowego jak na Rysunku 5.



Rysunek 5. Okno dialogowe menadżera szablonów procesów.

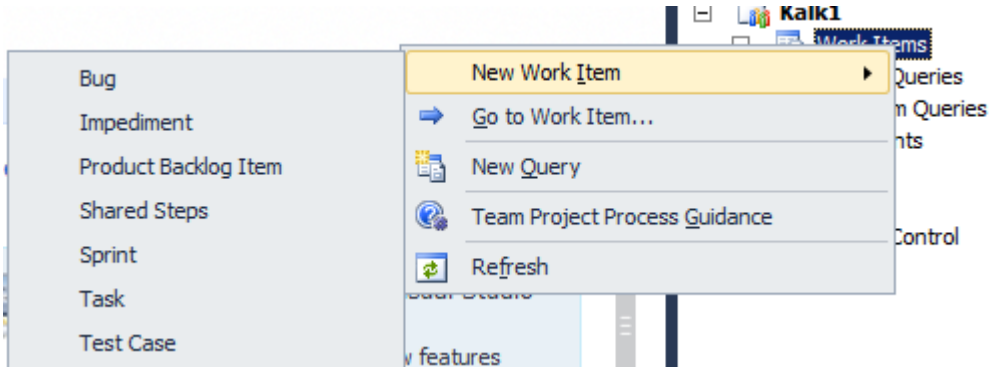
W oknie tym naciskamy przycisk Upload i wskazujemy katalog do którego został rozpakowany szablon (w naszym przypadku C:\Program Files\Microsoft\Microsoft Visual Studio Scrum 1.0\Project Template), jeśli wgrywanie powiedzie się, to otrzymamy komunikat Process Temaplate uploaded successfully i jego nazwa pojawi się w oknie dialogowym z Rysunku 5. Teraz możemy zamknąć to okno dialogowe i wrócić do powyższego punktu 3, w którym przerwaliśmy tworzenie nowego projektu, zatem powtórzmy go raz jeszcze i powiedzmy co będzie dalej:

3. Pojawia się okno dialogowe zawierające kolejne kroki pozwalające utworzyć nowy projekt. Podobnie jak w module 3 tworzymy projekt nadając mu nazwę (u nas Kalk1), jednak teraz z listy dostępnych szablonów wybieramy ten świeżo wgrany (Microsoft Visual Studio Scrum 1.0), dalej postępujemy analogicznie jak w poprzednim module.
4. Projekt zostaje utworzony i jeśli wszystko zakończyło się sukcesem to nowy projekt pojawi się w oknie Team Explorer i będzie wyglądał jak na Rysunku 6.



Rysunek 6. Team Explorer zawierający projekt Kalk1.

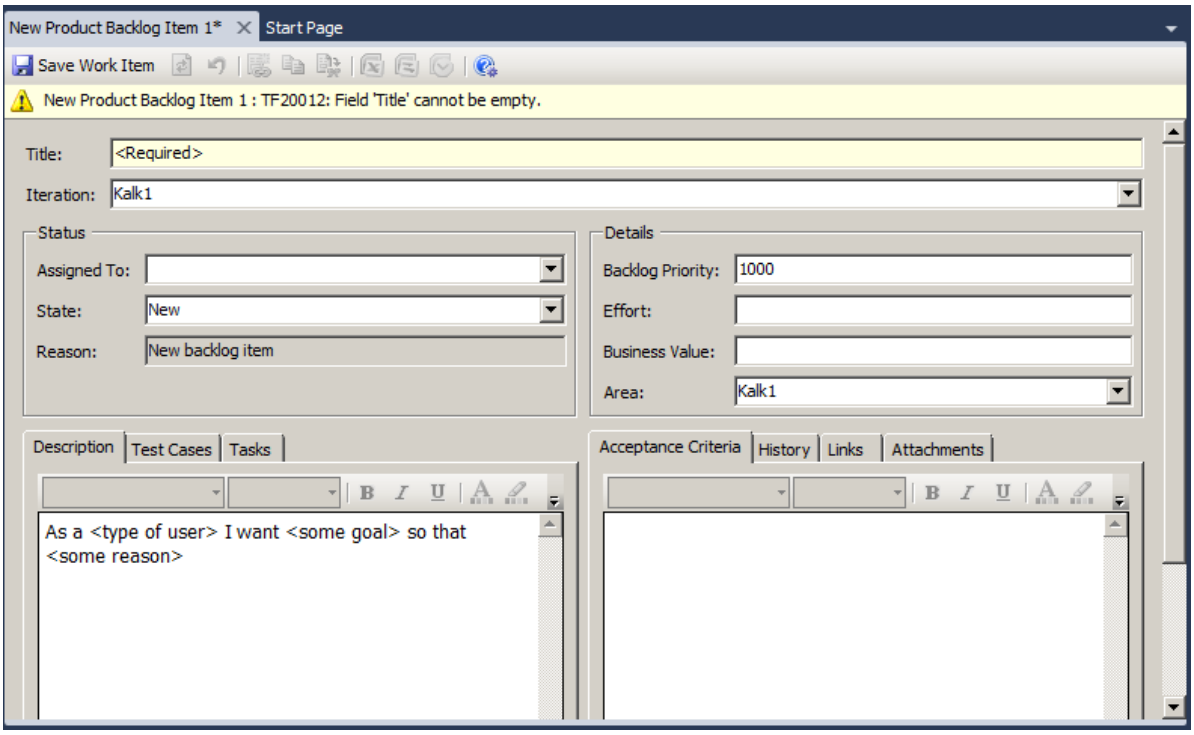
Co więcej, jeśli najedziemy kursorem na Work Item a następnie naciśniemy prawy klawisz myszy i dalej New Work Item, to otrzymamy listę zgodną z tym co podaliśmy wyżej, zatem widać, że założyliśmy projekt w oparciu o nowy szablon (Rysunek 7.).



Rysunek 7. Menu dodawania nowego Work Item zgodne z metodyką Scrum.

Praca z wykazem prac produktu i zadaniami

Nadszedł czas kiedy musimy przejść do zapisu wymagań Klienta, z opisu metodyki Scrum wynika, że musimy stworzyć, a właściwie uzupełnić wykaz prac produktu (Backlog). Dodajmy zatem pierwszy element do tego wykazu. W tym celu w oknie Team Explorer na drzewie naszego projektu najeżdżamy kursorem myszy na pozycję Work Item i naciskamy prawy klawisz – z menu wybieramy New Work Item, a następnie Product Backlog Item (Rysunek 7.). Po naciśnięciu tej pozycji menu pojawi się w miejscu ekranu powitalnego duże okno dialogowe definiowania nowego elementu wykazu prac produktu (Rysunek 8.).



Rysunek 8. Okno dialogowe pozwalające określić atrybuty dla nowej pracy dodawanej do wykazu prac produktu.

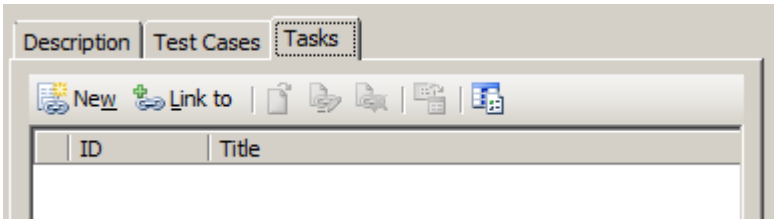
W oknie tym mamy do wypełnienia lub wyboru wiele pól, opiszemy je teraz:

- **Title** – jest to atrybut wymagany, który powinien być krótkim tekstem charakteryzującym tę pracę. W naszym przypadku „Szkielet kalkulatora”.
- **Iteration** – pozwala na określenie ścieżki iteracji. Na początku można przypisać element do korzenia aplikacji (Kalk1) a później, podczas planowania sprintu przenieść do innej iteracji.
- **Assigned To** – pole wyboru, w którym możemy dokonać przypisania, kto będzie realizował tę pracę. Przy czym w polu tym będą widoczni członkowie zespołu przypisani do grupy Contributors (Moduł 3).
- **State** – pozwala określić stan, na chwilę obecną pozostawmy New – w dalszej części omówimy je dokładniej
- **Reason** – pozwala identyfikować pochodzenie/przyczynę tego zdarzenia, na chwilę obecną pozostaje bez zmian.
- **Effort** – pozwala za pomocą wartości liczbowej wyrazić subiektywną ocenę pracochłonności danego zadania, większa wartość oznacza więcej pracy.
- **Business Value** – pozwala, za pomocą wartości liczbowej, wyrazić subiektywną ocenę wartości biznesowej danego zadania.
- **Backlog Priority** – pozwala, za pomocą wartości liczbowej, wyrazić subiektywną ocenę priorytetu danego zadania Większa wartość oznacza niższy priorytet. Co więcej dokumentacja Microsoft sugeruje, iż można ją wyliczyć jako oczekiwany zysk z inwestycji (ROI) poprzez podzielenie wartości biznesowej przez ocenę pracochłonności.
- **Area** – pozwala określić obszar.
- **Description** – pozwala na umieszczenie opisu tego elementu. Wpiszmy „Kalkulator ma posiadać graficzny interfejs użytkownika i klawiaturę przypominająca rzeczywistą, na początek ma on tylko pozwalać wprowadzać cyfry i wyświetlać je na wyświetlaczu”.
- **Test cases** – pozwala określać przypadki testowe dla tego elementu – więcej na ten temat w dalszych modułach.
- **Tasks** – pozwala określić zadania związane z tym elementem – więcej na ten temat dalej.

- **Acceptance Criteria** – miejsce, w którym możemy opisać kryteria akceptacji dla tego elementu.
- **History** – zakładka gromadząca historię aktywności na tym elemencie.
- **Links** – zakładka, w której można dokonywać połączeń z innymi elementami czy np. przeszkodami (Impediments).
- **Attachments** – w tej zakładce można dołączać różne dokumenty odnoszące się do tego elementu: maile, obrazki itp.

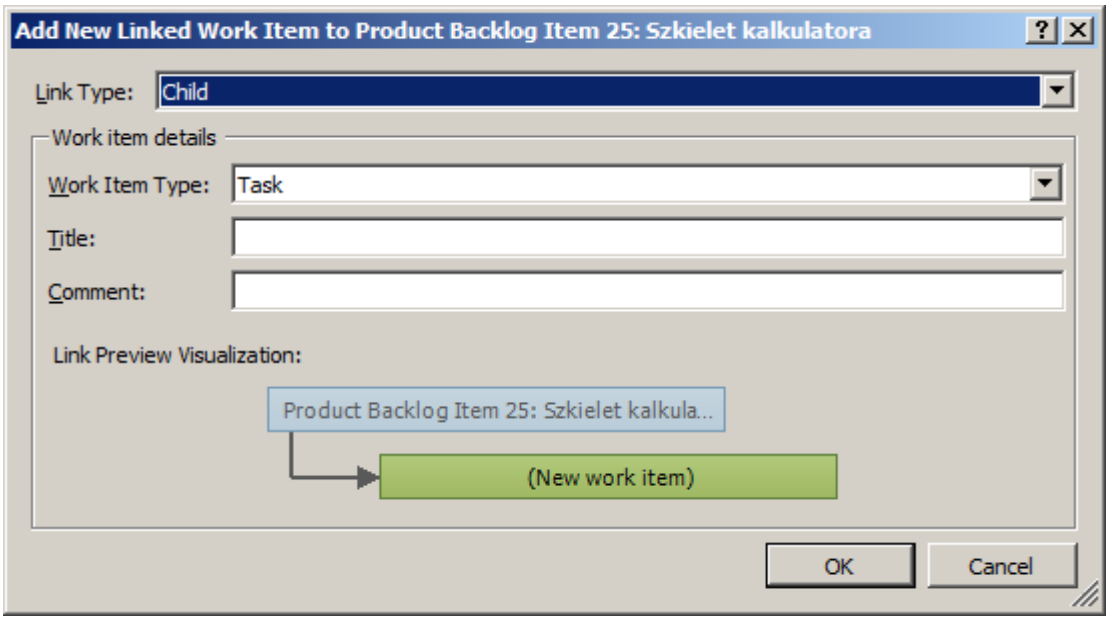
Po wypełnieniu pól sugerowanych w powyższym omówieniu naciskamy przycisk Save Work Item, w wyniku naszego działania dodaliśmy pierwszą pracę do Wykazu prac produktu (Product Backlog). Tu uwaga, by nie przejmować się numeracją prac w ramach wykazu, gdyż są to kolejne numery i ich wartość wynika z ogólnych działań i nie ma szczególnego wpływu na nasz projekt. Dla nas wygodniej będzie się posługiwać nazwami.

Zauważmy teraz, że wynikiem naszych działań jest dodanie nowego elementu w wykazie prac produktu, ale w Scrum dopiero zadania są elementami podlegającymi wykonaniu, przy czym na wykonanie pracy może przypadać wiele zadań. W tej chwili nie będziemy pokazywać całego przebiegu prac zgodnie z metodyką Scrum, gdyż wyłącznie chcemy zapoznać się z narzędziem. Zatem przejdziemy od razu do dodania zadania do wcześniej dodanej pracy. W tym celu klikamy na zakładkę Tasks, a następnie na przycisk New (Rysunek 9.)



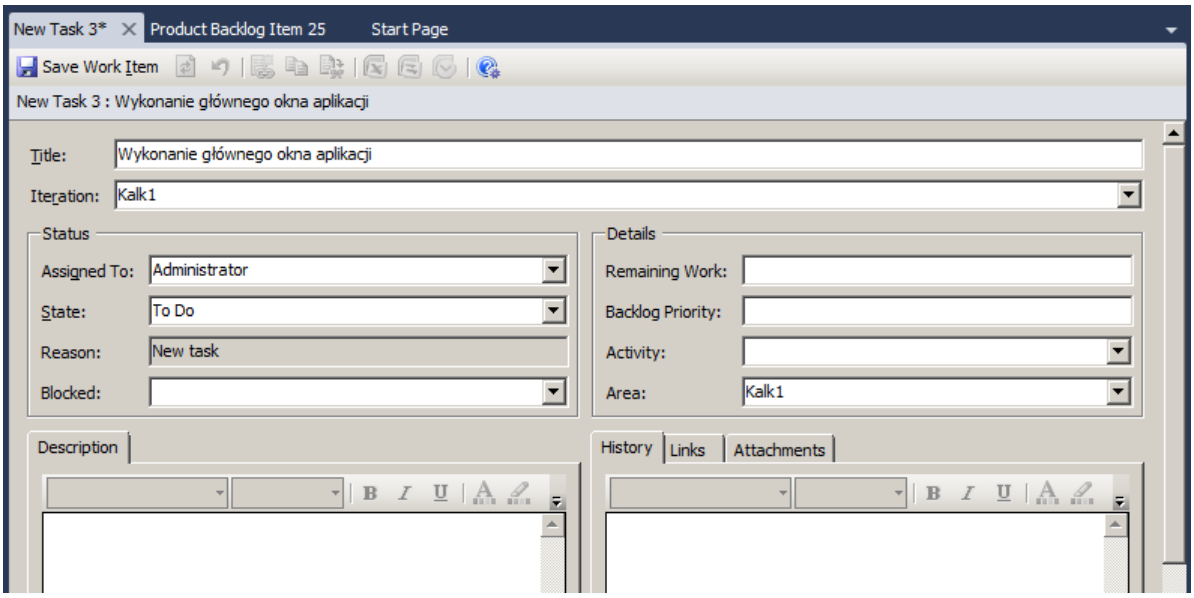
Rysunek 9. Widok panelu zadań.

Po naszej akcji pojawi się okno dialogowe (Rysunek 10.), Link Type pozostawiamy jako Child, wprowadzamy w nim w polu Title nazwę dla pierwszego zadania „Wykonanie głównego okna aplikacji” i klikamy OK.



Rysunek 10. Okno dialogowe pozwalające wprowadzić podstawowe informacje o zadaniu.

W efekcie naszego działania pojawi się duże okno pozwalające wprowadzić dalsze szczegóły zadania (Rysunek 11.).



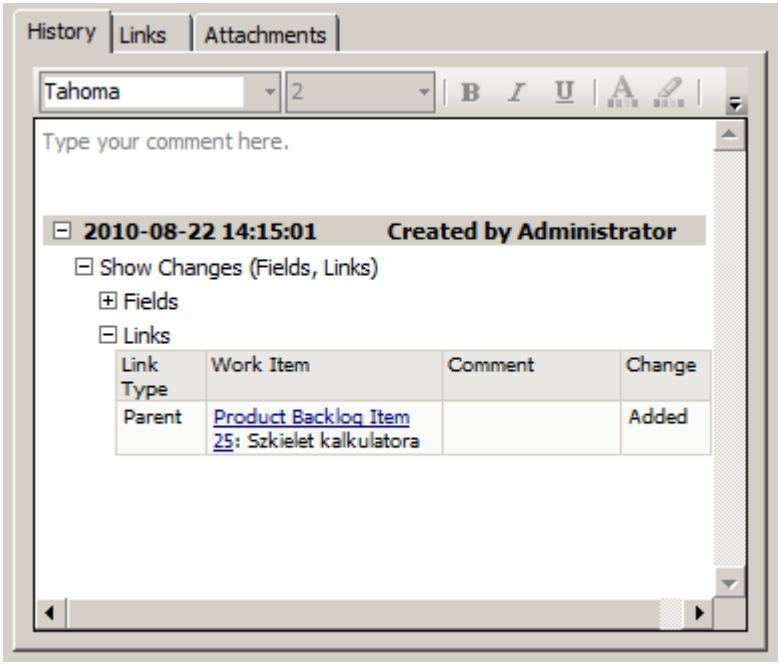
Rysunek 11. Okno dialogowe pozwalające określić szczegóły nowego zadania.

W oknie dialogowym dla zadania możemy określić następujące dodatkowe informacje/atrybuty:

- **Iteration** – pozwala przypisać zadanie do konkretnej iteracji.
- **Assigned To** – w tym polu wybieramy osobę odpowiedzialną za to zadanie, osoby które pojawiają się na tej liście muszą być, podobnie jak to było w przypadku pracy, członkami grupy Contributors. Zauważmy przy tym, że jedna osoba może być odpowiedzialna za całą pracę, podczas gdy inna za konkretne zadania składające się na tę pracę. Co więcej, pamiętajmy, że z dobrych praktyk zarządzania wynika, że odpowiedzialności nie wolno dzielić. Zatem, jeśli z jakichkolwiek przyczyn okaże się, że dwie osoby miałby odpowiadać za pewne zadanie, to należy je podzielić na dwa mniejsze podzadania z osobnymi odpowiedzialnymi.
- **State** – pozwala określić stan, w tym momencie pozostawiamy To Do – w dalszej części omówimy je dokładniej.
- **Reason** – pole analogiczne jak w przypadku pracy.
- **Blocked** – pole pozwalające zablokować zadanie – wybierzmy Yes.
- **Remaining Work** – pole w którym należy wpisać ilość godzin jaką powinno zająć wykonanie tego zadania. W przypadku zadań, które są podzielone na podzadania należy określać ilość godzin dla podzadań a nie zadania głównego.
- **Backlog priority** – pole analogiczne jak dla pracy.
- **Activity** – pozwala określić typ aktywności, który jest wymagany do wykonania tego zadania, są to: Deployment, Design, Development, Documentation, Requirements, Testing. W naszym przypadku będzie to Development.
- **Area** – pozwala na określenie w jakim obszarze będzie realizowane to zadanie. Można je zostawić puste, by określić jego wartość dopiero w trakcie planowania sprintu.
- **Description, History, Links, Attachments** – pola analogiczne jak w przypadku definiowania elementów wykazu prac produktu.

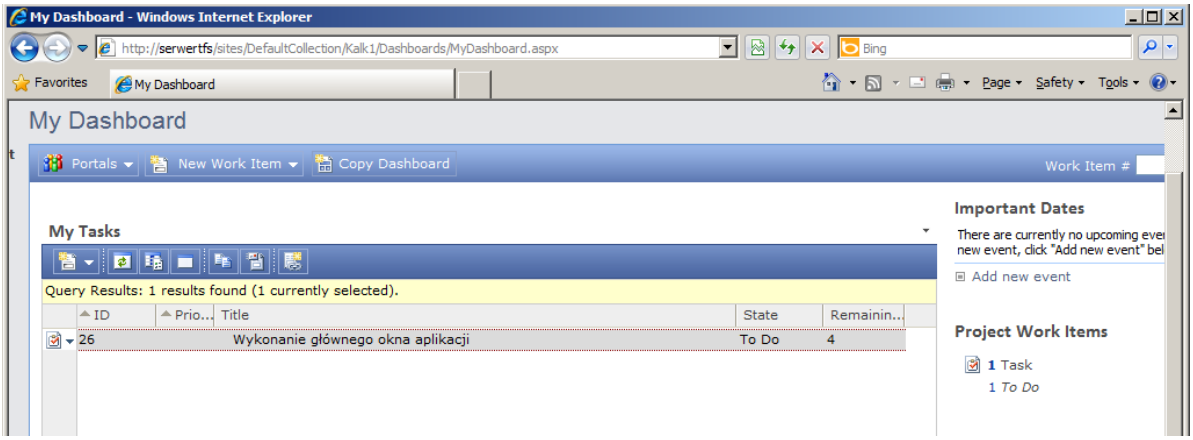
Wpiszmy lub wybierzmy pożądane wartości w omówionych polach i możemy nacisnąć przycisk Save Wotk Item. W wyniku naszych działań zostało utworzone nowe zadanie. Zauważmy, że również zostały utworzone odpowiednie powiązania i opisy. Dla przykładu w panelu Tasks i Links, w oknie

pracy, która zawiera to zadanie pojawi się ona jako Child, panel History w pracy będzie zawierał historie aktywności, zatem i fakt utworzenia, przed chwilą, zadania (wpisy w tym oknie wyświetlane są w kolejności od najnowszego do najstarszego). Z kolei, w panelu Links w zadaniu pojawi się powiązanie z rodzicem (pracą zawierającą to zadanie), a w panelu History znajdzie się zapis zmian, w tym informacja o zmienionych polach i utworzonych powiązaniach (Rysunek 12.), dodatkowo możemy umieścić opis tego zapisu historycznego.



Rysunek 12. Zawartość zakładki History w utworzonym zadaniu z rozwiniętą sekcją Links.

Dzięki tym wszystkim zautomatyzowanym czynnością zostały utworzone wszelkie powiązania oraz zapisy historyczne, dzięki czemu w trakcie realizacji projektu zawsze będziemy w stanie odtworzyć, przejrzeć wszelkie relację wraz z historią aktywności i wiedzą, kto dokonywał modyfikacji, to z kolei umożliwia lepsze zarządzanie projektem szczególnie w obszarze zarządzania jakością. Na sam koniec otwórzmy portal projektu, jeśli prawidłowo przypisaliśmy powyższe zadanie dla Administratora, to pojawi się ona w jego panelu My Tasks, a po prawej stronie w sekcji Project Work Items pojawi się jedno zadanie do realizacji (Rysunek 13.).



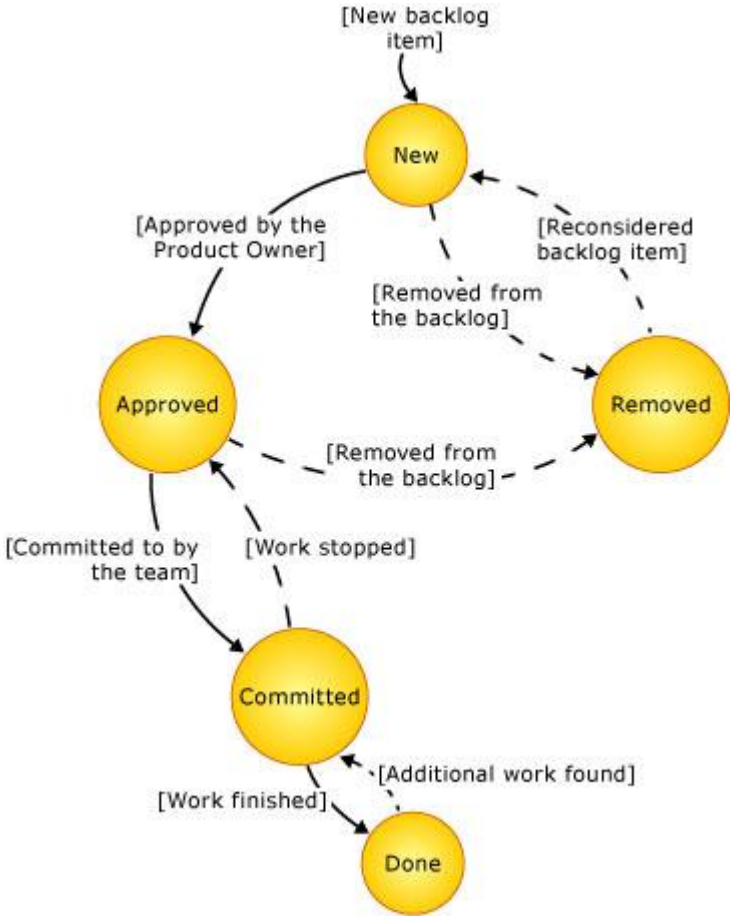
Rysunek 13. Widok panelu zadań dla użytkownika Administrator z przydzielonym dla niego zadaniem.

Cykl życia pracy oraz zadania

Prace znajdujące się w wykazie oraz zadania posiadają pole State, które określa ich stan, do tej pory pozostawialiśmy tam domyśle wartości. Jednak by efektywne używać szablonu Microsoft Visual Studio Scrum 1.0 musimy omówić jakie wartości moga te pola przyjmować, to z kolei najlepiej jest omówić na podstawie „cyklu życia" pracy oraz zadania. Dopiero zrozumienie tych rzeczy umożliwi efektywną i co ważniejsze poprawną pracę w środowisku VS2010 i TFS 2010 w oparciu o metodykę Scrum.

Cykl życia pracy

Zacznijmy od omówienia zmian stanów oraz ich przyczyn dla prac znajdujących się w wykazie prac produktu, spójrzmy na rysunek 13. Każdy element wykazu prac produktu może znaleźć się w jednym z pięciu stanów: New, Approved, Committed, Done i Removed. Jest oczywistym, że właściwe zrozumienie tych stanów jest podstawą do efektywnej pracy. A to najlepiej podać na przykładach przyczyn, dla których następuję zmiana stanu. Spójrzmy zatem na rysunek 14 – znajduje się na nim diagram przedstawiający stany i możliwe przejścia pomiędzy nimi, przy czym linia ciągła obrazuje typowe zmiany stanów, podczas gdy przerywana nietypowe. Opis szczegółowy znajduje się w poniższej tabeli (Tabela 1.).



Rysunek 14. Cykl życia pracy wyrażony za pomocą zmian jej stanu. Linia ciągła to zmiana typowa, przerywana to zmiany nietypowe.

Zmiana stanu pracy	Przyczyna
New >> Approved	Następuje, gdy Właściciel produktu akceptuje pracę w wykazu.
New >> Removed	Następuję, gdy Właściciel produktu decyduje, że dana praca nie będzie implementowana, pomimo iż znajduje się w Wykazie prac

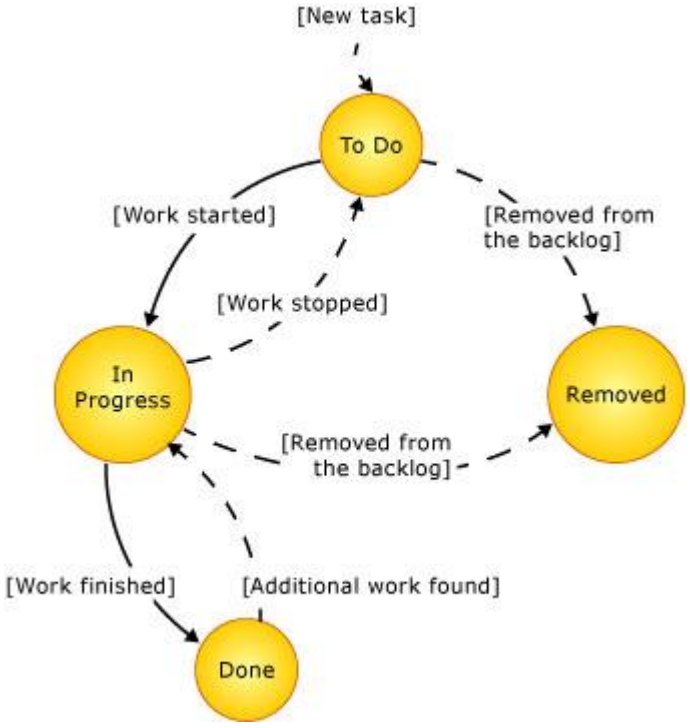
	produktu.
Approved >> Committed	Następuje, gdy zespół zaakceptował pracę do wykonania w bieżącym sprincie.
Approved >> Removed	Następuje, gdy zespół zadecydował, iż ta praca nie będzie wykonana, gdyż zmieniły się warunki np. wymagania.
Removed >> New	Następuje, gdy zespół zmienił zdanie i ponownie przyjął pracę do wykazu.
Committed >> Done	Następuje, gdy zespół wykonał pracę i zostały spełnione jej kryteria akceptacji.
Done >> Committed	Następuje, gdy zespół znalazł dodatkowe czynności, które muszą być wykonane, aby pracę uznać za zakończoną.
Committed >> Approved	Następuje, gdy zostały wstrzymane prace nad tym elementem z przyczyny zmiany osób w zespole lub zmiany priorytetu.

Tabela 1. Wykaz zmian statusu pracy i przyczyn tych zmian.

Jak widać, statusy te, oraz przepływy pomiędzy nimi wyczerpują to czego żąda metodyka Scrum. Zespół powinien dbać o aktualizację statusu pracy, tak by zawsze odpowiadał on rzeczywistości, skutkuje to między innymi w raportach o czym dalej.

Cykl życia zadania

Podobnie jak w przypadku prac, również dla zadania zaczniemy od omówienia zmian jego stanów oraz ich przyczyn, spójrzmy na rysunek 15. Każde zadanie może się znaleźć w jednym z czterech stanów: To Do, In Progress, Done, Removed. I tu również, jak poprzednio, podamy na przykładach przyczyny dla których następuję zmiana stanu zadania. Spójrzmy zatem na rysunek, gdzie znajduje się diagram przedstawiający stany i możliwe przejścia pomiędzy nimi, przy czym linia ciągła obrazuje typowe zmiany stanów, podczas gdy przerywana nietypowe.



Rysunek 14. Cykl życia zadania wyrażony za pomocą zmian jego stanu. Linia ciągła to zmiana typowa, przerywana to zmiany nietypowe.

Podobnie jak poprzednio poniższa tabela (Tabela 2.) opisuje szczegółowo przyczyny, dla których może nastąpić zmiana stanu zadania.

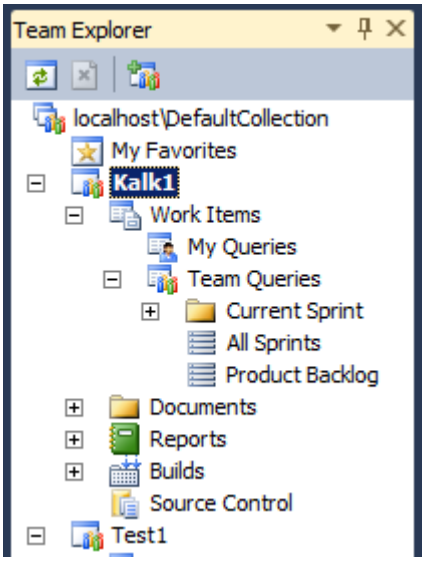
Zmiana stanu zadania	Przyczyna
To Do >> In Progress	Następuje, gdy członek zespołu rozpoczyna pracę nad danym zadaniem.
To Do >> Removed	Następuje w dwu przypadkach. Pierwszy, gdy praca, która zawiera to zadanie została uznana za zbędną. Drugi, gdy zespół zrezygnował z funkcjonalności opisanej tym zadaniem.
In Progress >> Done	Kiedy zespół zakończył to zadanie i zostały spełnione jego kryteria akceptacji.
In Progress >> Removed	Następuje w dwu przypadkach. Pierwszy, gdy praca, która zawiera to zadanie została uznana za zbędną. Drugi, gdy zespół zrezygnował z funkcjonalności opisanej tym zadaniem.
In Progress >> To Do	Następuje, gdy zostały wstrzymane prace nad tym zadaniem z przyczyny zmiany osób w zespole lub zmiany priorytetu.
Done >> In Progress	Następuje, gdy zespół znalazł dodatkowe czynności, które muszą być wykonane, aby zadanie uznać za zakończone.

Tabela 2. Wykaz zmian statusu zadania i przyczyn tych zmian.

Podobnie jak w przypadku prac, tak i tu, statusy te, oraz przepływy pomiędzy nimi wyczerpują to czego żąda metodyka Scrum. Zespół powinien dbać o aktualizację statusu zadania z tych samych przyczyn co wcześniej.

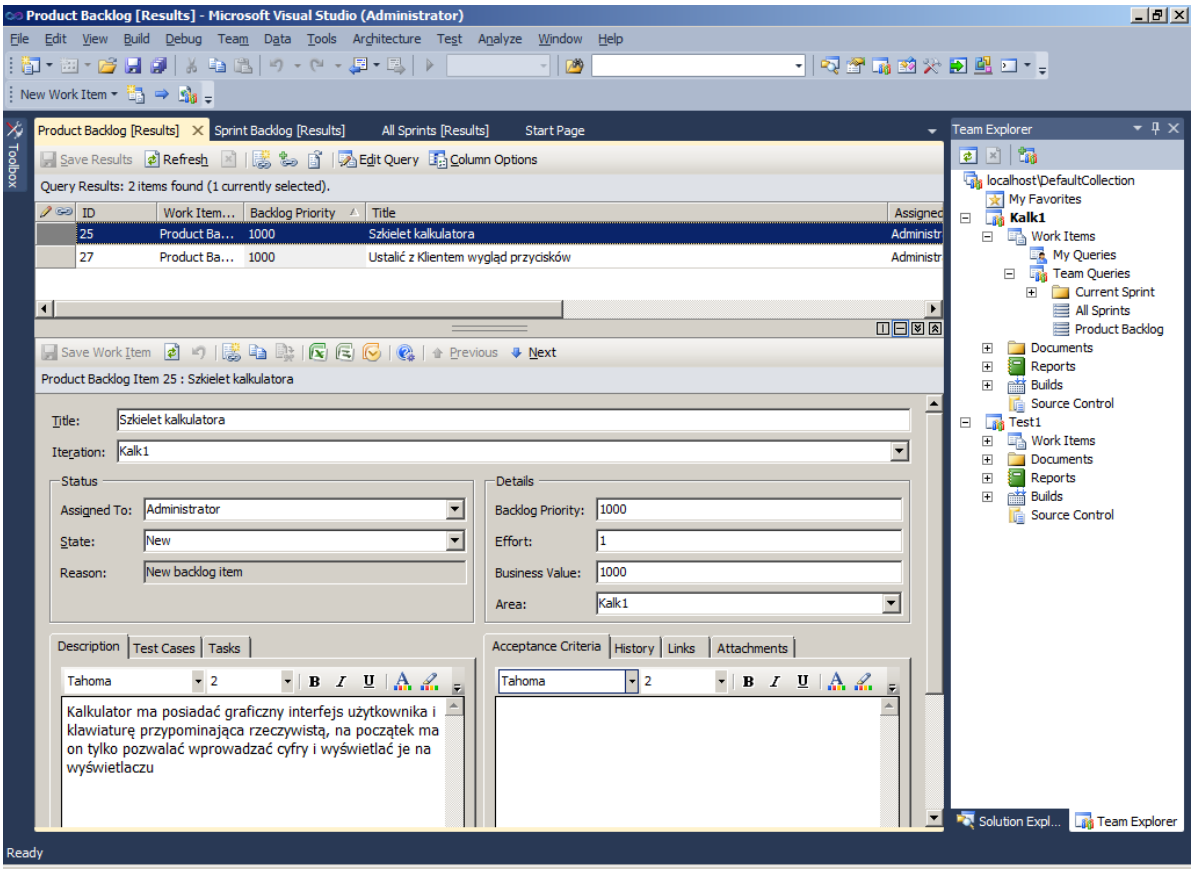
Wykaz prac produktu

Na sam koniec może warto wspomnieć w jaki sposób można przeglądać prace zawarte w wykazie. Najprostszym sposobem jest skorzystanie w tym celu z drzewa projektu w oknie Team Explorer w Visual Studio 2010. Należy je rozwinąć w węźle Team Queries, pojawią się tam kolejno: węzeł Curent Spint, All Sprints, Product Backlog (Rysunek 15.).



Rysunek 15. Drzewo projektu w oknie Team Explorer z rozwiniętym węzłem Team Queries.

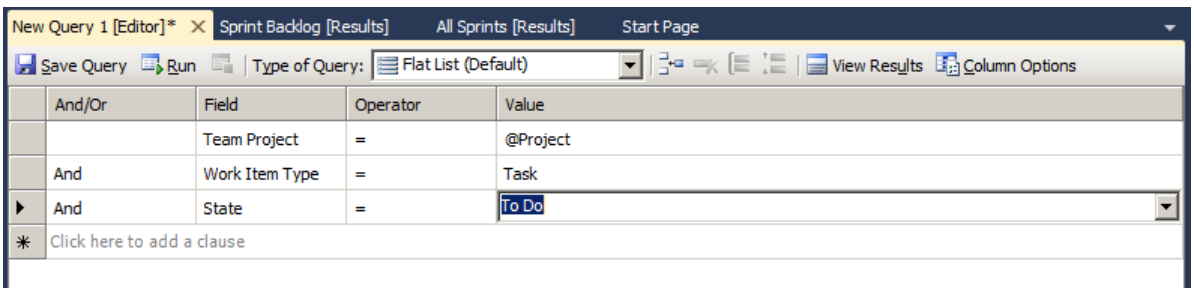
Wystarczy teraz najechać kursorem myszy na Product Backlog i dwukrotnie nacisnąć lewy klawisz myszy – spowoduje to pojawienia się okna zawierającego informacje o Wykazie prac produktu (Rysunek 16.).



Rysunek 16. Okno pokazujące wszystkie prace zawarte w Wykazie prac produktu.

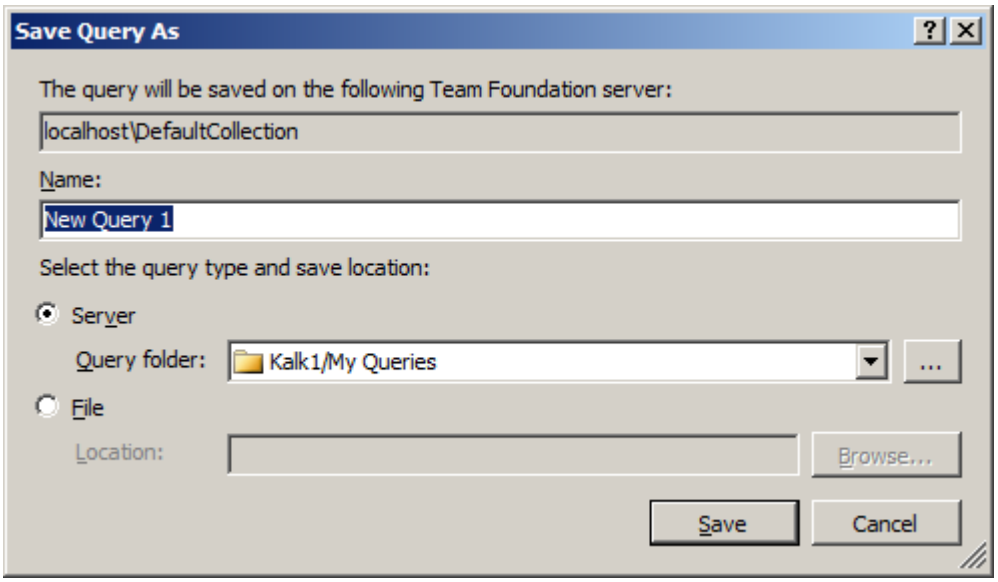
W oknie tym mamy możliwość przechodzenia pomiędzy kolejnymi pracami zdefiniowanymi w Wykazie prac oraz oglądanie ich szczegółów. Co więcej, jeśli chcemy to za pomocą tego okna możemy dokonywać modyfikacji w pracach, zatem jest to wygodne narzędzie do przeglądania wszystkich prac produktu.

W tym samym drzewie projektu (Rysunek 15.) mamy możliwość przeglądania prac bieżącego Sprintu oraz wszystkich sprintów, a powyżej Team Queries znajduje się My Queries, który domyślnie jest pusty, ale naciskając prawym klawiszem myszy, gdy kursor znajduje się nad nim, mamy możliwość definiowania nowych własnych zapytań. Naciśnijmy i kliknijmy New Query – pojawi się nowe okno jak na rysunku 17. W oknie tym możemy zdefiniować nowe nasze zapytanie jak w przykładzie (Work Item Type – tylko zadania, State – Tylko To Do).



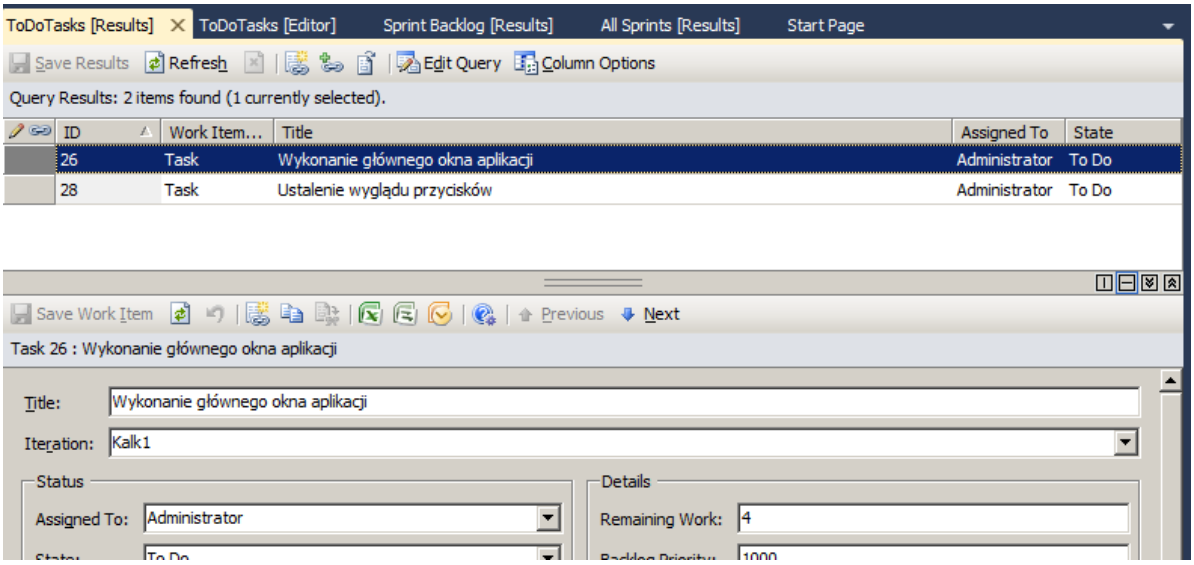
Rysunek 17. Okno dialogowe definicji nowego zapytania.

Po zdefiniowaniu nowego zapytania naciskamy Save Query i pojawia się okno dialogowe zapisu zdefiniowanego zapytania, gdzie mamy możliwość określenia m.in. jego nazwy (Rysunek 18.). Wprowadźmy nazwę ToDoTasks i naciśnijmy przycisk Save – jeśli wszystko przebiegło prawidłowo, to w węźle My Queries pojawi się nowa pozycja o nazwie, którą wpisaliśmy.



Rysunek 18. Okno dialogowe zapisu nowego własnego zapytania.

Teraz możemy wykonać nowe zapytanie poprzez dwukrotne kliknięcie lewym przyciskiem myszy na jego nazwie – wykonajmy to, w wyniku wykonania zapytania powinno pojawić się okno podobne do tego, które prezentuje rysunek 19. Zawartość tego okna może być inna, w zależności ile zadań zdefiniowano.



Rysunek 19. Przykładowy wynik wykonania zapytania zdefiniowanego przez użytkownika.

W tym momencie zakończymy wprowadzenie do metodyki Scrum i jej realizacji w ramach środowiska Visual Studio 2010 oraz Team Foundation Studio 2010 w oparciu o szablon Microsoft Visual Studio Scrum 1.0. Na koniec warto tylko przypomnieć, że gdybyśmy użyli szablony dostarczanego z VS2010, to przebieg omówionych procesów byłby inny.

Porady praktyczne

Metodyka Scrum, podobnie jak dowolna inna metodyka, posiada swoje nazewnictwo, techniki, narzędzia i z tym wszystkim przyjdzie się zmierzyć w trakcie realizacji pierwszych projektów. Ciężko jest oczekiwać, że pierwszy czy drugi projekt zostanie zrealizowany bezbłędnie i w zgodzie ze wszystkimi regułami tej metodyki. Stąd też, warto spróbować zastosować reguły Scrum nawet na bardzo prostych projektach, nawet troszkę sztucznie w stosunku do wielkości projektu, ale celem tego będzie dobre poznanie wszystkich reguł i narzędzi nawet na tym początkowym etapie. Zalecamy skupienie się – jak w tym module – jedynie na definicji prac w ramach Wykazu prac produktu, jeszcze bez wchodzenia w implementację i definiowanie sprintów.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium jeśli:

- rozumiesz idee metodyki Scrum oraz jej elementów,
- umiesz wykorzystać wiedzę o Scrum w trakcie tworzenia projektu w Visual Studio 2010,
- potrafisz zainicjować projekt przy użyciu metodyki Scrum oraz określić role Scrum i potrafisz przeprowadzić początkowe prace,
- potrafisz w środowisku Visual Studio 2010 dodawać nowa pracę, zadanie oraz wykonywać podstawowe operacje dla nich,
- rozumiesz znaczenie ról, iteracji, Sprintu i pozostałych pojęć metodyki Scrum,
- rozumiesz znaczenie statusu prac oraz zadań oraz znasz cykl ich życia.

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. Ken Schwaber, *Agile Project Management with Scrum*, Microsoft Press 2004

   W książce autor omawia aspekty zarządzania projektami przy zastosowaniu metody Scrum.

2. Ken Schwaber, *The Enterprise and Scrum*, Microsoft Press 2007

   W książce autor omawia aspekty wykorzystania metodyki Scrum przy zarządzania projektami o dużej skali.

Laboratorium podstawowe

Twoim zadaniem będzie dalsze samodzielne zapoznanie się z szablonem Microsoft Visual Studio Scrum 1.0. W szczególności dobre zrozumienie zmian stanów dla prac, zadań oraz powiązanie ich ze zdarzeniami je wywołującymi. W tym celu posłużymy się przykładem z programem kalkulator, który był rozpoczęty powyżej.

Zadanie	**Tok postępowania**
1. Dodanie nowej pozycji do wykazu prac produktu	• Przy rozmowach z Właścicielem produktu okazało się, iż nie jasne jest do końca jaki ma być wygląd kalkulatora, zatem należy sprecyzować np. wygląd przycisków. • Najeżdżamy kursorem myszy na węzeł Work Item w drzewie projektu w oknie Team Explorer w VS2010. • Naciskamy prawy klawisz myszy i w menu, które się ukazało najeżdżamy kursorem na pozycję New Work Item i klikamy w pozycję menu Product Backlog Item • W oknie dialogowym wpisujemy lub wybieramy w polach następujące wartości: – **Title**: „Ustalić z Klientem wygląd przycisków”, – **Assigned To**: Administrator, – **Backlog Priority**: 1000, – **Effort**: 1, – **Bussiness value**: 1000, – **Description**: „Należy z Klientem ustalić, czy przyciski w kalkulatorze mają być zwykłymi przyciskami "button", czy też mają mieć podłączoną grafikę.” • Zatwierdzamy zmiany poprzez naciśnięcie przycisku Save Work Item.
2. Dodanie zadania do pracy	• W oknie dialogowym pracy (Product Backlog Item) klikamy na panel Tasks. • Następnie naciskamy na przycisk New, by stworzyć nowe zadanie dla wcześniej dodanej pracy. • W oknie, które się pojawiło wpisujemy w polu **Title**: „Ustalenie wyglądu przycisków”, pozostałe pola pozostawiamy bez zmian i naciskamy OK.
3. Określanie szczegółów zadania	• W wyniku powyższego kroku pojawi się okno dialogowe jak na rysunku 11. • W oknie dialogowym Task wpisujemy lub wybieramy w polach następujące wartości: – **Assigned To**: Administrator, – **State**: To Do, – **Remaining Work**: 1, – **Backlog Priority**: 1000, – **Activity**: Requirements. • Zatwierdzamy zmiany poprzez naciśnięcie przycisku Save Work Item.
4. Sprawdzenie połączeń zadania i pracy oraz historii	• W oknie Task naciskamy na zakładkę Links i sprawdzamy jej zawartość. • Jeśli poprzednie kroki zostały wykonane prawidłowo, to okaże się, że znajduje się tu węzeł Parents zawierający jeden element, który w kolumnie Work Item Type będzie miał wartość Product Backlog Item, a

	w kolumnie Title wartość zgodną z tytułem, który wpisaliśmy przy definiowaniu pracy. • Kliknijmy dwa razy na tej pozycji listy w zakładce Links, w wyniku naszego działania powinno otworzyć się okno dialogowe rodzica (Product Backlog Item). • Kliknijmy teraz na zakładkę Links i w niej znajduje się węzeł Child, który zawiera jeden element – zadanie przez nas zdefiniowane. Oczywiście w kolumnie Work Item Type będzie wartośćTask, a w tytule znajdzie się to co wpisaliśmy przy definiowaniu zadania. • Nadal w oknie pracy kliknijmy na zakładkę History. Zawiera ona informacje o dokonywanych zmianach. Znajdują się tam dwie pozycje (pamiętajmy, że kolejność pozycji od najnowszego wpisu do najstarszego): – Data i godzina Edited by Administrator – rozwińmy węzeł Show Changes (Links). Znajduje się w nim wpis świadczący o tym, że zostało utworzone połączenie do dziecka (nasze dodane zadanie). – Data i godzina Created by Administrator – znów rozwińmy węzeł Show Changes (Fields). Pojawi się cała lista pól i nadanych im wartość obrazująca fakt utworzenia pracy. • Przejdźmy teraz do okna zadania. W tym celu w oknie Product Backlog Item kliknijmy na zakładkę Links, a następnie dwa razy na pozycji zadania w węźle Child. • W oknie zadania kliknijmy teraz na zakładkę History. Widzimy w niej pozycje: – Data i godzina Created by Administrator – rozwińmy węzeł Show Changes (Fields, Links). Widać w nim dwie kolejne sekcje: Fields i Links. W nich są uwidocznione zapisy w polach, oraz utworzenie linku do rodzica.
5. Zmiana statusu zadania	• Jesteśmy w oknie naszego zadania. Obecny status to To Do, rozwińmy okno statusu i zobaczmy jakie tam znajdują się dopuszczalne wartości: To Do, In Progress, Removed. Jak widać brakuje statusu Done, spójrzmy jednak na diagram z rysunku 14. Widać wyraźnie, iż nie istnieje droga przejścia bezpośrednio ze stanu To Do do Done. • Zmieńmy stan z To Do na In Progress, a następnie z In Progress na Done (za każdym razem w liście statusów będziemy mieli tylko te stany, które w danym momencie są dopuszczalne). • Po tych czynnościach spójrzmy do zakładki History, wszystkie nasze akcje znajdują tam swoje odzwierciedlenie.
6. Definiowanie własnego zapytania	• Chcemy stworzyć zapytanie, które będzie pozwalało wybierać te zadania, które mają status In Progress i dotyczą tylko użytkownika „bieżącego” – czyli „mnie”. • W oknie Team Explorer na drzewie projektu naciskamy prawy klawisz myszy, gdy kursor znajduje się nad węzłem My Queries, następnie z menu wybieramy New Query. • Pierwszy wiersz zapytania pozostawiamy bez zmian. • W drugim wierszu, w kolumnie Field ustawiamy wartość Work Item Type, a w kolumnie Value wybieramy wartość Task. • W trzecim wierszu w kolumnie Field ustawiamy wartość State, a w kolumnie Value wybieramy wartość In Progress.

	• W kolejnym wierszu w kolumnie And/Or wybieramy And, w kolumnie Field wybieramy Assigned To, a w kolumnie Value wartość @Me. • Naciskamy przycisk Save Query, w oknie dialogowym, które się pojawi wpisujemy nazwę InProgressMyTasks i naciskamy OK.
7. Wykonanie własnego zapytania	• Rozwijamy węzeł My Query i dwukrotnie klikamy na zapytanie InProgressMyTasks. • Jeśli powyższe czynności wykonaliśmy poprawnie, to powinnyśmy otrzymać okno zawierające wynik zapytanie, gdzie nie będzie żadnego wiersza – oznacza to, że żadne z zadań nie jest aktualnie w trakcie realizacji. • Otwórzmy zatem ponownie okno wcześniej utworzonego zadania i zmieńmy status na In Progress, zapiszmy zmiany. • Wracamy do okna naszego zapytania (jeśli go nie zamknęliśmy to wystarczy kliknąć na jego zakładkę na górnej belce) i naciskamy przycisk Refresh. Zdanie, w którym zmieniliśmy status powinno się pojawić na liście.

Laboratorium rozszerzone

Twoja znajoma, chciałby byś stworzył dla niej prostą stronę WWW zawierającą informację o niej i formularz do kontaktu. Zależy jej na tym by strona wyglądała bardzo estetycznie i w związku z tym przyniosła gotową propozycję graficzną opracowaną przez grafika jako plik JPEG. Twoim zadaniem jest przeprowadzenie wywiadu z nią mającego na celu określenie wymagań co do strony oraz wpisanie tych prac do Wykazu prac produktu. Aby ułatwić ćwiczenie, w grupie podzielcie się na dwie podgrupy i jedna podgrupa symuluje naszą koleżankę i jej wymagania, a druga prowadzi rozmowę i wypisuje potrzeby produktu.

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 5

Wersja 1

Cykl życia oprogramowania

Spis treści

Informacje o module

Opis modułu

W tym module znajdziesz opis cyklu życia oprogramowania, który stanowi główną oś procesów wytwórczych w oparciu o który to cały zespół wytwarza dane oprogramowanie. Cykl życia oprogramowania i procesy z nim związane pomagają uporządkować prace wytwórcze, dzięki czemu każdy członek zespołu powinien rozumieć swoją rolę oraz zawsze jest wiadomo w którym momencie wytwarzania znajduje się aplikacja. Co więcej, dzięki wprowadzaniu standaryzacji procesów wytwórczych łatwiej jest zachować spójność i jakość projektu. Omówiony w tym rozdziale cykl jest jednym z wielu, jakimi można się posłużyć w trakcie wytwarzania aplikacji. Został ona wybrany jako proponowany przez Microsoft w środowisku Visual Studio 2010 i Team Foundation Studio 2010.

Cel modułu

Celem modułu jest zapoznanie z proponowanym cyklem życia oprogramowania.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- wiedział z jakich etapów składa się cykl życia oprogramowania,
- wiedział jaki cykl życia aplikacji proponuje przyjąć firma Microsoft,
- wiedział jak cykl życia aplikacji odnosi się do metodyki Scrum,
- potrafił zidentyfikować poszczególne etapy cyklu życia oprogramowania,
- potrafił stworzyć pierwszy cel Sprintu oraz wykaz jego prac i zadań,
- rozumiał rolę i wagę wykresów malejących oraz wiedział jak je uzyskać oraz jaką niosą ze sobą informację,
- rozumiał zakres prac i rolę procesów w ramach cyklu życia oprogramowania.

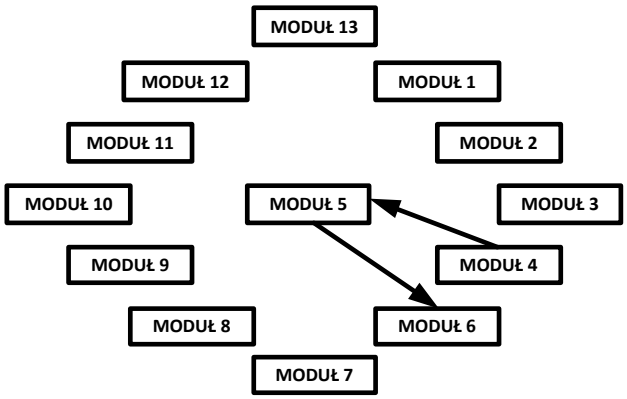
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawową obsługę programu Visual Studio 2010,
- znać sposób utworzenia projektu w oparciu o Team Foundation Studio 2010,
- rozumieć pojęcia ról w zespole programistycznym,
- rozumieć podstawowe pojęcia inżynierii oprogramowania np. cykl życia oprogramowania.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w module



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Wraz z resztą zespołu mieliście dobry pomysł, by wypróbować podstawowe zasady Scrum oraz nauczyć się go stosować na prostym przykładzie. Dzięki projektowi kalkulatora, będącego jednocześnie pomocą dla kolegi z młodszych lat studiów, ugruntowaliście podstawową wiedzę o Wykazie prac produktu (Backlog). Zauważyliście, że lepiej rozumiecie jak powstaje ten wykaz prac, jak można go umieścić w spisie w środowisku VS 2010, co więcej nauczyliście się podstawowych operacji na tym wykazie. Jednak, gdy spojrzeliście do raportów związanych z projektem kalkulatora, to okazało się, że nie ma tam żadnych wykresów, co więcej pokazuje się komunikat o braku danych do raportów.

Powoli dociera do Was, iż co prawda zdobyliście wstępną wiedzę o metodyce Scrum, to jednak nie umiecie pójść krok naprzód. Należy zaplanować pierwszy Sprint, ale co dalej? Jak w ogóle przebiega proces wytwarzania oprogramowania? Z jakimi dalszymi zasadniczymi czynnościami się spotkacie?

Z wiedzy wyniesionej z przedmiotów takich jak np. Inżynieria Oprogramowania pamiętacie, iż występowało pojęcie cykl życia oprogramowania. Pamiętacie również, w mniejszym lub większym stopniu, o metodykach wytwarzania oprogramowania, takich jak np. model kaskadowy czy przyrostowy. Jednak jak odnieść je do metodyki zarządzania jaką jest Scrum?

Zauważacie, że brak całościowego spojrzenia na cykl wytwarzania oprogramowania zaczyna Was nie tylko irytować, ale również obawiacie się, że nie jesteście w stanie porządnie zaplanować dalszych działań, gdyż po prostu nie wiedząc co następuję po czym, nie umiecie powiedzieć Klientowi kiedy może spodziewać się pierwszej wersji finalnej. Ostatecznie podjęliście decyzję, że wpierw uporządkujecie wiedzę, wypracujecie podstawowe wewnętrzne procedury wytwarzania oprogramowania w Waszej firmie i dopiero wtedy podejmiecie wyzwanie w postaci zlecenia od prawdziwego Klienta.

Podstawy teoretyczne

Cykl życia oprogramowania – przypomnienie

Jak pamiętamy z zajęć z Inżynierii oprogramowania, jest to dziedzina naukowa, która skupia się wokół zagadnień dotyczących tworzenie produktów jakimi są programy czy systemy informatyczne wraz z ich dokumentacją i pozostałymi elementami, które stanowią części pełnowartościowego produktu. Z naszego punktu widzenia Inżynieria oprogramowania zajmuje się analizą, projektowaniem, tworzeniem (kodowaniem), weryfikacją końcowego produktu i zarządzaniem projektem informatycznym. Do realizacji tych zadań używa się określonych narzędzi (np. te omówione w module 3), zaś sam proces tworzenia opisany jest przez jeden lub więcej modeli lub metodyk tworzenia oprogramowania. Na metodyki te możemy spojrzeć jak na poradniki, czy przewodniki, jak postępować w określonych sytuacjach, jak analizować problem, jak dzielić go na mniejsze zadania, jak testować czy rozwijać oprogramowanie.

Metodyki cyklu życia oprogramowania służą zespołom do opracowywania produktu, są tak samo niezbędne jak narzędzia, gdyż wyznaczają sposób postępowania w ramach projektu. Metodyki wprowadzają porządek i standaryzację w cykl wytwarzania oprogramowania. Zauważmy, że zwykle podkreśla się, iż produkcja oprogramowania ma wiele przeważającą ilość cech twórczości nad procesami. Jednak im więcej elementów produkcji uda się ująć za pomocą procesów i reguł, tym mniej pozostawiamy w gestii przypadku. Na świecie istniej wiele metodyk zrządzana projektami (patrz moduł 1) jak i wiele metodyk wytwarzanie oprogramowania. W idealnym przypadku można założyć, że metodykę zarządzania jak i wytwarzania wybiera się taką, która najlepiej do danego rozwiązania. Należy jednak zauważyć, iż w zdecydowanej ilości przypadków, jest to z jednej strony jest to nierealne, a z drugiej zupełnie nie opłacalne ekonomicznie. Stąd też zespół projektowy zna

jedną lub dwie metodyki i to raczej on próbuje je dostosować do danego projektu, niż uczyć się danego procesu zarządczego czy wytwórczego pod potrzeby danego projektu.

Obecnie, po wielu latach badań nad procesami wytwarzania oprogramowania, można spotkać w literaturze podział na następujące cztery grupy modeli cyklu życia oprogramowania:

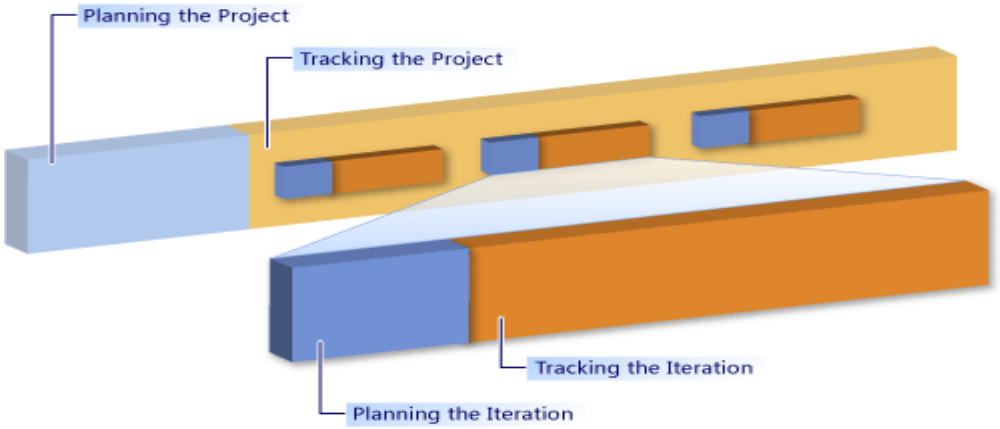
- **Sekwencyjne** – grupa ta reprezentuje podejście, w którym kolejne etapy produkcji następują po sobie bezpośrednio np.: specyfikacja wymagań, projektowanie, kodowanie i wreszcie testowanie. Każdy z tych etapów zakończony jest zatwierdzonym raportem po którym można przejść do następnego. Przykładem takiego podejścia będzie model kaskadowy.
- **Ewolucyjne** – grupa ta reprezentuje podejście, w którym aktywności się przeplatają. Zwykle pierwsza wersja systemu jest tworzona bardzo szybko na podstawie uproszczonej specyfikacji wymagań. Dalsze działania sprowadzają się do rozbudowy powstałego szkieletu poprzez stopniowe doprecyzowanie wymagań Klienta. Dobrym przykładem reprezentanta tej grupy jest model przyrostowy.
- **Formalna transformacja** – grupa ta reprezentuje bardzo specyficzne podejście do wytwarzania, w którym oparta o formalizację matematyczną specyfikacja systemu jest poddawana, za pomocą matematycznych metod, transformacji, by ostatecznie otrzymać program. W teorii takie podejście zapewnia, że transformacja jest niepodatna na błędy, a zatem finalny program od razu spełnia stawiane przed nim wymagania – czyli nie wymaga żmudnego procesu testowania i weryfikacji. Co więcej, przy takim podejściu można uniknąć problemu złego zrozumienia wymagań. Jednak ze względu na ogromny koszt stosowania tych metod, używa się ich wyłącznie do tworzenia systemów o wysokim ryzyku, poziomie bezpieczeństwa itp.
- **Składanie systemu z komponentów** – w podejściu tym zakłada się, iż wszystkie niezbędne bloczki przyszłego systemu już istnieją i jedynie należy je poskładać w całość.

Zauważmy, że już doświadczenia ze studiów pokazują, iż wiele metod programowania zawiera kilka podejść z powyżej zaprezentowanych. Dotyczy to szczególnie składania systemu z komponentów, obecnie dość często wprost mówi się o programowaniu komponentowym, wcale nie wyszczególniając go jako osobnej metodyki, ale jako czegoś, co bardzo usprawnia pracę w ramach dowolnej innej metodyki.

Proponowany cykl życia oprogramowania

W środowisku Visual Studio 2010 wraz z Team Foundation Studio 2010 został zaproponowany cykl życia oprogramowania zwany tam dokładnie Cykl Życia Aplikacji (ang. Application Lifecycle Management – ALM). Nie należy traktować go jako czegoś co wymyśliła firma Microsoft, a jest to odrębne od innych podejść. Tak naprawdę ALM jest jednym z możliwych podziałów, które znane są w inżynierii oprogramowania.

Doświadczenie pokazuje, iż w oparciu o ALM można wytwarzać oprogramowanie efektywne i wygodnie. Niewątpliwie podejście jakie proponuję Microsoft jest poparte praktyką wyniesioną z lat doświadczeń. Opiera się ono na podejściu iteracyjnym wytwarzania oprogramowania, które obecnie występuję najczęściej i jest dość charakterystyczne szczególnie dla metodyk zwinnych (w nomenklaturze z poprzedniego punktu będzie to grupa ewolucyjna) – Rysunek 1.



Rysunek 1. Plan nadrzędny i jedna z iteracji. (źródło: strona Microsoft)

W ALM zakłada się, iż praca zaczyna się od planu na ogólnym poziomie szczegółowości i powoli, stopniowo schodzimy coraz niżej, poznając wszelkie detale wymagań Klienta. Szczegółowy plan wykonania pojawia się na początku każdej z iteracji, dzięki temu można mieć większą pewność, zarówno co do tego co jest do wykonania, jak i znać precyzyjnie kryteria akceptacji. Zauważmy, że np. w modelu kaskadowym byłoby to trudne do osiągnięcia. Idąc dalej, po zakończeniu każdej iteracji, zespół ma okazję do przejrzenia planu głównego i wprowadzenia niezbędnych korekt. Co więcej, ALM pozwala stosować różne, nawet własne, szablony procesów wytwarzania oprogramowania, co czynie środowiska VS 2010 i TFS 2010 bardzo elastycznym i funkcjonalnym (mieliśmy tego przykłady już w modułach 3 i 4). Przypomnijmy, że wraz ze środowiskiem VS2010 i TFS2010 dostarczane są dwa szablony: MSF for Agile Software Development v5.0, MSF for CMMI Process Improvement v5.0, oraz jeden który sami zainstalowaliśmy: Visual Studio Scrum 1.0.

Czytając powyższe można popaść w konsternację i zapytać, jak to jest, że z jednej strony Microsoft proponuje ALM, a z drugiej dowiadujemy się, że możemy tworzyć własne szablony procesów wytwarzania oprogramowania. Otóż, ALM należy traktować, jako pewną ogólną koncepcję, która pozawala spojrzeć niejako z góry na cały cykl produkcyjny, podczas gdy konkretny szablon, wyznacza dokładne reguły postępowania, jak na przykład nazewnictwo poszczególnych elementów projektu (przypomnijmy, że z tego powodu zastosowaliśmy inny szablon dla Scrum – moduł 4). Zatem patrząc na dowolny projekt z perspektywy ALM wyróżnimy następujące grupy czynności:

- **Planowanie i śledzenie** – grupa procesów mających na celu wspomaganie zarządzaniem projektem jako takim. Zatem będzie to planowanie czynności, pozyskiwanie wymagań, śledzenie przebiegu prac i wszelkie pozostałe procesy mające na celu pomoc w zamianie wymagań klienta w rzeczywisty program. To właśnie ta grupa stanowi zbiór procesów niejako nadrzędnych nad pozostałymi grupami (wymienionymi poniżej) i to ona podlega zmianom w przypadku stosowaniu konkretnego szablonu.
- **Projektowanie** – grupa procesów, które zapewniają wsparcie dla projektowania całej aplikacji czy też jej poszczególnych składowych.
- **Programowanie** – grupa procesów, takich kodowanie, debugowanie, analizowanie, testowanie czy optymalizacja. W tej grupie znajdą się również takie elementy jak dobre praktyki programistyczne czy wersjonowanie kodu.
- **Budowanie** – grupa procesów, która wspiera budowanie aplikacji oraz umożliwia sprawdzanie, czy każdy build (kolejna zbudowana wersja systemu) spełnia wymagania klienta.
- **Testowanie** - grupa procesów, które umożliwiają budowanie i wykonywanie ręcznych i automatycznych testów, zarówno jednostkowych, systemowych, wydajnościowych czy obciążeniowych.

- **Wypuszczanie** – umożliwia wypuszczanie danego buildu aplikacji w środowisko przypominające produkcyjne (virtualne środowiska), dzięki czemu w lepszy sposób można testować poprawności działania systemu.

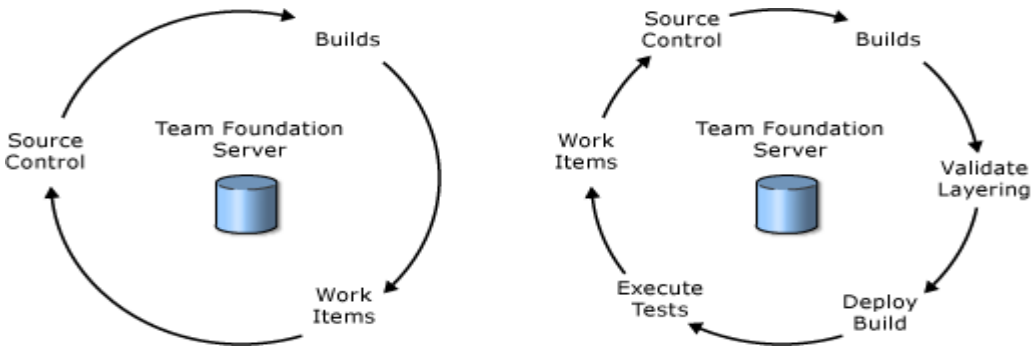
Ważne jest by zapamiętać, że pełne wsparcie dla ALM otrzymujemy jeśli zastosujemy Team Foundation Server 2010 oraz Visual Studio 2010 Ultimate. Inne elementy tej rodziny wspierają tylko część tej funkcjonalności. Dla dalszej lektury, by uniknąć nieporozumień, niezbędne jest teraz uściślenie pojęć, które do tej pory występowały, jednak nie definiowaliśmy je dość precyzyjnie. W ramach środowiska TFS 2010 zintegrowano trzy bardzo ważne rozwiązania:

1. **System kontroli wersji** (source control) – który posiada cechy dojrzałego takiego systemu (podstawy obsługi tej części były opisane w module 3).
2. **Śledzenie błędów** (bug tracking) – podsystem umożliwiające zaawansowane wsparcie dla obsługi zgłoszeń o błędach oraz śledzenie cyklu życia takiego błędu (więcej w module ???).
3. **Budowanie aplikacji** (Builds) – podsystem umożliwiający zautomatyzowanie budowanie aplikacji w oparciu o kod umieszczony w systemie kontroli wersji. Podsystem ten również generuje raporty z budowania, zawierające np. błędy, które to raporty są dystrybuowane do odpowiedzialnych za ich analizę osób (więcej w module ???).

Powyższy podział wyznacza w naturalny sposób podstawowe grupy odpowiedzialności za system, w zespole kto inny piszę aplikacje, kto inny ją testuje, a jeszcze kto inny docelowo wdraża. Wszystkie te role jak i powyższe systemy obracają się wokół takich pojęć jak:

- **Work Item** – oznacza podstawową jednostkę „pracy”, która z jednej strony jest na tyle niezależna, że można mówiąc o niej osobo, ale z drugiej strony może posiadać powiazania z innymi takimi jednostkami. Niestety trudno jest znaleźć dobre tłumaczenie na język polski, stąd też czasem będziemy pisali po angielsku, czasem będziemy o niej mówili jako o pracy, a czasem daną jednostkę będziemy nazywali w zależności do tego czym ona jest. I to kolejny ważny element związany z ALM, otóż Work Item, może być błędem (Bug), czy elementem z Wykazu prac produktu (Product Backlog Item). Tu widać wyraźnie oddziaływanie zastosowanego szablony – pisaliśmy o tym w module 4.
- **Build** – oznacza kolejną kompilację/budowę całej aplikacji, zwykle budowa następuje raz na dobę, a w wyniku budowy i przeprowadzonych na aplikacji testów, osoby odpowiedzialne otrzymują raporty z budowy i testów. Takie podejście zapewnia ciągła kontrolę nad budowaniem docelowego systemu. Zatem na budowę czy też kompilację, należy patrzeć nie jak na finalny produkt, ale kolejny krok zbliżający nas do końcowego rezultatu.
- **Version** – oznacza konkretną wersje produktu dla Klienta, jest to oznaczenia mające charakter bardziej marketingowy i upraszczający komunikację, wszak łatwiej jest powiedzieć wersja 1.0 niż kompilacja 14521.

Ponieważ, środowisko TFS 2010 jest bardzo elastyczne, to zespół może zdecydować, czy wykorzystuje prostszy (Rysunek 2a.), czy też bardziej skomplikowany przepływ elementów w ramach budowanego projektu (Rysunek 2b.).



Rysunek 2. Przepływ w TFS 2010, a) wersja prosta, b) wersja złożona. (źródło: strona Microsoft)

Cykl życia oprogramowania a Scrum

Do tej pory sporo już powiedzieliśmy o metodyce Scrum, pozostaje zatem pytanie, jak ta metodyka odnosi się do cyklu życia oprogramowania? Przypomnijmy, że Scrum jest metodyką zarządzanie projektami. Jako taka metodyka mówi o podstawowych procesach i ich kolejności w trakcie tworzenia oprogramowania. Stąd też można traktować, że wyznacza ona część cyklu życia oprogramowania, dla przykładu Scrum nie mówi wprost o niektórych aspektach powiedzianych powyżej, jak np. testy jednostkowe. Niemniej jednak nie przeszkadza to w dobrej implementacji Scrum w środowisku VS2010+TFS2010. Szablon Visual Studio Scrum 1.0 zawiera takie pojęcia jak Test case czy Share steps, które nie występują w Scrum jako takim. Poniżej wypisujemy z jakimi podstawowymi grupami procesów możemy się spotkać stosując szablon o którym mówimy oraz metodykę Scrum:

- **Definiowanie i śledzenie wymagań klienta** – grupa procesów, które umożliwiają definiowanie, aktualizację i śledzenie postępów realizacji wymagań wobec produktu. Do tego celu używamy funkcjonalności związane z Wykazem prac produktu (Backlog) oraz jego elementami (Product Backlog Item). Już zapoznaliśmy się z większością tych zagadnień w module 4.
- **Ewidencja i śledzenie błędów** – grupa procesów, które pozwalają na ewidencję, aktualizację oraz śledzenie błędów znalezionych w produkcie. Będą to procesy skupione wokół pojęcia błąd (Bug), które to stanowi kolejny rodzaj typu Work Item. Elementy tego znajdziesz w module 6, a więcej na ten temat znajdziesz w ???.
- **Szacowanie i śledzenie zadań** – grupa procesów, które pozwalają na definiowania, aktualizację oraz śledzenie zadań i podzadań. Zauważmy, że zadanie (Task) stanowi kolejny rodzaj typu Work Item. Cześć zagadnień dla tych procesów już doświadczyliśmy w modułach 4 i 5.
- **Definiowanie i śledzenie sprintu** – grupa procesów, które pozwalają na definiowanie celu sprintu, dat rozpoczęcia i zakończenia oraz śledzenie jego przebiegu i wprowadzania informacji z retrospekcji. Spint jest kolejnym elementem typu Work Item. Więcej o tych procesach znajduje się w dalszej części tego modułu.
- **Ewidencja i zarządzanie ograniczeniami** – grupa procesów, które pozwalają na ewidencję i zarządzanie znanymi lub potencjalnymi problemami czy ryzykami, będącymi tym samym ograniczeniami dla powstającego systemu. Ograniczenie (Impediment) to następny element typu Work Item. Więcej o ograniczeniach znajdziesz w module ???.
- **Testy** – grupa procesów, które pozwalają na definiowanie przypadków testowych dla elementów z wykazu prac. Jak wcześniej napisaliśmy jest to rozszerzenie w stosunku do czystego Scrum. Przypadek testowy (Test Case) jest znów elementem typu Work Item. Więcej o testach znajdziesz w module ???.
- **Definiowanie wspólnych kroków** – umożliwia definiowanie sekwencji kroków, będących częścią wspólną kilku przypadku testowych. Jest to rozszerzenie w stosunku do standardowego Scrum i wspólne kroki (shared steps) również są elementem typu Work Item. Więcej na ten temat znajdziesz w module ???.

Jak widać wszystkie powyższe procesy mają swoich reprezentantów w postaci elementów typu Work Item, i możemy je dodawać poprzez menu New Work Item i wybór właściwego elementu (zobacz rysunek 7. w module 4).

Osobną grupę stanowią procesy pozwalające na monitorowanie i raportowanie postępu przebiegu prac w ramach tworzonego systemu. Będą to następujące procesy:

- **Śledzenie postępu prac i zadań** – postęp prac zawartych w wykazie prac jak i odpowiadających im zadań możemy obserwować za pomocą tzw. wykresów malejących (burndown). Wyróżniamy tu dwa wykresy, jeden dla wydania (release), a drugi dla sprintu. Więcej na ten temat dalej w tym module.

- **Pomiar szybkości zespołu** – szybkość wykonania prac przez zespołu daje obraz ilość wysiłku jaki zespół poświęcił dla realizacji każdego sprintu. Do obrazowania szybkości zespołu służy raport szybkości (velocity report).
- **Śledzenie budowania** – śledzenie sukcesów poszczególnych kompilacji pozwoli na określanie jakości produktu. Do tego służą raporty: raport podsumowujący z kompilacji (Build Summary Report) oraz raport kompilacji zakończonych sukcesem w czasie (Build Success Over Time Report).
- **Śledzenie testów** – w trakcie tworzenia aplikacji wykonywana jest duża ilość testów ręcznych i automatycznych, istnieje zatem potrzeba nadzoru nad postępem tych procesów. Testy możemy śledzić za pomocą dwu raportów: Test Case Readiness Report, Test Plan Progress Report.

Podsumowanie

W tym rozdziale przypomniane zostały fakty z inżynierii oprogramowania w zakresie cyklu życia oprogramowania. Przedstawione również zostały informacje na temat cyklu życia aplikacji (ALM) proponowanego przez firmę Microsoft a realizowanego w środowiskach Visual Studio 2010 oraz Team Foundation System 2010. Następnie przedstawione zostało podejście realizowane w oparciu o szablon Visual Studio Scrum 1.0 oraz procesy jakie są w nim proponowane i będące elementami cyklu życia aplikacji.

Przykładowe rozwiązanie

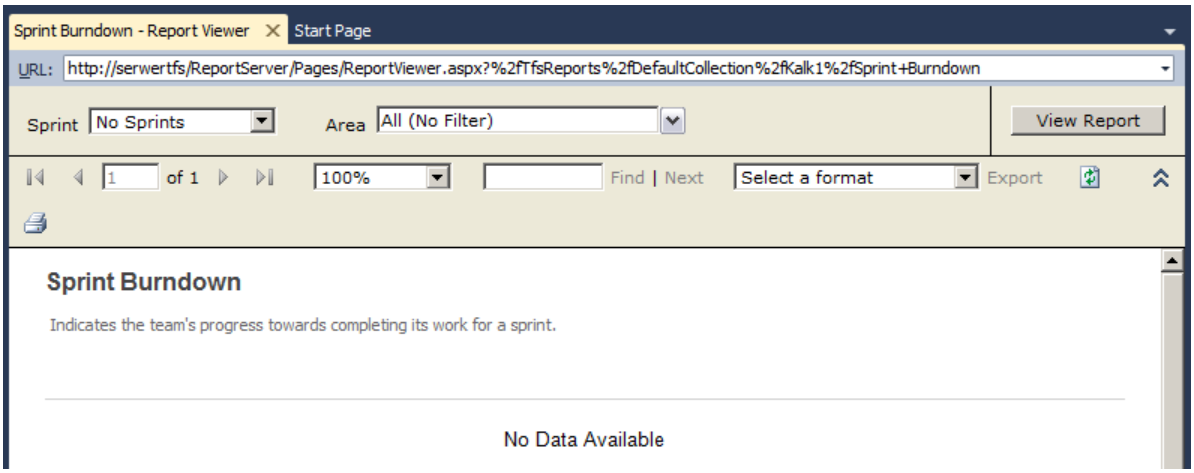
Planowanie pierwszej iteracji

Wróćmy do przykładu rozpoczętego w poprzednim module i zróbmy krótkie przypomnienie. Za pomocą środowiska VS2010 zdefiniowaliśmy dwa elementy w wykazie prac i przypisaliśmy im zadania, były to:

1. Szkielet kalkulatora (praca)
    a) Wykonanie głównego okna aplikacji (zadanie)
2. Ustalić z Klientem wygląd przycisków (praca)
    a) Ustalenie wyglądu przycisków (zadanie)

Możemy je obejrzeć w Wykazie prac produktu (w środowisku VS2010, w oknie Team Explorer, rozwijamy drzewo Kalk1, następnie węzeł Work Item i węzeł Team Queries, na koniec dwa razy klikamy Product Backlog). Zadania przeglądamy w zakładce Tasks w dowolnym elemencie Wykazu prac produktu.

Z punktu widzenia zarządzania projektem, a zatem śledzenia cyklu wytwórczego oprogramowania, jest spojrzenia na to co się dzieje poprzez pryzmat raportów – jak wiemy jest to wygodne i proste. Zatem rozwińmy węzeł w drzewie projektu (nadal w oknie Team Explorer) węzeł raportów (Reports), widzimy w nim dwa węzły: Builds i Tests – które w taj chwili nas nie interesują oraz trzy elementy: Release Burndown, Sprint Burndown i Velocity – które to raporty sygnalizowaliśmy w poprzednim punkcie. Kliknijmy zatem na pierwszy z nich, czyli Release Burndown, w głównym oknie VS2010 ukaże się wynik działania, w którym poza informacjami nagłówkowymi, oraz informacją co ten raport powinien zawierać otrzymaliśmy informację „No Data Available”. Jeśli wykonamy podobne działanie dla raportu szybkości (Velocity) znów otrzymamy taki komunikat. Niestety podobnie dla sprintu (Rysunek 3.), pytanie czemu te raporty są puste?

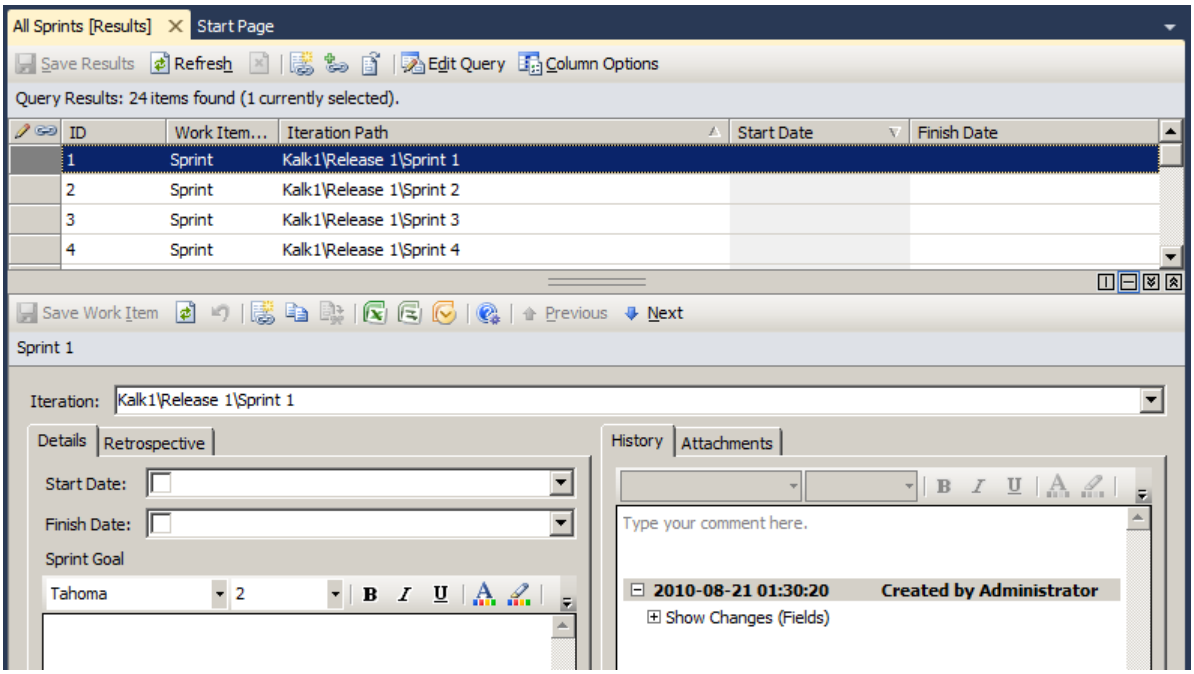


Rysunek 3. Widok pustego raportu Sprint Burndown.

Odpowiedź jest prosta, tak naprawdę z punktu widzenia metodyki Scrum nie zaczęliśmy jeszcze na dobre projektu. Przypomnijmy (patrz podpunkt Przebieg prac w Scrum w module 4), że tak naprawdę prace projektowe czy programistyczne zaczynamy dopiero w pierwszym Sprincie w cyklach wyznaczanych poprzez jednodniowe Scrumy, a przecież w omawianym przykładzie kalkulatora dopiero stworzyliśmy Wykaz prac produktu (oczywiście przykładowy i niekompletny). Zatem nie odbyliśmy pierwszego spotkania projektowego, ani też nie określiliśmy celu pierwszego Sprintu! Tak naprawdę wszystko to co zrobiliśmy poprzednio obejmuje działania w ramach kroku „Właściciel Produktu sporządza początkowy wykaz prac produktu oraz budżet" (patrz podpunkt Przebieg prac w Scrum w module 4).

W takim razie nadszedł czas, by przystąpić do definiowania pierwszego sprintu, zatem wraz z Właścicielem Produktu przystępujemy do planowania pierwszej iteracji. W pierwszym kroku Właściciel produktu ma określić cel sprintu oraz wykaz prac, które mają być realizowane podczas tej iteracji. Właściciel Produktu jako cel sprintu określił „Stworzenie graficznego wyglądu kalkulatora", a wykaz prac jakie mają znaleźć się w tym sprincie, to obie prace które znajdują się w wykazie. Teraz według zaleceń metodyki Scrum czas na Szefa Scruma i zespół, którzy mają podjąć decyzję jak podzielić prace na zadania i jak je wykonać. Przyjmijmy dla uproszczenia, że zadania przypisane do prac, które Właściciel Produktu wybrał do realizacji, są wystarczające i pokrywają wszystkie potrzeby. Zatem dwie prace wymienione wcześniej trafią do wykonania w tym sprincie. Zanim jednak wprowadzimy powyższe ustalenia do środowiska VS2010+TFS2010, spójrzmy co mażemy uzyskać w ramach już posiadanych zasobów.

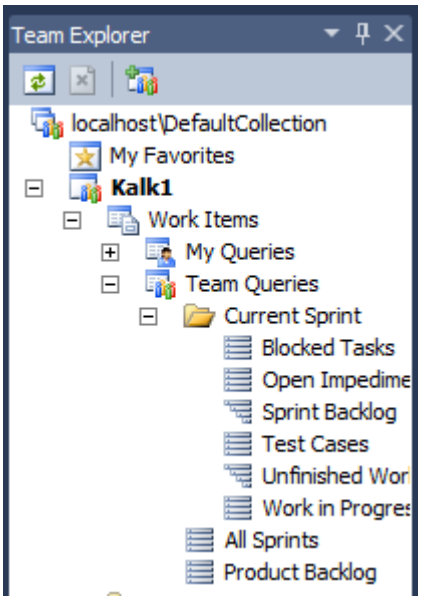
Zauważmy, że w drzewie projektu w ramach okna Team Explorer w węźle Work Items a następnie Team Queries znajduje się pozycja All Sprints (wcześniej używaliśmy pozycji Product Backlog), kliknijmy tę pozycję, otrzymamy okno zawierające informacje o wszystkich zdefiniowanych wydaniach (Release) oraz sprintach w ramach tych wydań (Rysunek 4.).



Rysunek 4. Okno zawierające wykaz wszystkich sprintów w projekcie.

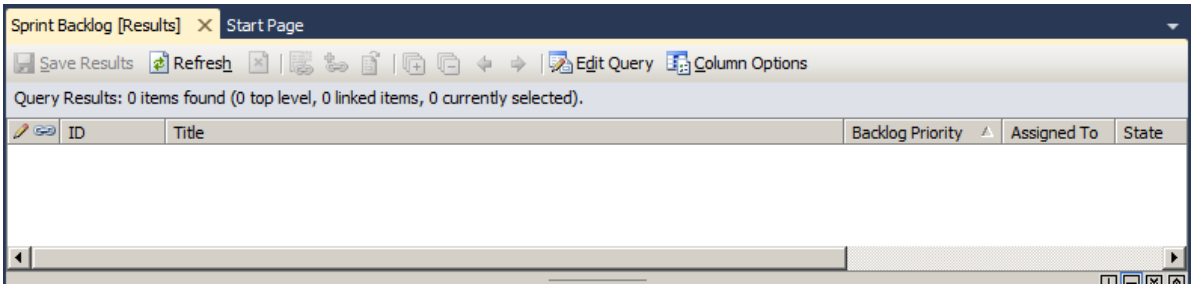
Jeśli dla razy klikniemy na pozycji All Spints i przejrzymy listę sprintów, to przekonamy się, że w projekcie mamy zdefiniowane cztery wydania, każde zawierające po sześć sprintów. Powstaje oczywiste pytanie skąd wzięły się te wydania oraz sprinty, to jest również „dzieło" szablonu, który użyliśmy do tworzenia naszego projektu. Zauważmy, że w historii dla dowolnego sprintu widnieje data i godzina utworzenia, i nie jest to obecna data, jest to moment w którym projekt ten był tworzony. Osoby zainteresowane modyfikacją lub stworzeniem własnego podziału odsyłamy do Laboratorium zaawansowanego.

Zauważmy teraz, że żaden ze sprintów nie posiada określonej daty rozpoczęcia oraz zakończenia, zatem tak naprawdę projekt z punktu widzenia środowiska w ogóle nie się jeszcze nie rozpoczął! By przekonać się, iż rzeczywiście sprint nie ma przypisanych automatycznie żadnych zadań, możemy ponownie użyć akcji udostępnionej na drzewie projektu, ale do tej pory jeszcze nie wykonywanej. W drzewie projektu, w ramach okna Team Explorer, rozwijamy węzeł Work Items, następnie Team Queries i dalej Current Sprint – widzimy w tym węźle wiele pozycji, które w dalszej części będziemy używali (Rysunek 5.).



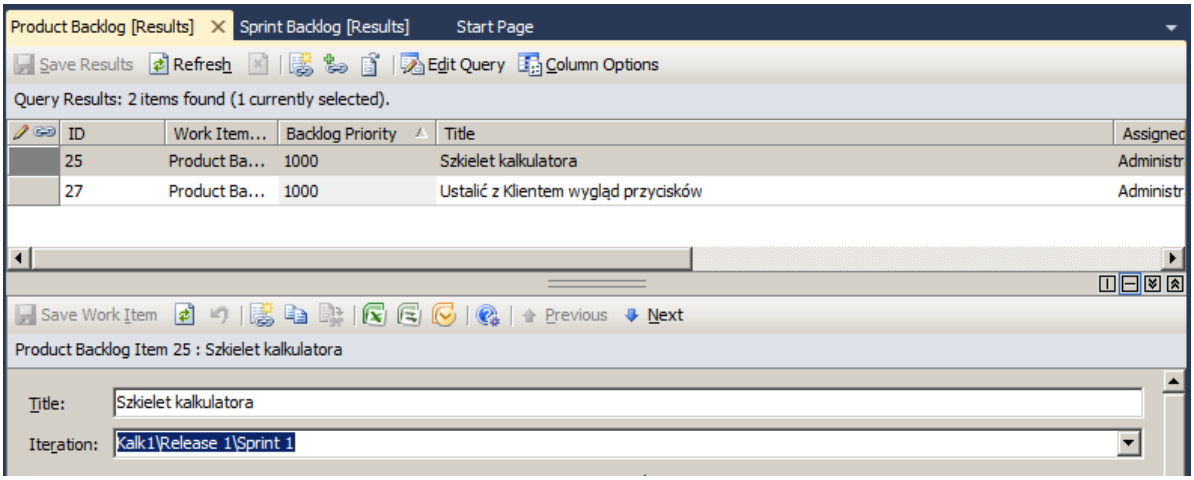
Rysunek 5. Fragment drzewa projektu w oknie Team Explorer zawierającego rozwinięty węzeł Current Sprint.

W rozwiniętym węźle Current Sprint widzimy element Sprint Backlog, który zawiera informacje o pracach przypisanych do tego sprintu – kliknijmy dwa razy a ten element, widać wyraźnie, iż lista prac w tym sprincie jest pusta (Rysunek 6.).



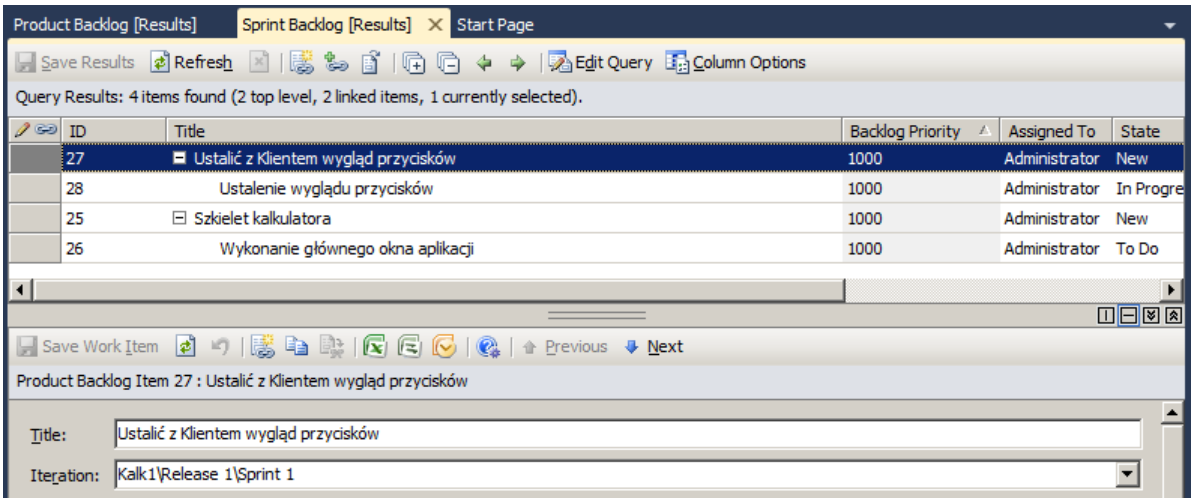
Rysunek 6. Pusta lista prac dla danego sprintu.

Wróćmy w takim razie do Wykazu prac produktu (np. poprzez dwukrotne kliknięcie na elemencie Product Backlog w drzewie projektu). Wejdźmy w pierwszą pozycję (Szkielet kalkulatora), a następnie w oknie dialogowym zawierającym szczegóły dla tej pracy w oknie wyboru iteracji (Iterations) wybierzmy pierwszy sprint w ramach pierwszego wydania (Rysunek 7.), na koniec naciśnijmy przycisk Save Work Item.



Rysunek 7. Pierwsza praca przypisana do iteracji Sprint 1 w ramach wydania 1.

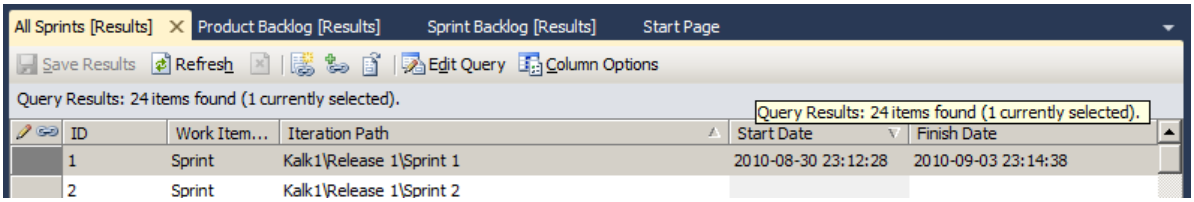
Tą samą czynność wykonajmy dla drugiej pracy (Ustalić z Klientem wygląd przycisków). Następnie wróćmy do wykazu prac dla bieżącego sprintu (dwukrotne kliknięcie na elemencie Sprint Backlog w ramach węzła Current Sprint w drzewie projektu). W wykazie tym pojawią się przypisane do sprintu prace wraz z zadaniami, które na nie się składają (Rysunek 8.). Jednocześnie prace, które przydzielone zostały do sprintu zniknęły z wykazu prac produktu, zatem po odświeżeniu okna Produckt Backlog (Rysunek 7.) nie pojawią się one.



Rysunek 8. Lista prac dla bieżącego sprintu.

Spróbujmy zobaczyć, czy którykolwiek z wcześniej wywoływanych raportów będzie zawierał jakieś dane (Release Burndown, Sprint Burndown, Velocity), znów nic?! Niestety nie trzymamy się zasad metodyki, co miało być pierwszym krokiem? Wróćmy do podpunktu Przebieg prac w Scrum w module 4, dokładnie do punktu 2, jest tam wyraźnie napisane, iż „Właściciel Produktu, Szef Scruma oraz pozostali członkowie zespołu podejmują decyzję o czasie trwania pierwszej iteracji (i tym samym wszystkich pozostałych).”.

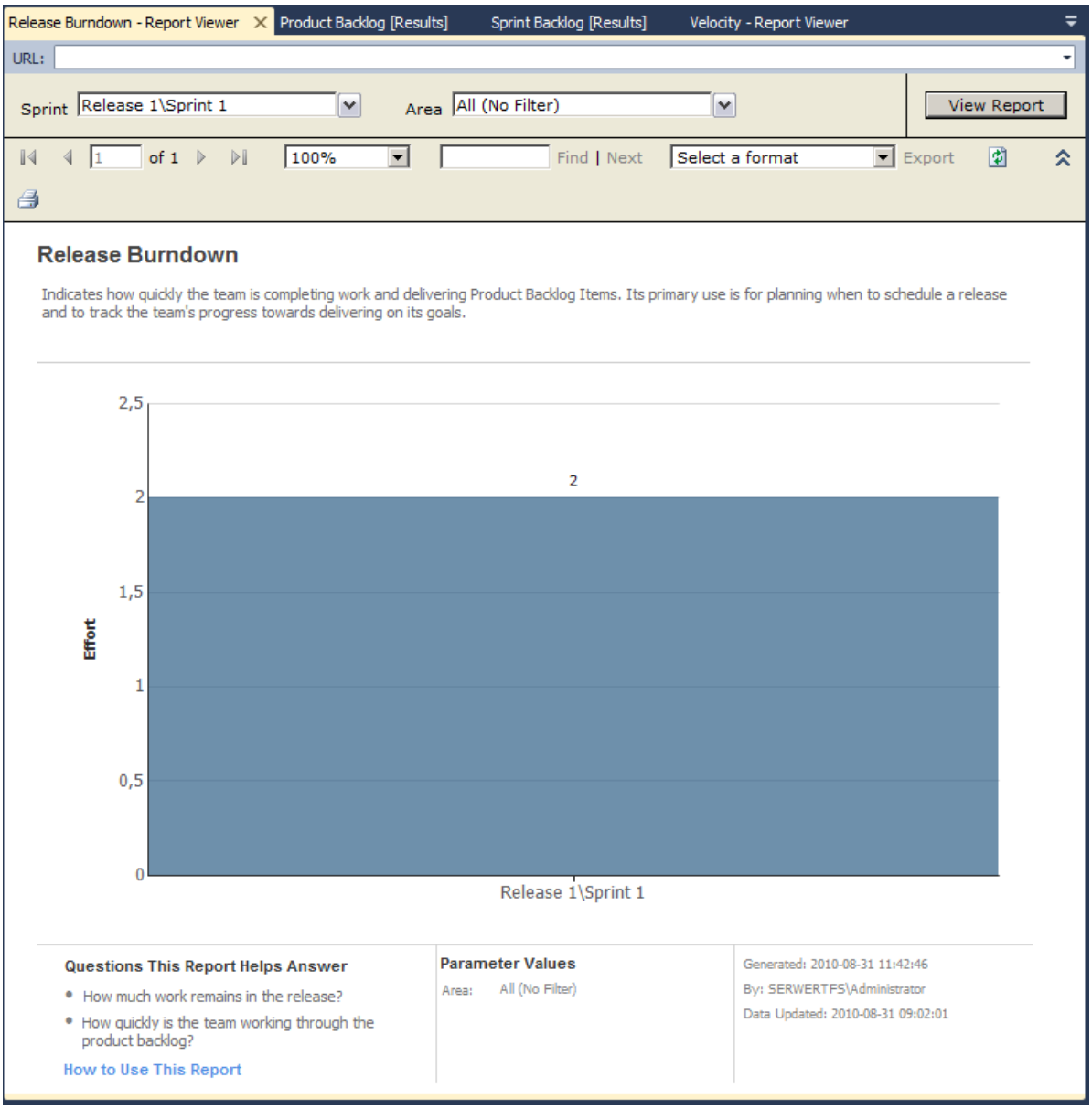
Spójrzmy jeszcze raz w listę wszystkich sprintów, na przykład poprzez dwukrotne kliknięcie w pozycji All Sprints w drzewie projektu, pojawi się zawartość znana z rysunku 4. Jak widać wciąż jest nie określona data rozpoczęcia i zakończenia pierwszego sprintu. Zatem wybierzmy te daty, w przypadku początku wystarczy kliknąć w okienku zaznaczania przy dacie początkowej (Start Date) i automatycznie zostanie wybrana bieżąca data, oczywiście możemy określić inną (nawet wcześniejszą), oraz określmy datę końcową, na przykład przyjmijmy, że nasze sprinty będą trwały 5 dni roboczych. Po wybrani dat naciśnijmy przycisk Save Work Item, daty zapisane zostaną uwidocznione na liście sprintów (Rysunek 9.).



Rysunek 9. Wykaz wszystkich sprintów wraz z określonymi datami początku i końca dla pierwszego z nich.

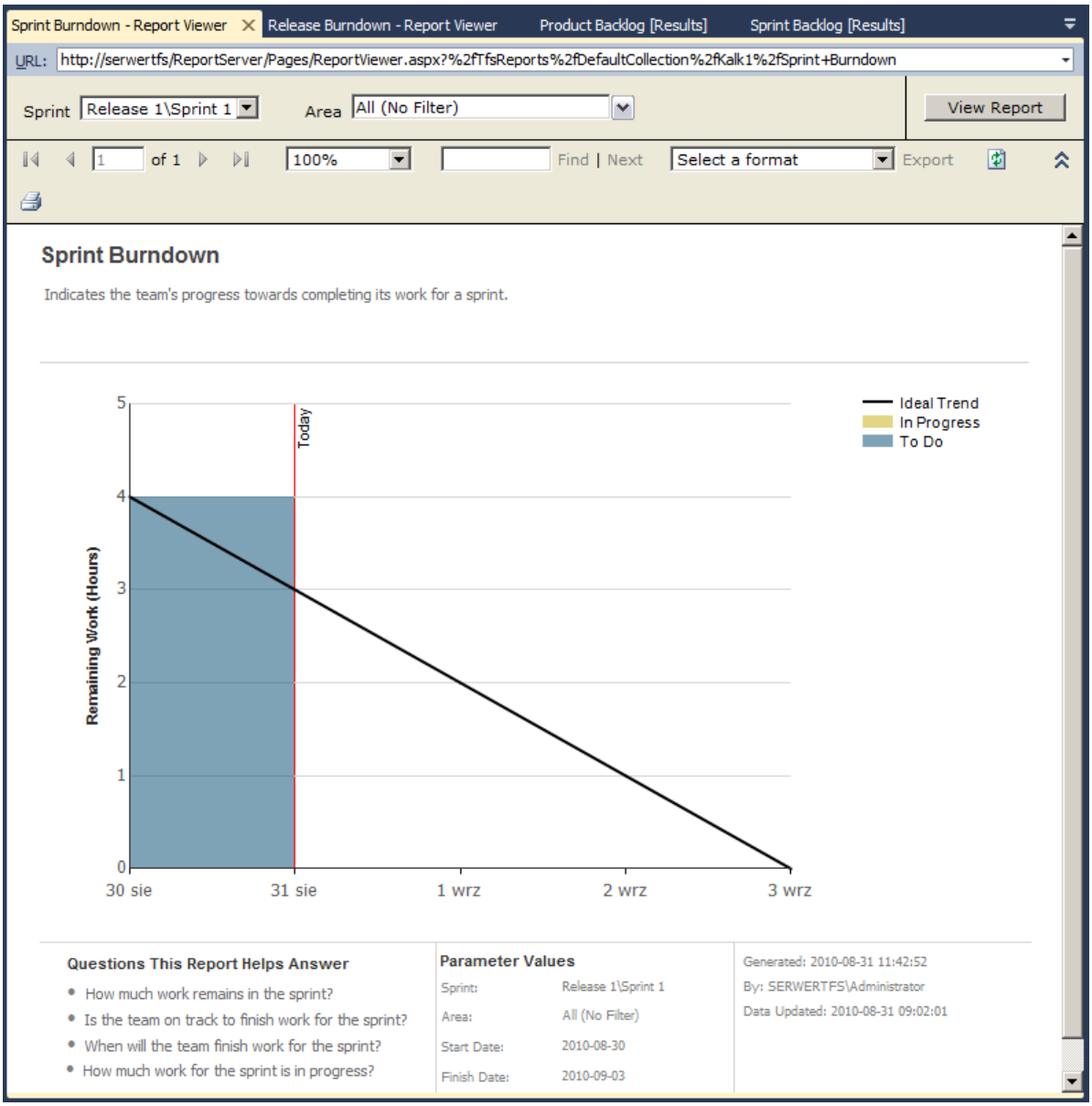
Dzięki określeniu dat możemy wreszcie uzyskać pierwszy z raportów – Release Burndown (Rysunek 10.). Sam wykres wyrażony jest w jednostkach Effort, których wartości określaliśmy dla każdej pracy (Rysunek 8 w module 4). Omówmy pokrótce zawartość tego okna. W liście rozwijanej Sprint możemy wybrać dla którego ze sprintów generujemy raport, podobnie w liście Area (w naszym przypadku mamy tylko jeden obszar). Następnie mamy przełącznik stron – przydatny, gdy mamy kilka wykresów, listę skali powiększenia, oraz listę wyboru formatu w którym chcemy uzyskać ten raport. Dalej znajduje się przycisk odświeżania wykresu. Na dole, pod wykresem, znajdziemy

informacje o jego parametrach, oraz dacie, godzinie wykonania, informacji o tym, kto go wykonał oraz dacie aktualizacji. Podobne opcje będziemy mieli w przypadku pozostałych raportów.



Rysunek 10. Wykres malejący dla pierwszego wydania w ramach pierwszego sprintu.

Zauważmy, że wykres Valocity jest nadal pusty. Wygenerujmy teraz wykres malejący dla sprintu (Sprint Burndown), przy czym, wykres ten możemy generować dopiero następnego dnia, gdyż jest on wykonywany w odniesieniu do upływającego czasu (Rysunek 11.).



Rysunek 11. Wykres malejący sprintu po pierwszym dniu projektu.

Zauważmy, że wykres malejący dla sprintu podaje ilość godzin jakie pozostały do zakończenia sprintu, ilość ta jest określana na podstawie informacji, które sa zawarte w oknie Remaining Work dla każdego z zadań. W naszym przypadku są to 4 godziny, gdyż początkowo tylko to zadanie było przypisane do sprintu (dokładnie praca „Szkielet kalkulatora" zawierająca zadanie „Wykonanie głównego okna aplikacji"). Spójrzmy dalej na ten wykres, jednym kolorem zaznaczona jest ilość prac o statusie To Do, a drugim o statusie In Progress. Początkowo zadanie o którym piszemy miało status To Do, stąd całe pole zakolorowane jednym kolorem. Zmieńmy teraz status tego zadania na In Progress i odświeżmy wykres malejący. Jak widać zmienił się słupek i jest on teraz podzielony na dwie części w obu kolorach. Wróćmy raz jeszcze do naszego zadania, i zmieńmy ilość pozostałej pracy na 2 godziny, oraz przypiszmy do bieżącego sprintu druga pracę z naszego wykazu prac, czyli „Ustalić z Klientem wygląd przycisków".

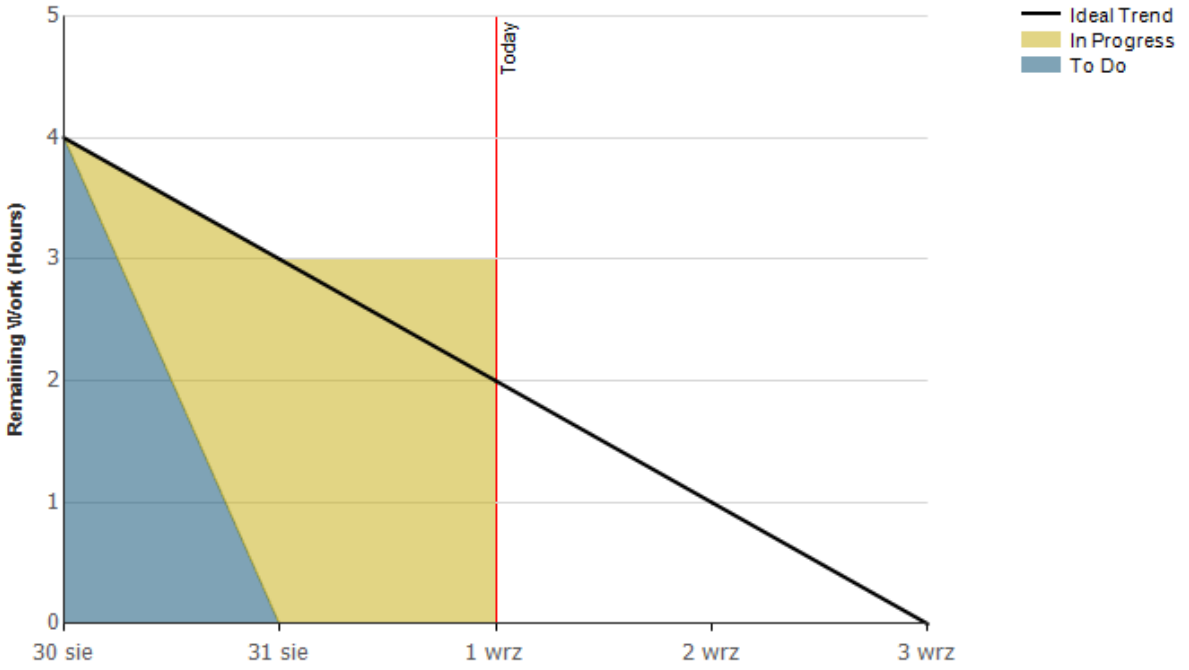
Najwyższy czas, by przybliżyć, nieco bardziej szczegółowo, jak interpretować te trzy wykresy o których piszemy, same opisy mogą nie być wystarczająco jasne, zatem:

- **Release Burndown** – umożliwia określenie postępu prac dla bieżącego wydania (Release), zatem pozwala na określenie ile pracy pozostało w tym wydaniu, oraz jak szybko zespół wyczerpuje prace z Wykazu prac produktu.

- **Sprint Burndown** – podobnie jak powyższy raport, tak i ten umożliwia określenie postępu prac w wydaniu. Jednak obserwujemy to z innego pryzmatu, mianowicie:
  - ile pracy pozostało w tym sprincie?
  - Czy zespół jest blisko idealnego trendu, czy tez są duże odchylenia?
  - Kiedy prawdopodobnie zespół zakończy prace?
  - Ile zadań w tym sprincie jest w trakcie realizacji (status In Progress)?
- **Velocity** – pokazuje postęp prac jakie już wykonał zespół i wyniki te wynikają z ilości pracy jaką została wykonana, by zakończyć każdy sprint. Wykres ten pozwala odpowiedzieć na pytania o maksymalną, minimalną i średnią szybkość zespołu.

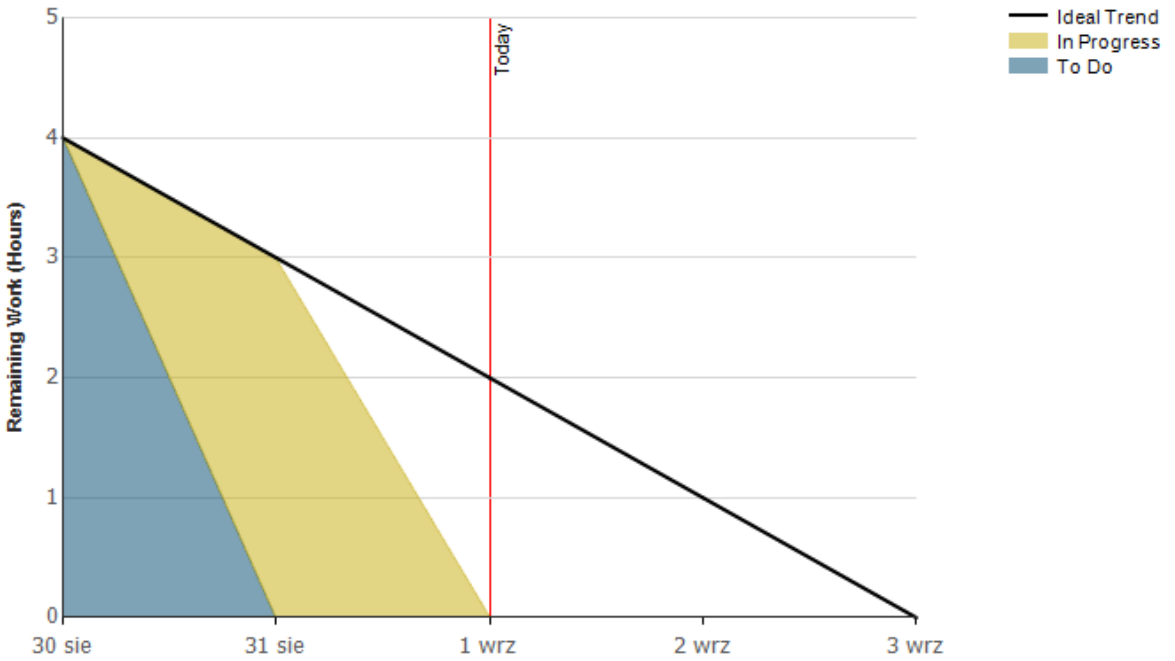
Jeśli teraz zastanowimy się dokładniej, o przestanie dziwić, czemu wykres Velocity nie zawiera na razie żadnych danych – przecież nie zakończyliśmy ani jednego sprintu.

Wyobraźmy sobie, że minął kolejny dzień, wykonaliśmy pewną część pracy związanej z zadaniami i wartości w polach Remaining Work zostały zaktualizowane. W tej sytuacji zmieni się wykres malejący dla sprintu, spójrzmy jak wygląda (Rysunek 12.).

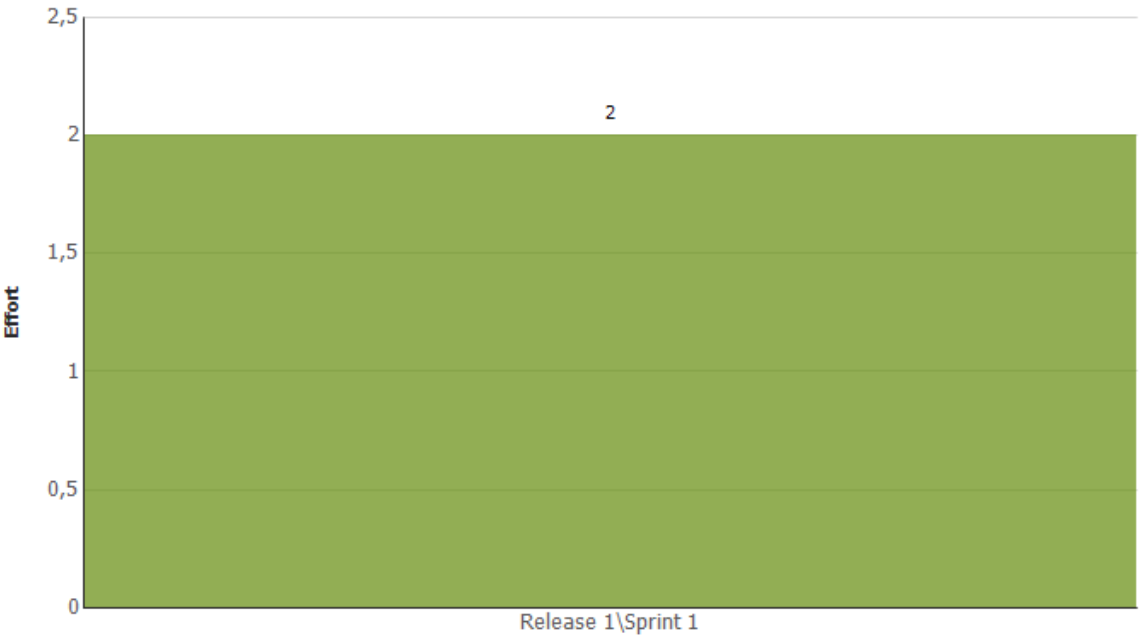


Rysunek 12. Wykres malejący sprintu po drugim dniu projektu.

Ostatecznie wykonaliśmy wszystkie zadania, zatem zmieniamy ich status na Done, oraz status prac na Done. Po tych czynności, gdy raporty zostaną odświeżone, zobaczymy, że wykres malejący sprintu będzie miał końcową postać – ilość pozostałej pracy wynosi zero (Rysunek 13.), a wykres szybkości wreszcie zostanie wygenerowany i będzie zawierał informację o ilości pracy jaką poświęciliśmy na wykonanie tego sprintu (Rysunek 14.).



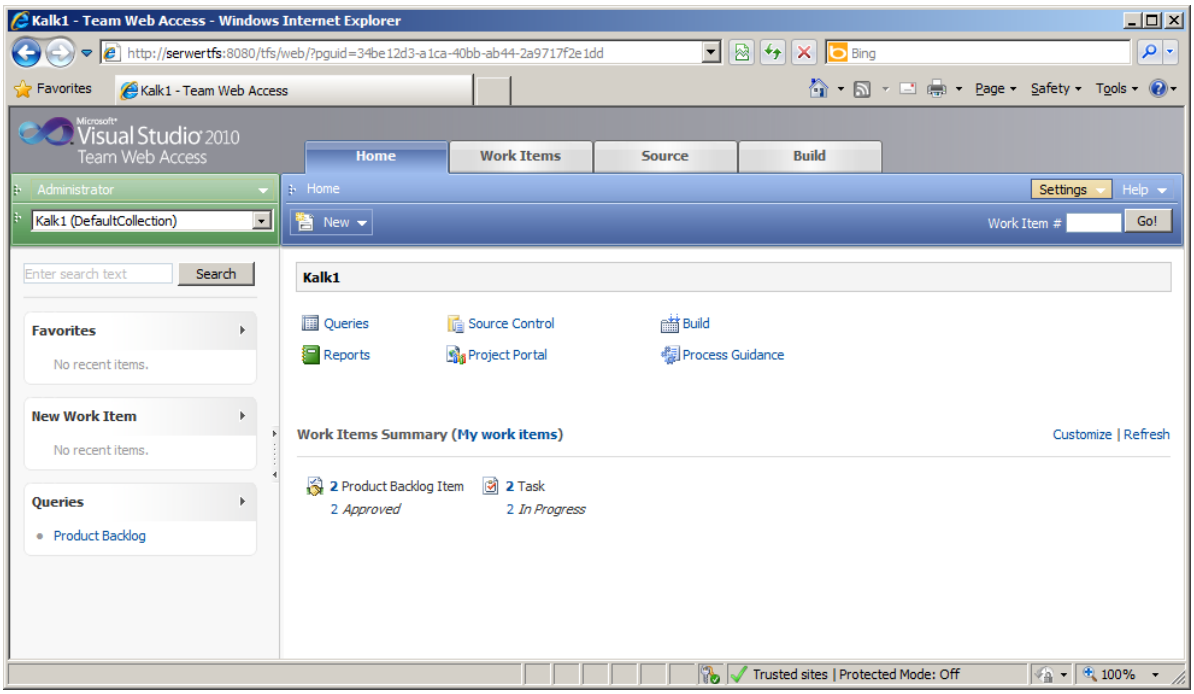
Rysunek 13. Wykres malejący dla sprintu w którym zakończono prace.



Rysunek 14. Wykres szybkości po zakończonym pierwszym sprincie.

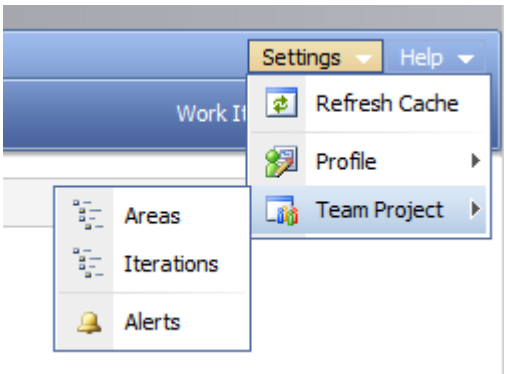
Na sam koniec warto przypomnieć, że każdy projekt ma swój portal (patrz np. rysunek 14 w module 3). W tym portalu każdy z członków projektu będzie widział swoje zadania, błędy oraz przypadki testowe. W tym oknie (My Dashboard) widzimy, poza tymi informacjami, które wymieniliśmy, również linki do pozostałych części informacji. Dla przykładu link do raportów (Reports) prowadzi do dostępnych raportów dla projektu, tych samych do których mieliśmy dostęp w ramach Visual Studio. Może pojawić się pytanie, w jakim celu ta sama informacja jest i w portalu i w Visual Studio. Odpowiedź wynika z faktu, iż w dużych zespołach nie każda osoba jest programista i nie będzie dla niej naturalnym środowiskiem Visual Studio. Zresztą osoba śledząca prace nie potrzebuje w ogóle dostępu do niczego innego poza raportami – system uprawnień pozwala bardzo elastycznie nadawać uprawnienia dla poszczególnych członków zespołu (część tych możliwości zaprezentowaliśmy w module 3).

Warto zajrzeć jeszcze na stronę projektu nie tylko z perspektywy własnego panelu ale również poprzez Team Web Access (link o tej nazwie jest widoczny po lewej stronie panelu użytkownika). Okno to zawiera bardziej ogólne informacje o projekcie (Rysunek 12.).



Rysunek 12. Zawartość strony Team Web Access dla projektu.

Na stronie tej mamy dostęp do zapytań (Queries), raportów i pozostałych elementów, które są odpowiednikiem tego co widzimy w VS2010. Natomiast w prawym górnym rogu znajduje się przycisk umożliwiający zmianę ustawień (Settings), to właśnie po naciśnięciu jego ukaże się menu w którym możemy zmienić ustawienia np. obszarów (Area) czy iteracji (Iterations), dla naszego projektu (Rysunek 13.).



Rysunek 13. Rozwinięte menu Team Project w ramach ustawień na stronie Team Web Access.

Oczywiście, przy posiadaniu odpowiednich uprawnień, również z poziomu tego portalu mamy możliwość przeglądania a nawet zmiany elementów typu Work Item, wykonywania, edycji i dodawania nowych zapytań, czy zarządzania kompilacjami.

Porady praktyczne

Praca w złożonych środowiskach programistycznych i do tego przy zastosowaniu nowej i nieznanej metodyki wymaga treningu i praktyki. Nie należy na początku oczekiwać spektakularnych rezultatów, ani też spodziewać się, iż zespół przyjmie wszystkie czynności w naturalny sposób. Wiele błędów popełnionych powyżej, co prawda specjalnie, wynikało z braku doświadczonego Szefa Scruma, którego zadaniem jak pamiętamy jest pilnowanie reguł metodyki. Stąd też, nie

oczekujmy iż pierwszy czy drugi projekt zostanie zrealizowany bezbłędnie i w zgodzie ze wszystkimi regułami tej metodyki. Zdecydowania warto poznać możliwości środowiska VS2010+TFS2010 zanim zaczniemy realizować w oparci o te środowiska pełnowartościowy projekt.

Osobną sprawą jest odpowiednia hierarchia uprawnień poszczególnych członków zespołu. Złożoność omawianej rodziny produktów powoduje, że w początkowych projektach zwykle nadajemy uprawnienia wszystkim do wszystkiego, wychodząc z założenia, że wtedy będzie to działać na pewno. Jednak na dłuższą metę takie podejście może źle się skończyć dla projektu. Na przykład, ktoś nieświadomie skazuje jakiś element. Stąd też w miarę poznawania środowiska, należy powoli opracowywać właściwy podział uprawnień odpowiadający rzeczywistym potrzebom członków zespołu.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- wiesz z jakich etapów składa się cykl życia oprogramowania,
- wiesz jaki cykl życia aplikacji proponuje firma Microsoft,
- wiesz jak cykl życia aplikacji odnosi się do metodyki Scrum,
- potrafisz zidentyfikować poszczególne etapy cyklu życia oprogramowania,
- potrafisz stworzyć pierwszy cel Sprintu oraz wykaz jego prac i zadań,
- rozumiesz rolę i wagę wykresów malejących oraz wiedział jak je uzyskać oraz jaką niosą ze sobą informację,
- rozumiesz zakres prac i rolę procesów w ramach cyklu życia oprogramowania.

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. http://en.wikipedia.org/wiki/Formal_methods

   Strona omawia ogólne cechy formalnych metod dla procesów wytwarzania oprogramowania.

2. http://vs2010quickref.codeplex.com/

   Strona zawiera ciekawy podręcznik dla Visual Studio 2010.

Laboratorium podstawowe

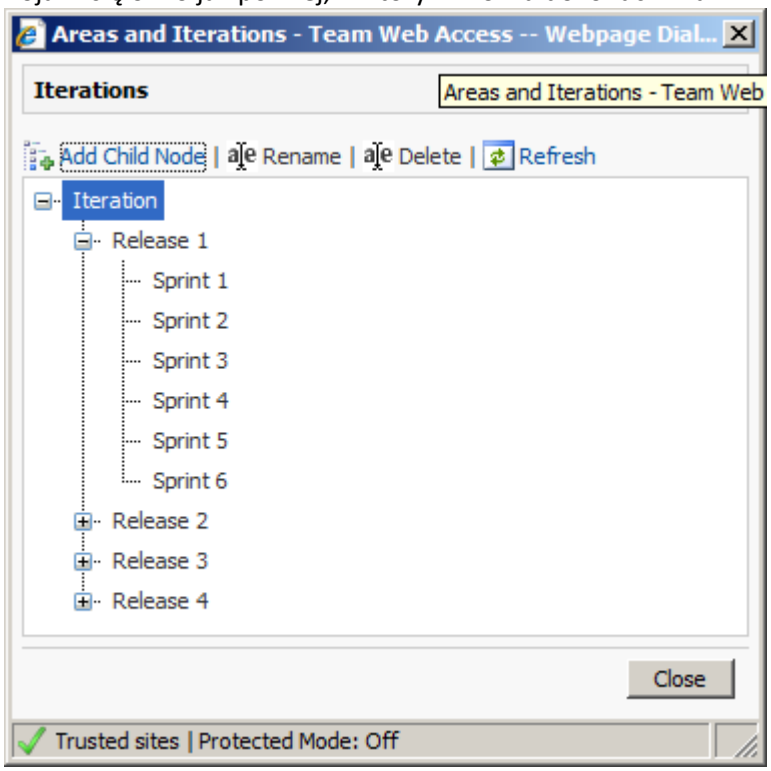
Kontynuujemy pracę nad kalkulatorem. Twoim zadaniem jest samodzielne przećwiczenie omówionych zagadnień. W tym celu, w pierwszym etapie zweryfikujesz poprawności procesu zaprezentowanego w Przykładowym rozwiązaniu. Następnie samodzielnie uzupełnisz wykaz prac o kolejny element przeniesiesz go do wykazu prac sprintu dbając przy tym o właściwy cykl czynności.

Zadanie	**Tok postępowania**
1. Sprawdzenie prac w Wykazie prac produktu	• Otwórz Wykaz prac produktu (Product Backlog). W oknie Team Explorer rozwiń drzewo projektu Kalk1, a następnie węzeł Work Items i dalej Team Queries. • W węźle Team Queries odszukaj element Product Backlog – kliknij na niego dwukrotnie. • W oknie Product Backlog sprawdź czy lista zawiera jakieś elementy? Jeśli nie, to oznacza, że wszystkie prace przeniosłeś do Wykazu prac sprintu. W przeciwnym razie powinieneś to teraz zrobić. W tym celu w oknie pracy zmień iteracje na Kalk 1\Release 1\Sprint 1.
2. Sprawdzenie statusu prac w Wykazie prac sprintu	• Otwórz Wykaz prac sprintu (Sprint Backlog). W oknie Team Explorer rozwiń drzewo projektu Kalk1, a następnie węzeł Work Items i dalej Team Queries i jeszcze dalej Current Sprint. • W węźle Current Sprint odszukaj element Sprint Backlog – kliknij na niego dwukrotnie. • W oknie Sprint Backlog sprawdź status każdej pracy, która jest tu zawarta. Czy mają one status Commited? Nie? To dlaczego rozpoczęliśmy pracę nad nimi? Czy to jest zgodne z metodyką Scrum? • Porównaj przebieg powyższego przykładowego problemu oraz cyklu zmian statusu w metodyce Scrum (Rysunek 14 w module 4). Zastanów się w którym momencie popełniliśmy błąd? • Kiedy zadania należące do pracy może zmienić status na In Progress? Jaki status powinna mieć praca, by rozpocząć prace nad zadaniem wchodzącym w skład tej pracy?
3. Dodanie nowej pracy do Wykazu prac produktu	• Postępując w sposób zaprezentowany w module 4, oraz ćwiczeniach tam zawartych dodaj nowa pracę do wykazu prac produktu. • Praca, którą dodasz powinna móc być wykonana w bieżącym sprincie (Sprint 1).
4. Pobranie pracy do realizacji w sprincie	• Chcesz pracę, która została dodana do Wykazu prac produktu pobrać do realizacji do bieżącego sprintu. Zastanów się kiedy jest to możliwe do realizacji, by było to zgodne z metodyką Scrum? • Kiedy Klient powinien wyrazić akceptację do wykonania danej pracy? A kiedy zespół aprobuję ją do sprintu? • Nadaj odpowiedni status i przypisz tę pracę do Wykazu prac sprintu. • Praca, którą dodałeś nie ma żadnego zadania z nią skojarzonego. Tak może się stać, gdyż nie ma obowiązku by praca będąca w Wykazie prac produktu już wtedy posiadała podział na zadania. To zespół w trakcie planowania prac sprintu może dokonać odpowiedniego podziału (patrz moduł 4). • Dodaj nowe zadanie do tej nowej pracy, które pozwoli na jej realizację.
5. Określanie	• Gdy zadanie zostanie dodane, czas na to byś określił jego szczegóły.

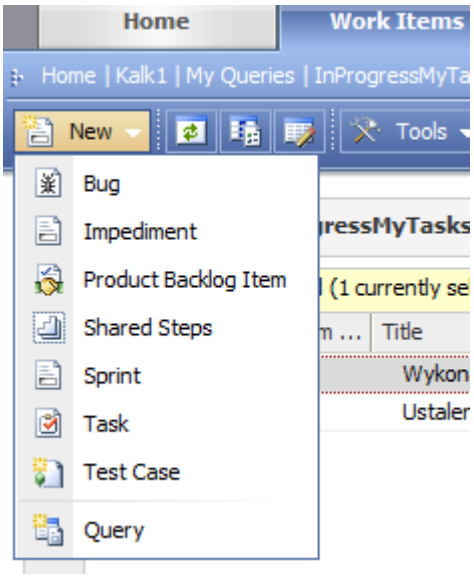
szczegółów zadania	• Przypisz osobę mającą zrealizować to zadanie, status pozostaw na To Do. • Określ ilość pracy jaka jest wymagana do realizacji tego zadania, jego priorytet i aktywność (patrz rysunek 11 w module 4). • Zapisz to zadanie.
6. Rozpoczęcie pracy nad zadaniem	• Niech osoba, której zostało przydzielone to zadanie do realizacji wejdzie w portal projektu. Czy widać w panelu projektu tej osoby, to zadanie? Jeśli tak to świetnie, możemy iść dalej. W przeciwnym razie cofnij się do poprzedniego kroku i sprawdź czynności. • Osoba, która ma realizować zadanie, niech zmieni status z To Do na In Progress. • Spójrz na wykresy, czy coś się zmieniło?
7. Zakończenie zadania i pracy	• Osoba, która realizuje zadanie kończy nad nim pracę. • Należy w tym celu zmienić status zadania na Done. • Czy coś zmieniło się w wykazie zadań tej osoby w portalu projektu? • Gdzie możecie znaleźć zadania, które są zakończone? • Czy zakończenie zadania wymusza zakończenie pracy z nim związanej?

Laboratorium rozszerzone

Jak powiedzieliśmy w punkcie Przykładowy problem, domyślny podział projektu zawiera cztery wydania, po szczęść sprintów każde. Jednak Ty dla potrzeb swojego projektu potrzebujesz dokonać innego podziału.

Zadanie	Tok postępowania
1. Otwarcie portal TWA	• Musisz otworzyć portal (stronę) Team Web Access (TWA), można to uczynić na kilka sposobów, podajemy dwa: – W przeglądarce wpisujemy adres http://serwertfs:8080/tfs/web/, gdzie słowo „serwertfs” wymieniamy na nazwę naszego serwera. – Otwieramy projekt w środowisku VS2010, w drzewie Team Explorer na nazwie naszego projektu klikamy prawym klawiszem myszy i następnie z menu wybieramy „Show Project Portal”. Po otwarciu portalu projektu znajdziemy się w oknie naszych zadań i innych elementów. Po lewej stronie klikamy na link Team Web Access.
2. Edycja iteracji	• W prawym górny rogu okna TWA znajduje się przycisk zmiany ustawień (Settings). • Kliknij go, by ukazało się menu, najedź następnie na podmenu Team Project i na kolejnym menu, które się ukazało, kliknij pozycję Areas (Rysunek 13.). • Pojawi się okno jak poniżej, w którym można dokonać zmian.  • Skasuj niepotrzebne wydania (powiedzmy, iż zaplanowałeś tylko dwa wydania). • Skasuj niepotrzebne sprinty (załóżmy, że zaplanowałeś tylko 4 sprinty w wydaniu). • Sprawdź czy widać zmiany w ramach środowiska VS2010.
3. Edycja obszarów	• Edycje obszarów wykonujemy analogicznie jak edycję iteracji. • Wprowadź dodatkowy obszar.

Zmiana i wywoływanie zapytań dotyczących projektu w portalu TWA.

Zadanie	Tok postępowania
1. Otwarcie portal TWA	• Proszę wykonać analogiczne kroki jak w powyższym ćwiczeniu.
2. Wykonanie wbudowanego zapytania	• W głównym oknie TWA klikamy na link do zapytań (Queries). Pojawia się widok dwu typów, znanych już z VS2010 (My Queries i Team Queries). • Wejdźmy do Team Queries i wykonajmy dalej zapytanie Product Backlog. Lista będzie pusta – pamiętasz, że wszystkie nasze zapytania zostały przypisane do Sprintu? • Wyjdźmy z powrotem do Team Queries, a następnie wejdźmy do Current Queries i dalej wykonajmy Sprint Backlog. Czy są nasze prace oraz zadania do nich przypisane?
3. Wykonanie zapytania własnego	• Wracamy do okna zapytań i wchodzimy teraz do obszaru My Queries. • Jeśli wcześniej dodaliśmy własne zapytania, tak jak w module 4, to teraz będziemy je mieli na liście. Wykonajmy dowolne zapytanie InProgressMyTask. Czy na liście widać dwa zadania?
4. Definiowanie własnego zapytania	• W oknie w którym jesteśmy na niebieskiej belce znajduje się przycisk New, pojawi się menu jak poniżej.  • Z menu wybieramy pozycję Query. Pojawi się okno podobne do tego z rysunku 17 w module 4. • W menu wpisz kolejne zapytanie, na przykład wybieramy wszystkie zadania o statusie Done. Zapisze to pytanie, a następnie wykonaj. Czy otrzymałeś pustą listę? • Zmień teraz status zadania, które było w toku (In Progress) na zakończone (Done). • Wykonaj ponownie zapytanie. Czy pojawiło się to zadanie na liście?

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 6

Wersja 1

Wstęp do zapewniania jakości

Spis treści

Informacje o module

Opis modułu

W module tym znajdziesz podstawowe informacje dotyczące bardzo szerokiego zagadnienia jakim jest zapewnianie jakości wytwarzanego oprogramowania. W tak krótkim opracowaniu nie sposób jest, by poruszyć wszystkie aspekty. Stąd tez w module przeczytasz o zapewnieniu jakości kodu, procesach refaktoryzacji, czy wadze dokumentacji w tworzeniu oprogramowania. Zostanie zasygnalizowany problem standaryzacji nazewniczej i stylistycznej w pisaniu kodu oraz jak to się przekłada na jakość całego programu. Wspomnimy o testach jako jednej z metod weryfikacji poprawności działania programu.

Cel modułu

Celem modułu jest zapoznanie czytelnika z podstawowymi informacjami nt. zapewniania jakości w wytwarzanym oprogramowaniu przede wszystkim w od strony programistycznej.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- wiedział z jakimi zagadnieniami wiąże się zapewniania jakości,
- wiedział co to jest zapewnianie jakości kodu, programu czy też dokumentacji,
- potrafił określić czym jest standaryzacja kodowania,
- potrafił wskazać na źródła dotyczące standardów kodowania czy dokumentowania,
- potrafił przeprowadzić prostą refaktoryzację kodu,
- potrafił stworzyć prosty przypadek testowy,
- rozumiał role zapewniania jakości w całym cyklu życia oprogramowania.

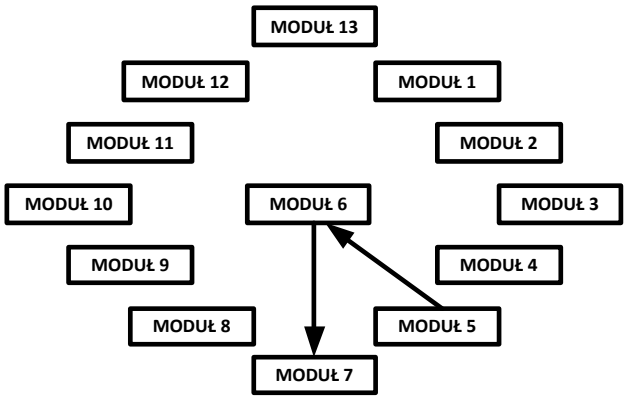
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawy obsługi środowiska Visual Studio 2010 oraz Team Foundation Studio 2010,
- znać podstawy pracy z szablonem Visual Studio Scrum 1.0,
- znać podstawy tworzenia aplikacji Windows (preferowany WPF) w środowisku Visual Studio 2010 przy użyciu języka C#,
- rozumieć czym jest cykl wytwarzania oprogramowania i jakie procesy w nim występują,
- rozumieć jakie role występują w metodyce Scrum oraz za co one odpowiadają.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w module



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Poprzednie Wasze doświadczenia pozwoliły Wam na zdobycie podstawowej wiedzy o rodzinie produktów VS 2010 i TFS 2010 oraz o tym, jak używając tych narzędzi i szablon Visual Studio Scrum 1.0, rozpocząć efektywną prace nad projektem w oparciu o metodykę Scrum.

Nadszedł czas, by przestać bawić się w sama zmianę statusów prac czy zadań, a rozpocząć pracę nad projektem i stworzyć jednak ten zamówiony kalkulator. Jednak nauczenie poprzednimi błędami, teraz już podchodzicie do tego ostrożniej i w związku z tym pojawiają wam się kolejne pytania?

Pierwsze z nich, to jak prawidłowo zainicjować drzewo kodu programu i jak umieścić je w repozytorium? Tu już znacie częściową odpowiedź, gdyż zainicjowaliście szkielet programu w module 3 i wstępnie znacie kolejność czynności jak i umiecie wykonywać podstawowe operacje na repozytorium kodu.

Pojawia się zatem kolejne pytanie, na co zwracać uwagę przy pracy w grupie programistów, jak rozdzielić pracę, czy stosować jakieś standardy kodowania (niby coś wspominano o tym na zajęciach, ale kiedy to było…)? Powoli zaczyna do Was docierać, że posiadacie podstawową wiedzę na temat tego jak zorganizować prace od strony zarządczej, ale brak Wam doświadczenia i wiedzy jak go zorganizować od strony inżynierskiej. Zatem spróbujmy nakreślić najważniejsze fakty i pokazać, jak do tego można wykorzystać poznane już narzędzia.

Podstawy teoretyczne

Zwykle w metodykach zarządzania projektami zapewnienie jakości jest elementem procesu zarządzania jakością. My w niniejszym module zajmiemy się tylko pewnymi elementami, związanymi głównie z zapewnieniem jakości, bez wnikania w szczegóły planowania jakości, czy audytów, wspomnimy o nich jedynie tam gdzie to niezbędne. Na uwagę zasługuje fakt, iż o pojęciu jakości, zapewnianiu jej, czy całym zarządzaniem, można mówić, co najmniej, z dwu punktów widzenia: jakości zarządzania projektem oraz jakości produktu jako takiego. Co więcej w przypadku produktu będą występowały konkretne miary i techniki jakości, w zależności od tego o jakim produkcie mówimy. Dla przykładu inne miary stosujemy w przypadku tworzenia oprogramowania, jako takiego, inne w przypadku tworzenia dla niego dokumentacji, a jeszcze inne dla obu tych rzeczy wspólnie – jako elementów składowych produktu inżynierii oprogramowania. Należy pamiętać, że niespełnienie wymagań jakościowych w którymkolwiek obszarze zwykle ma poważne negatywne następstwa dla projektu jak i osób z nim związanych. Konsekwencje dla projektu to zwykle: opóźnienie, utrata stabilności pracy. W przypadku osób, to może być to zwiększenie wysiłku w pracy, wzrost ilości nadgodzin i wiele innych.

Ogólnie o zapewnianiu jakości

Podstawowym pytaniem, jakie powinniśmy postawić na samym początku to, czym jest jakość? Według źródeł: *jakość to stopień, w jakim zbiór cech właściwych dla pewnej rzeczy spełnia stawiane tej rzeczy wymagania*. Dla przykładu, program komputerowy będzie miał wysoką jakość jeśli będzie miał mało błędów, nie będzie się zawieszał. Zarządzania jakością stanowi ważny element zarządzania projektami informatycznymi i należy patrzeć na jakość wielowymiarowo, wymieńmy najważniejsze z tych wymiarów:

- **Wymiar klienta** – spełnienie wymagań klienta jest jednym z najważniejszych elementów zapewnienia jakości, przy czym należy pamiętać, że wymagania klienta nie mogą stanąć w sprzeczności z ich zdatnością użytkową (ang. fitness for use), czyli produkt musi spełniać rzeczywiste potrzeby.

- **Wymiar zespołu** – spełnienie wymagań klienta nie może odbyć się za wszelką cenę, nadmierne obciążenie zespołu (w postaci np. nadgodzin), częste zmiany w wymaganiach mogą wpłynąć na ogólną frustrację.

Zauważmy, że oba powyższe wymiary są ze sobą ściśle powiązane, dla przykładu frustracja w zespole może wpłynąć na zwiększenie ilości błędów i ostatecznie spadku jakości (wymiar Klienta), brak konsekwencji co do wymagań u Klienta spowoduje spadek motywacji w zespole itd.

Zawsze, gdy myślimy o zapewnieniu jakości należy mieć na uwadze kilka zasad:

- **Odpowiedzialność kierownictwa** – zasada obowiązująca właściwie w każdym obszarze zarządzania projektem, co prawda każdy członek zespołu jest istotnym ogniwem łańcucha jakości, ale ostatecznie to kierownictwo ponosi odpowiedzialność.
- **Zapobieganie zamiast kontroli** – obecne podejście do zapewnienia jakości podkreśla, że jakość osiąga się przede wszystkim poprzez planowanie i wdrażanie, a nie poprzez kontrolę. Wynika to z faktu, że koszt związany z zapobieganiem powstawania błędów jest zwykle znacznie niższy niż koszty ich usuwania, choćby tych wykrytych w trakcie kontroli. Koszt usunięcia błędów wykrytych w trakcie użytkowania jest już wielokrotnie droższy od zapobiegania.
- **Ciągłe doskonalenie** – jest to proces ilustrowany przez tzw. cykl Deminga (koło Deminga lub cykl PDCA – ang. Plan-Do-Check-Act lub cykl PDSA – ang. Plan-Do-Study-Act) sprowadzający się do ciągłego udoskonalania procesu. Cykl ten zaproponował W. E. Deming, amerykański statystyk. W oryginalnej cykle ten został opisany następującymi krokami (wersja uproszczona):
  - **Plan** (zaplanuj) – planuj każdą zmianę z wyprzedzeniem, analizuj obecną sytuację oraz potencjalne skutki wprowadzenia zmiany zanim ją podejmiesz.
  - **Do** (wykonaj) – przeprowadź pilotażowe wdrożenie w kontrolowanych warunkach.
  - **Study** (zbadaj) – przeprowadź gruntowną analizę wniosków i wyciągnij wnioski.
  - **Act** (zastosuj) – podejmij właściwe działania.

  Należy zaznaczyć, że każde działania mające na celu doskonalenie procesu zarządzania jakości na poziomie organizacji mogę wpływać na jakość tak samego projektu jak i wytwarzanych produktów.

Zapewnienie jakości, w projekcie jako takim, musi odbywać się na każdym jego obszarze: czas, zakres, budżet jak i sama jakość. Ostatnie z wymienionych obszarów może wydawać się trochę nie na miejscu, jednak należy pamiętać, że zapewnienie jakości jako proces można również kontrolować – zatem zapewniać jego jakość. W każdym razie my skupimy się i będziemy mówić o tych pierwszych trzech obszarach. Aby kontrola w tych obszarach była możliwa, należy w projekcie posiadać, dla każdego z tych obszarów, punkty odniesienia, dlatego ważne są m.in. elementy:

- **Zakres prac** – w przypadku metodyki Scrum będzie to Wykaz prac produktu jako całość prac, oraz Wykaz prac sprintu, który będzie najważniejszy w danej iteracji. Będzie to podstawa do określenia, czy spełnimy wymagania klienta.
- **Kryteria akceptacji produktu** - ważne jest by produkt, jako całość, jak i każda praca, którą mamy wykonać posiadała jasno sprecyzowane kryterium akceptacji, w przeciwnym razie odbiór jej może być utrudniony, a czasem wręcz nie możliwy. Należy również pamiętać, że kryteria akceptacji dla produktu mogą mieć również charakter niefunkcjonalny np. „raport miesięczny” powinien się generować nie dłużej niż 15 sekund.
- **Wykaz zadań** – wykaz zadań realizowanych w trakcie danego sprintu zawiera ważne informacje techniczne nt. każdego z zadań i może stanowić punkt odniesienia do dalszych działań (o czym dalej).
- **Rejestr interesariuszy** – określa osoby i grupy osób, które spodziewają się określonych korzyści po produkcie. W naszym przypadku ograniczymy się tylko do Właściciela produktu.

- **Budżet bazowy** – stanowi on punkt odniesienia do kontroli budżetu w projekcie.
- **Harmonogram bazowy** – stanowi on punkt odniesienia do kontroli czasu w projekcie.
- **Spis ryzyk** – bardzo ważny element, z którego mogą wynikać zagrożenia dla jakości w projekcie (więcej na ten temat w module ???).

Powyższe obszary stanowią punkt odniesienia w zakresie zapewnienia jakości, tak w wymiarze organizacyjnym, jak i funkcjonalnym. Natomiast zupełnie osobną sprawą jest zapewnienie jakości w szeroko rozumianym wymiarze technicznym czy inżynieryjnym. I w przypadku projektów informatycznych będziemy tu mówić o różnego rodzajach miarach dotyczących oprogramowania jak i jego dokumentacji oraz technikach zapewniania jakości tychże. Można tu wymienić kolejne obszary:

- **Zapewnienie jakości kodu** – tu można wymienić takie działania jak standardy kodowania i dokumentowania kodu, czy zarządzanie zmianami i refaktoryzacja.
- **Zapewnienie jakości programu** – tu warto wspomnieć o testach, a właściwie całej rodzinie testów, czy podejściu TDD (Test Driven Development).
- **Zapewnienie jakości dokumentacji** – w tym miejscu zwróćmy uwagę, że mówimy o co najmniej dwu rodzajach dokumentacji: projektowej i użytkowej. Pierwszy z nich obejmuje dokumentacją projektu i kodu, a drugi to dokumentacji dla użytkownika (np. podręcznik) jak i dla serwisu informatycznego po stronie użytkownika (instrukcje serwisowe, konserwacyjne).

W dalszych częściach modułu skupimy się tylko na tych dwu pierwszych obszarach, traktując iż trzeci z nich intuicyjnie jest łatwy do wyobrażenia.

Zanim jednak do tego przejdziemy należy jeszcze wspomnieć o metodach i technikach zapewniania jakości, tu zwykle mówi się o dwu podstawowych:

- **Audyt jakości** – audyt to zaplanowany, metodyczny i niezależny przegląd wybranego obszaru projektu pod kontem jego zgodności z zasadami, normami, procesami i procedurami określonymi w projekcie. Celem audytu jest wykrycie wszystkich niezgodności i zasugerowania działań naprawczych.
- **Analiza procesów** – sprowadza się do przeglądu istniejących procesów zapewniania jakości i wskazania stwierdzonych problemów.

Pamiętając o zasadzie „zapobieganie zamiast kontroli” nie wolno ignorować kontroli, gdyż jest to ważny element wczesnego wykrywania nieprawidłowości. W przeciwnym razie te nieprawidłowości może wykryć klient, lub mogą objawić się w niespodziewanych sytuacjach, a wtedy, zwykle, będzie to miało, dla zespołu i innych zainteresowanych stron, znacznie większe konsekwencje. Przypomnijmy jeszcze, iż każde działanie w projekcie, w tym również zapewnianie jakości, ma swój koszt, zatem planując jakość i procesy z nią związane musimy to wszystko mieć na uwadze.

Ostatecznie już na sam koniec warto zastanowić się jak zarządzanie jakością odnosi się do Scrum i odwrotnie. Otóż sama metodyka Scrum nie posiada w sobie zawartych konkretnych technik czy wskazań jak należy zarządzać jakością, traktuje ona, że są do tego osobne dobre praktyki, niemniej jednak Scrum poprzez swoją częstą interakcję z Właścicielem Produktu oraz iteracyjny model budowy wpisuje się dobrze w ogólny schemat kontroli jakości, gdyż jest wiele miejsc w którym zadajemy pytania „Co się dzieje?”, „Czy nie ma problemów?”, wystarczy choćby wspomnieć o codziennym scrumie, w którym robimy mini przegląd projektu.

Zapewnienie jakości kodu

Właściwe przechowywanie kodu

Już wcześniej pisaliśmy o tym, że jednym z najważniejszych narzędzi wspierający pracę nad oprogramowaniem są systemy kontroli wersji. Przypomnimy jednak najważniejsze fakty z nimi związane, gdyż nie ma mowy o zapewnianiu jakości kodu bez posługiwania się tymi rozwiązaniami.

Zatem zacznijmy od nadrzędnej reguły od której nie ma wyjątku! **Zawsze i w każdej sytuacji tworząc aplikację produkcyjną umieszczamy jej kod w repozytorium, które wspiera zarządzanie kodem i jego wersjami.** Można zadać pytanie co mamy na myśli mówiąc „w każdej sytuacji", otóż tak naprawdę nie ma znaczenia, czy tworzymy sami tę aplikację, czy w grupie, w obu sytuacjach kod ma być w repozytorium, bez wyjątku. Wynika to z faktu, iż nawet, gdy tworzymy aplikacje w pojedynkę, dobrze jest z jednej strony posiadać kopię plików lokalnych (trochę spełnia to funkcję kopii bezpieczeństwa, o ile serwer i środowisko programistyczne to dwa różne komputery), a co ważniejsze, dzięki temu sami będziemy mogli cofać się w historii zmian aplikacji. Oczywiście, gdy nie tworzymy aplikacji dla potrzeb produkcyjnych, a jedynie coś testujemy to nie ma takiej konieczności, stąd sformułowanie o „aplikacji produkcyjnej".

Jako drugą regułę należałoby wymienić kilka powiązanych ze sobą rad dotyczących tego, kiedy i co zatwierdzać jako zmiany do repozytorium. Pamiętajmy jednocześnie, że pojedyncza zmiana (tzw. check-in) może, i zwykle tak jest, dotyczyć kilku plików.

1. Pojedyncza zmiana, którą zatwierdzamy do repozytorium powinna być kompletna, tzn. nie wolno umieszczać w repozytorium np. kodu który nie przechodzi poprawnie kompilacji, lub kodu, który implementuje tylko kawałek funkcjonalności. W pierwszym przypadku spowoduje to niemożność budowania automatycznie na serwerze aplikacji i przeprowadzenia na niej testów, w drugi możemy zapomnieć, że ta funkcjonalność jest nie kompletna.
2. Ideałem by było, by pojedyncze zatwierdzenie naszych zmian w kodzie odpowiadało dokładnie implementacji jednego zadania (takiego, które poznaliśmy przy omawianiu metodyki Scrum). Jeżeli jest to nie możliwe, to lepiej jest podzielić zadanie na dwa mniejsze.
3. Jeśli implementacja zadania się przedłuża (np. gdy zaszły nieprzewidziane okoliczności), to nie powinniśmy czekać dłużej niż jeden dzień z rejestracją zmian. Ale wtedy umieśćmy ją na naszej półce (shelve), a nie w głównym repozytorium, inaczej możemy spowodować problem z punktu 1.
4. Kod, który umieszczamy powinien być dobrze udokumentowany. Dokumentacja powinna być na tyle jasna, by inna osoba przy jej wykorzystaniu mogła zrozumieć do czego ten fragment programu służy.
5. Kod, który umieszczamy powinien być wstępnie przez nas przetestowany. Nie wolno dopuszczać do sytuacji, gdy funkcja, którą napisaliśmy powoduje np. błąd przy dowolnym wywołaniu.

Standard kodowania

Zapewnienie jakości kodu osiąga się m.in. poprzez wprowadzenie standardu kodowania, który obejmuje wszystkie aspekty budowy kodu od nazewnictwa poprzez komentarze, a nawet formatowanie – można powiedzieć, że jest to niejako ortografia i stylistyka języka programowania. Każdy programista, który pracuje w ramach pewnego projektu powinien dostosować się do standardu ustalonego w ramach tego projektu lub organizacji, której jest częścią. Jeśli projektu dopiero się rozpoczyna, to powinien zostać ustalony standard kodowania, by wszyscy deweloperzy mieli punkt odniesienia – nie ma sensu wprowadzać standardu kodowania na koniec projektu. Jeśli natomiast projekt zawiera inne oprogramowanie z istniejącym już kodem źródłowym, to w miarę możliwości w trakcie budowy systemu, powinno ujednolicać się standard – można to robić w ramach procesu refaktoryzacji, o czym dalej. Wprowadzenie standardu kodowania wynika z naturalnego dążenia, by kod źródłowy odzwierciedlał jakby jeden styl pisania, prowadzi to do następujących pozytywnych rezultatów:

- Standard kodowania zapewnia łatwiejszą pielęgnację kodu w dłuższej perspektywie, za co za tym idzie obniżenie kosztów utrzymania. Dzieje się tak między innymi, dlatego, że czytelność kodu źródłowego ma bezpośredni wpływ na to jak dobrze rozumie go programista czytając ten kod po pewnym czasie. W tak napisanym oprogramowaniu, posiadając dobrą

dokumentacją, dużo łatwiej jest rozbudowywać program o nowe funkcje czy modyfikować już istniejące.
- Wprowadzenie standardu w zakresie technik kodowania i dobrych praktyk programistycznych, pozwala stworzyć kod wysokiej jakości, który to ostatecznie odgrywa kluczową rolę w jakość i wydajność oprogramowania.
- Standard kodowania wpływa na skrócenie czasu potrzebnego na wdrożenie nowego programisty do pracy, co z kolei wpływa na zmniejszenie kosztów projektu. Oczywiście oszczędność wynika z faktu, iż można wymagać znajomość pewnych standardów w trakcie zatrudniania.

Jako przykład standardu kodowania można podać zalecenia wg. dokumentacja MSDN Microsoft (http://msdn.microsoft.com/en-us/library/aa291596(VS.71).aspx).

Jedną z najprostszych i zarazem najskuteczniejszych metod utrzymania kodu w zgodzie z ustalonym standardem są okresowe przeglądu kodu. Wykonywana są one najczęściej w ramach zespołu, wzajemnie przez samych programistów. Takie przeglądy mają na celu nie tylko utrzymanie zgodności ze standardem, ale przede wszystkim dają możliwość wykrycia na wczesnym etapie defektów programistycznych.

Refaktoryzacja

Komplementarnym działaniem do wprowadzenia standardów kodowania jest refaktoryzacja kodu. Refaktoryzacja jest to proces wprowadzania zmian w kodzie źródłowym lub w architekturze całego systemu, w wyniku którego nie zmienia się funkcjonalność, zatem celem refaktoryzacji jest utrzymywanie odpowiedniej, wysokiej jakości kodu. My ograniczymy się w tym miejscu jedynie do zagadnień refaktoryzacji kodu i nie będziemy mówili o architekturze. W ramach tego procesu podejmowane są następujące przykładowe działania:

- modyfikowanie elementów systemu (zmienne, funkcje, itd.) w celu wpasowania ich w przyjęte standardy i wzorce,
- identyfikacja wspólnych fragmentów kodu i podejmowanie decyzji o wydzieleniu ich do funkcji a potem bibliotek,
- analiza i identyfikacja czy nie pojawiają się nowe standardy i wzorce.

Dzięki refaktoryzacji w systemie ogranicza się redundancję, porządkuje kod, wprowadza standardy, uzupełnia dokumentację. I znów należy pamiętać, że refaktoryzacja jest kosztowna, i choć jest istotnym elementem zapewniania jakości kodu należy zawsze przemyśleć, czy wykonywać ją, a jeśli tak to w jakim zakresie. Z pewnością w dużych i złożonych projektach koszt dobrze prowadzonej refaktoryzacji (prowadzonej a nie przeprowadzonej, bo refaktoryzacja jest procesem ciągłym!), powinien zostać zrekompensowany dużo niższym kosztem wprowadzania późniejszych zmian. Refaktoryzację w stosunku do standardów kodowania należy postrzegać jako działanie wtórne, zatem najpierw „zapobiegamy” poprzez uczenie wszystkich właściwych sposobów pisania.

Jako przykład refaktoryzacji można podać następujące fragmenty kodu.

Kod przed zmianą:

```
int TaskX;
int TASKy;
int sum = TaskX + TASKy;
```

W powyższym fragmencie widać wyraźnie brak konsekwencji nazewniczej, można wskazać następująca propozycję zmiany:

```
int taskX;
int taskY;
```

```
int sum = taskX + taskY;
```

Następny przykład prezentuje brak porządku w kodzie instrukcji `if`:

```
if (x <12&& y>X) { i++;
  cout << x+y<<endl;
cout << i; }
```

po zmianie:

```
if (x < 12 && y > X)
{
  i++;
  cout << x+y << endl;
  cout << i;
}
```

Refaktoryzacja ma jeszcze jeden wymiar, otóż pomaga pozbyć się na przykład nadmiarowych fragmentów programu, nie tylko poprzez wyrzucenie ich do biblioteki (jeden przypadek), ale również wyrzucenie ich w ogóle (drugi przypadek). Być może, na początku może dziwić fakt, że może pojawić się kod programu, który jest zbędny. Praktyka pokazuje jednak, że zawsze tworząc oprogramowania, możemy pozostawić fragmenty, które służyły do testów i zostały za komentowane lub nie, gdzieś w programie. Potem o nich się zapomina a one zaśmiecają kod, powodując z jednej strony zmniejszenie czytelności, a z drugiej wprowadzając frustrację w późniejszym czasie w trakcie pielęgnacji. Zwłaszcza ten drugi aspekt jest ważny, gdyż po jakimś czasie sami zaglądamy i tracimy czas zastanawiając się po co ten fragment był, co gorsze, czy jest nadal potrzebny.

Zapewnienie jakości programu

Prawdą jest to, co było powiedziane w poprzednim punkcie, iż znaczna część jakości ostatecznego produktu w projekcie informatycznym zależy od jakości kodu programu, niemniej jednak nie można naiwnie sądzić, iż to wystarczy. Z faktu, iż zespół będzie spełniał jedynie standardy kodowania, nie wynika, że program będzie działał poprawnie. Stąd też potrzebne są dodatkowe czynności pozwalające zapewnić jakość kodu w zakresie jego działania, czy pomagające unikać przykładowych następujących błędów:

- Błędy implementacyjne – wynikają ze złej implementacji danej funkcji czy bardziej złożonego fragmentu kodu (np. błędy obliczeń), wykrycie ich jest stosunkowo łatwe, a usunięcie najczęściej szybkie.
- Błędy przepełnienia bufora (pamięci) – w zależności od skutków jakie przynoszą, bywają łatwe lub bardzo trudne do zlokalizowania.
- Błędy złej interpretacji danych – wynikają najczęściej ze złych ustaleń z klientem lub braku zrozumienia tychże, są dość łatwe do wykrycia, o ile przepływ danych nie jest bardzo złożony i dość proste do usunięcia w niedużych systemach. Jednak im większy system tym gorzej.

Jak to zauważyliśmy powyżej, część tych błędów jest łatwo znaleźć, inne trudniej. Niewątpliwie jednym z najlepszych sposób unikania błędów jest planowanie testów już w trakcie tworzenia oprogramowania. Różne metodyki tworzenie oprogramowania w różny sposób do tego podchodzą, są takie jak np. TDD (Test Driven Development) czy XP (eXtreme Programming), które wręcz wymagają stworzenia testów przed rozpoczęciem kodowania danej funkcjonalności, a są takie, które mówią o testowaniu na końcu projektu (np. model kaskadowy). Ponieważ Scrum nie określa żadnych surowych zasad związanych z testowaniem, to my w tym module wspomnimy jedynie o

tym procesie, oraz zaprezentujemy w części ćwiczeniowej sposoby wykonania testów, jednak bez narzucania samego cyklu testowania (więcej o testach w module ???).

Podsumowanie

W tym rozdziale przedstawione zostały przedstawione ogólne zasady dotyczące zapewniania jakości oraz niektóre elementy związane z szerszym pojęciem – zarządzanie jakością. Następnie przybliżone zostały zagadnienia specyficzne dla inżynierskiej części wytwarzania oprogramowania, a zatem zapewniania jakości kodu oraz programu. Zatem wyjaśniliśmy jak dbać o kod, co to są standardy kodowania, czym jest refaktoryzacja oraz w jaki sposób można zapewnić badania poprawności programu poprzez testy. Wszystkie omówione zagadnienia należy traktować jedynie jako zarys bardzo szerokiej problematyki, której obecnie poświęca się bardzo dużo miejsca zarówno w praktyce w przemyśle jak i w nauce – w trakcie badań teoretycznych i empirycznych. Jest jasne, iż zapewnienie jakości będzie jednym z najważniejszych obszarów badań i rozwoju w ramach inżynierii oprogramowania w najbliższych latach.

Przykładowe rozwiązanie

Wróćmy ponownie do przykładu z kalkulatorem. Jak pamiętamy w poprzednim module zakończyliśmy obie prace zawarte w Wykazie prac sprintu, oraz oba zadania, która były do nich przypisane. Jednak zrobiliśmy to niejako „na sucho”, bez kawałka kodu programu. Spróbujmy teraz stworzyć szkielet aplikacji i powiązać czynności jakie będziemy wykonywać w zakresie technicznym (inżynierskim) z czynnościami zarządczymi wynikającymi z metodyki Scrum.

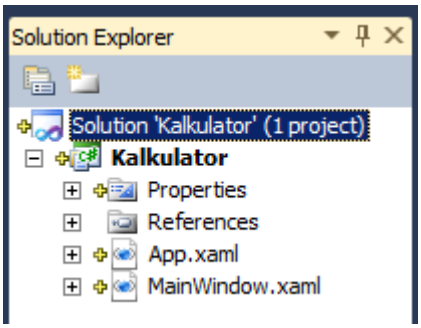
Wróćmy w pracach do statusu Commited, a w zadaniach im przypisanych do statusu In Progress.

Generujemy szkielet aplikacji

Rozpoczynamy pracę od uruchomienia środowiska Visual Studio 2010 oraz otwarcia projektu Kalk1. Następnie dodamy do tego projektu początkowy kod aplikacji i stopniowo go będziemy rozbudowywać. Przypomnijmy sobie informacje zawarte w module 3 począwszy od opisu Rysunku 15. Wykonujemy kolejno kroki:

1. Z menu File wybieramy podmenu New, a następnie klikamy Project… (skrót klawiszowy Ctrl+Shift+N).
2. W oknie dialogowym, które się pojawiło kolejno (New Project):
   a) W lewej kolumnie (Installed Templates) klikamy na węzeł Visual C#.
   b) W środkowej części w okienku wyboru framework wybieramy .NET Framework 4, a poniżej WPF Application.
   c) W dolnej części wpisujemy nazwę aplikacji – Kalkulator. Wybieramy preferowaną lokalizację plików, a nazwę w oknie Solution pozostawiamy taką jaką sugeruje środowiska (Kalkulator).
   d) Jeśli nie jest zaznaczone, to wybieramy opcję Create directory for solution i koniecznie Add to source control.
   e) Naciskamy przycisk OK.
3. W tym momencie środowisko VS 2010 tworzy wszystkie niezbędne pliki projektu.
4. Następnie pojawi się nowe okno dialogowe (Add Solution Kalkulator to Source Control), w którym pozostawiamy wszystkie ustawienia bez zmian i naciskamy przycisk OK.
5. Jeśli wszystkie czynności wykonaliśmy poprawnie, to w oknie Solution Explorer, które powinno znajdować się w tym samym miejscu co Team Explorer, najwyżej należy kliknąć na zakładkę o odpowiednim tytule pojawi się drzewo aplikacji z wyglądzie jak na poniższym rysunku. Zwróćmy uwagę na tym rysunku, iż przy większości węzłów znajduje się ikona małego znaku plus. Oznacza to, że ten węzeł i pliki w nim zawarte są świeżo dodane do projektu i oczekują, by zatwierdzić ich dodanie do drzewa repozytorium. Uwaga! Pliki te jeszcze nie znajdują się w repozytorium. Potwierdzić ten fakt możemy poprzez widok w oknie

zmian oczekujących (Pending Changes), które możemy otworzyć klikając kolejno menu View, Other Windows, Pending Changes. W oknie tym widać wyraźnie całą listę plików ze statusem Add – oczekujących na wykonanie operacji Check In. Może budzić zdziwienie, iż węzeł References nie ma znaczka plus, ale nie jest to węzeł zawierający pliki ani inne elementy, tylko wirtualne odwołania, jakby dynamiczną wiedze o projekcie. Zatem nie podlega wersjonowanie, jego zawartość jest określana na bieżąco przez środowisko.



Rysunek 2. Widok okna Solution Explorer tuż po utworzeniu szkieletu aplikacji Kalkulator.

6. W oknie Solution Explorer najedźmy kursorem myszy na nazwę solucji (Solution 'Kalkulator') i naciśnijmy prawy klawisz myszy. W menu kontekstowym wybierzmy pozycję Check In i naciśnijmy lewy przycisk myszy. Pojawi się okno o tytule Chech In – Source Files – Workspace: NAZWA_NASZEGO_SERWERA. W oknie tym widzimy długa listę plików, które zostały utworzone automatycznie przez środowisko i powinno być aktywne okno komentarza (Comment). Jeśli nie jest to pierwsza z ikon na górnym pasku włącza i wyłącza to pole. Jako komentarz wpiszmy „Stan początkowy", pamiętajmy, że komentowanie zmian jest bardzo ważne, pisaliśmy już o tym w module 3. Naciskamy przycisk Check In. Jeśli wszystko przebiegło bez błędów, to w oknie Solution Explorer znaki plusa zostaną zamienione na kłódki, a okno Pending Changes nie będzie zawierało żadnych plików oczekujących.
7. Od tego momentu zadbaliśmy w podstawowy sposób o nasz kod, czyli spełniliśmy warunek właściwego przechowywania kodu.
8. Kończymy proces generowania szkieletu aplikacji.

Wykonanie pierwszej wersji interfejsu

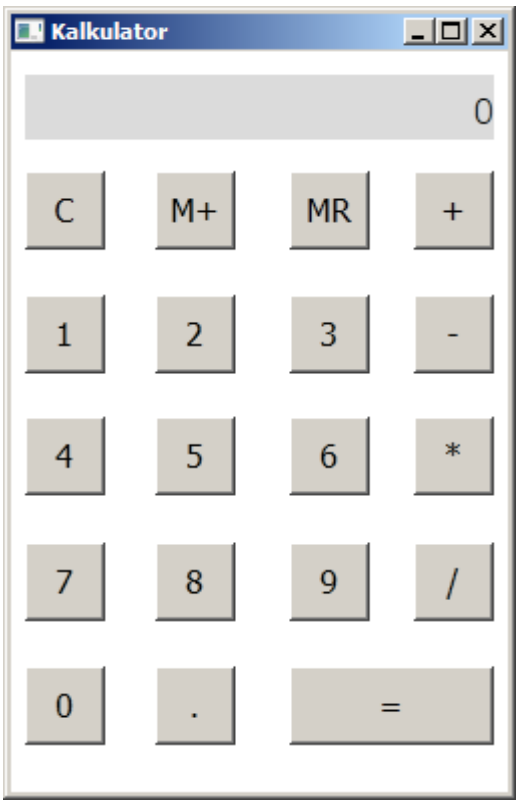
Zanim dokonamy jakichkolwiek zmian, to wykonajmy pierwszą kompilację, by upewnić się, iż wszystko jest w porządku. W tym celu wykonujemy budowanie solucji, możemy to zrobić np., poprzez wybranie z menu Build pozycji Build Solution, lub naciśnięcie przycisku funkcyjnego F6. Jeśli budowanie przebiegło prawidłowo, to nie pokaże się żadne okno z błędami. Jeśli chcemy upewnić się, że na pewno wszystko jest w porządku, to z menu View wybieramy pozycję Output, na końcu komunikatu w oknie powinna znaleźć się informacja jak poniżej:

```
Build: 1 succeeded or up-to-date, 0 failed, 0 skipped
```

Teraz uruchomimy zbudowaną aplikacje, z menu Debug wybierzmy pozycję Start Without Debugging, jeśli pojawiło się puste okno z tytułem MainWindow, to znów utwierdziliśmy się, iż aplikacja została prawidłowo utworzona. Zamknijmy aplikację i dokonajmy pierwszych zmian (ponieważ zakładamy znajomość podstawowej obsługi i budowania aplikacji w środowisku Visual Studio 2010, to będziemy podawali tylko główne kroki).

1. Ponieważ zaczynamy prace zmierzające do realizacji prac z wykazu prac sprintu, to pierwszym krokiem musi być zmiana statusu zadania „Wykonanie głównego okna aplikacji" z obecnego (To Do) na In Progress. W tym celu otwórzmy Sprint Backlog, odnajdźmy zadanie i zmieńmy status oraz zapiszmy zmiany.
2. Rozpoczynamy prace programistyczne.

3. Zmieńmy nazwę okna głównego z MainWindow na Kalkulator – w tym celu zmieńmy właściwość Title w pliku XAML. Zauważmy, że plik ten zostanie automatycznie pobrany z repozytorium (Check Out) do edycji (zmieni się np. ikona w drzewie Solution Explorer).
4. Dodajmy teraz z palety komponentów (Toolbox) pole tekstowe (TextBlock) do głównego okna aplikacji i umieśćmy je na górze okna aplikacji, tak jak to wygląda w zwykłym kalkulatorze.
5. Otwórzmy okno właściwości, np. poprzez wejście w menu View, a następnie kliniecie Properties Window. Kliknijmy komponent TextBlock, bo najpierw będziemy zmieniać jego właściwości.
   a) Ustawmy właściwość Text na wartość 0 (zero).
   b) Otwórzmy węzeł Brushes i zmieńmy właściwość Background na jakiś z jasnych odcienie szarości.
   c) Rozwińmy węzeł Text i w nim: zmieńmy rodzaj fontu na Courier New, wielkości fontu na 28 i wyrównanie do prawej.
   d) Możemy zweryfikować nasze zmiany poprzez uruchomienie aplikacji (np. Ctrl+F5).
6. Dodajmy z palety komponentów pierwszy przycisk, który będzie reprezentował np. kasowanie zawartości, oraz pozostałe (cyfry, kropkę, cztery podstawowe działania oraz znak =). Rozmieśćmy je w taki sposób, by wyglądały estetycznie oraz przypominały klawiaturę kalkulatora.
7. Po dodaniu przycisków, zbudujmy solucję, a następnie wykonajmy naszą aplikację, jeśli wszystko się powiodło, to powinna wyglądać podobnie jak na rysunku 3.

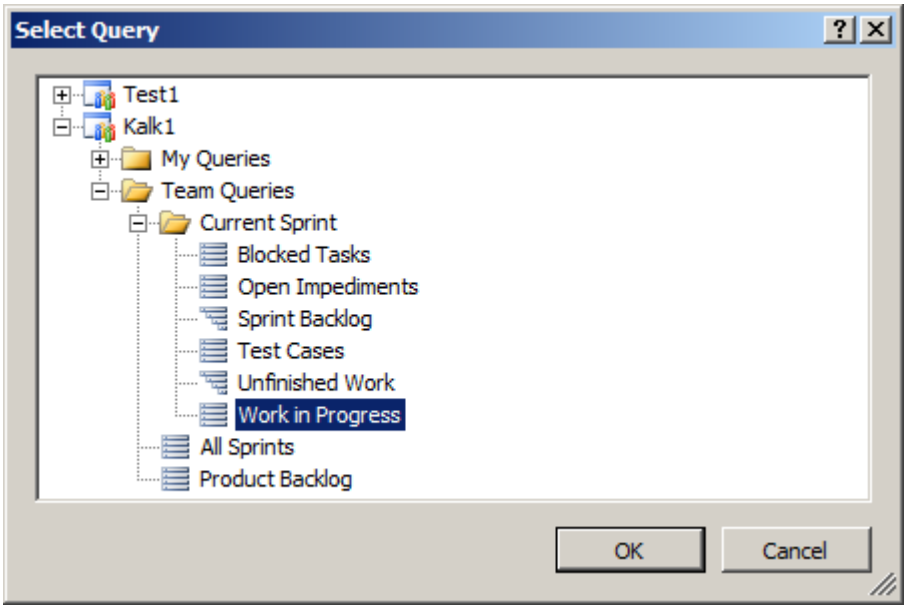


Rysunek 3. Widok wstępnej wersji wyglądu aplikacji Kalkulatora.

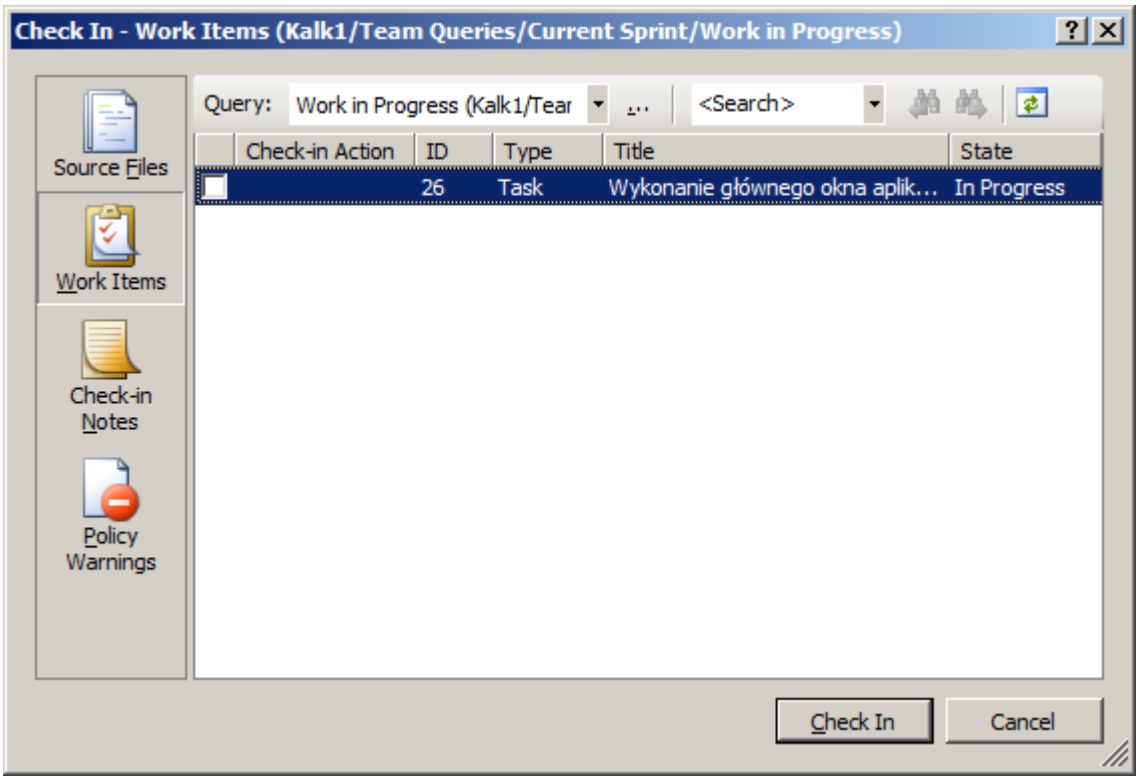
8. Załóżmy teraz, że programista wykonujący to zadanie uznał, iż jest to wszystko co do niego należy, to znaczy wykonał tylko co widać na rysunku.
9. Programista piszący wygląd graficzny postanawia zapisać zmiany do repozytorium. W tym celu klika prawym klawiszem na nazwie solucji i z menu wybiera pozycję Check In. W konsekwencji pojawia się znane już z wcześniejszego działania okno zawierające dwa pliki na

liście (tylko w nich dokonywane były zmiany). Jednak tym razem nie wpisujemy komentarza i od razu zapisujemy zmiany do repozytorium.

a) Wpierw w lewej kolumnie naciskamy ikonę Work Items, pojawi się okno, które w zależności od tego co wybierzemy w liście Query, to wyświetli nam takie elementy typu Work Items.
b) Z listy wybierzmy pozycję o nazwie Work In Progress (Kalk1/Team Queries), jeśli jej nie ma to obok okna Query naciśnijmy przycisk … (trzy kropki), pojawi się okno jak na rysunku 4. W oknie tym wyszykujemy zapytanie, które nas interesuje i naciskamy OK.
c) Powróciliśmy do okna zatwierdzania zmian do repozytorium, gdzie tym razem widać, po wybraniu naszego zapytania, listę zadań z bieżącego sprintu, które mają status In Progress. W naszym przypadku jest to tylko jedno zadanie, ale oczywiście mogłoby być ich więcej (Rysunek 5.).
d) Jeśli teraz zaznaczymy w okienku to zadania (check box), to w kolumnie Check-In Action pojawi się lista akcji, pozostawiamy w niej wartość Associate. Może pojawić się naturalne pytanie, po co w tym miejscu umożliwiono zaznaczanie tych zadań. Otóż, zauważmy, że prowadzenie prac programistycznych, ma na celu wykonanie określonego zadania, które z kolei jest elementem pracy pochodzącej początkowo z Wykazu prac produktu. Stąd też powiązanie wykonanie prac programistycznych z konkretnym zadaniem ułatwia późniejsze wyszukiwanie i analizę wytwarzania całej aplikacji. Co więcej, stanowi pomost pomiędzy światem inżynierskim a zarządczym, przez co wpływa na utrzymanie spójności całego projektu i w efekcie wspomaga zapewnienie jakości.

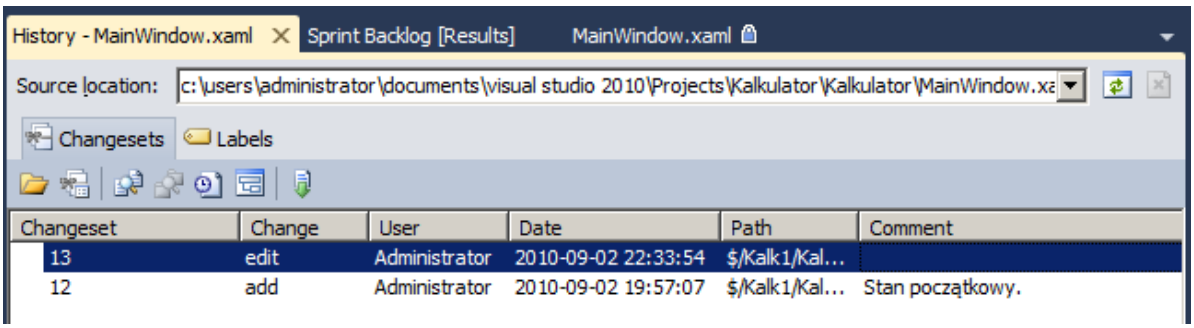


Rysunek 4. Okno pozwalające wybrać zapytanie na podstawie którego zostaną wybrane elementy typu Work Items.



Rysunek 5. Okno zatwierdzania zmian do repozytorium zawierające listę zadań w toku z bieżącego sprintu.

10. Zaznaczmy w oknie z rysunku 5 nasze zadanie, pozostawmy, jak pisaliśmy w kolumnie Check-In Action wartość Associate i naciśnijmy Check In. Czego zapomnieliśmy? Brawo! Zapomnieliśmy o komentarzu, na drugi raz pamiętajmy.
11. Kod został dodany do repozytorium, teraz możemy się przyjrzeć temu co zostało zrobione po stronie repozytorium i jakie informację stamtąd możemy uzyskać.
12. W oknie Solution Explorer kliknijmy prawym klawiszem na pliku MainWindow.xaml i z menu, które się pojawi wybierzmy History (Rysunek 6.). W oknie, które się pokazało, widać dwie wersje zapamiętane w repozytorium, pierwsza jako dodanie pliku (wartość add w kolumnie Change), a druga jako edycja (wartość edit w tej kolumnie). Oczywiście nie ma komentarza przy najnowszej wersji.



Rysunek 6. Historia kolejnych wersji pliku MainWindow.xaml zapamiętana na serwerze.

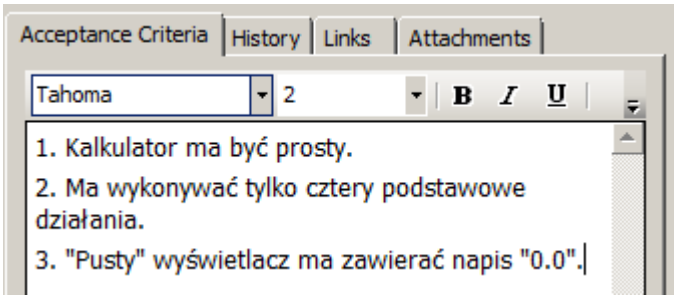
13. Pozostając w oknie historii najedźmy kursorem na najnowszy wiersz i kliknijmy prawy klawisz myszy. Z menu, które się pojawiło wybierzmy pozycję Changeset Details, pojawi się podobne okno jak na rysunku 5.
14. Możemy teraz w zakładce Work Items podejrzeć z którym zadaniem zmiana ta została skojarzona.

15. Wykonajmy tę samą czynność (Changeset Details) dla poprzedniej zmiany (dodania do repozytorium) i spójrzmy znów w Work Items, jak widać nie ma żadnego elementu skojarzonego z tą czynnością.
16. Na sam koniec wejdźmy ponownie do wykazu prac sprintu, odszukajmy nasze zadanie i sprawdźmy jego status, jak widać nadal jest to zadanie w toku, także samo zapisanie zmian do repozytorium wraz z podłączeniem ich do danego zadania, nie warunkuje jeszcze zmiany statusu tego zadania.

Błąd w wykonaniu zadania

Osoba, które weryfikowała wykonane zadanie zauważyła, że na wyświetlaczu pojawia się napis „0”, a powinien „0.0”, gdyż tego oczekiwał Klient. Jednak okazało się, iż zupełnie pominęliśmy kryteria akceptacji dla pracy „Szkielet kalkulatora”, która zawiera nasze zadania – cóż nie ma się co dziwić, że programista wykonał szkielet kalkulatora „na oko”, przecież nie miał pojęcia czego Klient oczekuje (prawdę powiedziawszy, to jest wyjątkowo naciągany przykład, gdyż każdy programista powinien od razu poprosić o kryteria akceptacji lub bardziej precyzyjny opis zadania). Niestety popełniliśmy duży błąd z punktu widzenia zapewniania jakości, nie można mówić o wykonaniu jakiejkolwiek

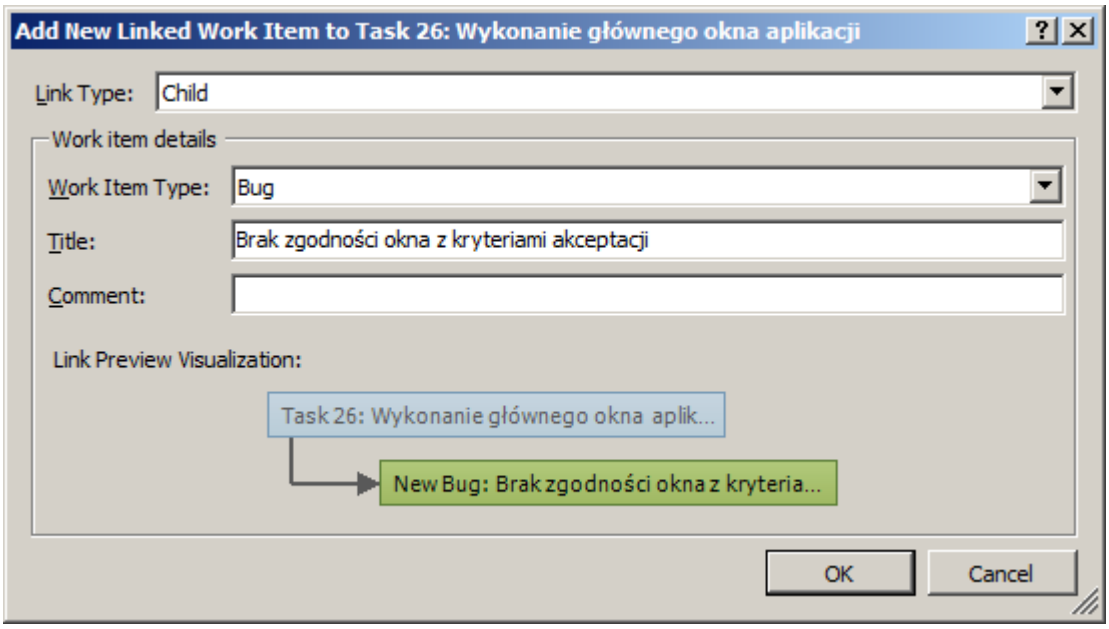
W takim razie, zanim zgłosimy zadanie do poprawy, musimy uzupełnić kryteria akceptacji, wchodzimy do wykazu prac sprintu, odszukujemy pracę zawierająca nasze zadanie (Uwaga! Kryteria akceptacji są związane z pracami a nie zadaniem) i wpisujemy pożądane kryteria akceptacji (Rysunek 7.).



Rysunek 7. Przykładowy opis kryteriów akceptacji dla pracy.

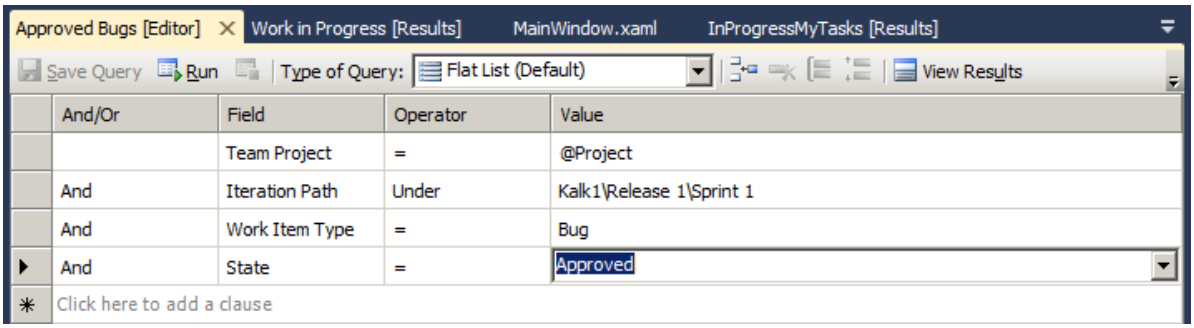
Teraz pojawia się pytanie, w jaki sposób poinformować o błędzie wykonującego. Przy czym poprzez poinformowanie mamy na myśli fakt ewidencji tego znaczenia w ramach środowiska, a nie to, że możemy wysłać do niego maila. Po pierwsze musimy przyjąć, czy zadanie jest już zamknięte, czy też nie. Jeśli jest zamknięte (status Done), to zmiana statusu z powrotem na In Progress, spowoduje, że programista zobaczy to zadanie w swoim wykazie, a czynności te będą zapamiętane w Historii dla tego zadania. Jeśli natomiast jest o statusie InProgress i koniecznie chcemy poinformować w systemie o tym zdarzeniu, to najprostszą drogą jest zgłoszenie błędu. By to zrobić możemy np. wybrać menu Bug w ramach menu Team, jednak wtedy będziemy musieli jeszcze podłączać ten błąd do naszego zadania, co nie jest wygodnie. Lepiej pójść inna drogą:

1. Ponownie otwieramy wykazu prac sprintu.
2. Na liście prac i zadań zaznaczmy wiersz naszego zadania i naciskamy prawy klawisz myszy, z menu wybieramy New Linked Work Item (skrót Shift+Alt+L).
3. Otworzy się znane już nam okno, które wykorzystywaliśmy do podłączania zadań do prac (Rysunek 8.). Teraz jednak w liście Work Item Type wybieramy Bug. A w tytule wpisujemy odpowiedni tekst (np. jak poniżej). Naciskamy OK.



Rysunek 8. Dodanie błędu skojarzonego z zadaniem.

4. Pojawi się okno pozwalające określić szczegóły zgłaszanego błędu. Ponieważ jednak szerzej je omówimy w module ???, to w tej chwili pozostawiamy wszystkie pola z domyślnymi wartościami i naciskamy Save Work Item.
5. Chcielibyśmy teraz gdzieś zobaczyć ten błąd przydzielony do nas. Pamiętamy, że już w poprzednim module zdefiniowaliśmy własne zapytanie o nazwie InProgressMyTasks oraz ToDoTasks, ale niestety po ich wywołaniu ten błąd nigdzie się nie pojawia. Nie ma w tym nic dziwnego zapytania dotyczyły przecież zadań. Jeśli nie pamiętamy ich treści, to na danym zapytaniu klikamy prawym klawiszem myszy i z menu wybieramy Edit Query – otworzy się okno w którym możemy zweryfikować jego warunki.
6. Możemy zdefiniować własne zapytanie, które pokaże nam błędy, lub po prostu wejść do portalu projektu i tam w oknie MyBugs, zobaczymy przypisany błąd.
7. Teraz kliknijmy w ten błąd, zmieńmy stan na Approved, zapiszmy zmiany i zamknijmy okno.
8. Przystępujemy do poprawienia wyglądu Kalkulatora. Wykonajmy niezbędne zmiany programistyczne (format zera poprawmy na „0.0” i zauważmy, że klient nie oczekiwał obsługi pamięci, zatem usuwamy dwa przyciski). Zbudujmy i wykonajmy aplikację, by przekonać się, ze nie wystąpiły nieprzewidziane problemy.
9. Po wykonaniu poprawek znów zatwierdzamy zmiany do repozytorium, w oknie chcemy zaznaczyć zarówno zadanie (jak poprzednio) jak i błąd, który był z tym zadaniem skojarzony, ale nie mamy w dostępnych pytaniach takiego, które pokazałoby błędy. Anulujemy zatem tę operację i dodamy nowe zapytanie.
10. W oknie Team Explorer otwieramy węzeł Current Sprint w ramach My Queries. Następnie klikamy w zapytanie Work In Progress i naciskamy Ctrl+C, a następnie Ctrl+V. Zostaniemy zapytani o nową nazwę wstawianego pytania, wpiszmy Approved Bugs i naciśnijmy OK. Teraz klikamy prawym na nazwie tego pytania i z menu wybieramy Edit Query, następnie modyfikujemy zapytanie jak na rysunku poniżej. Zapisujemy to zapytanie.



Rysunek 9. Definiowanie zapytania Approved Bugs, które pozwoli wybrać zaaprobowane błędy.

11. Ponownie wracamy do zatwierdzenie poprawek do repozytorium. Gdy otworzy się okno znane z rysunku 5, wchodzimy do zakładki Work Items, wybieramy zapytanie Work In Progress (Kalk1) i zaznaczamy zadanie, potem wybieramy zapytanie Approved Bugs (Kalk1) i zaznaczamy nasz błąd. Wracamy do zakładki Source i wpisujemy odpowiedni komentarz. Ostatecznie naciskamy Check In.
12. Teraz zamknijmy zadanie – wchodzimy do zadania i zmieniamy jego status na Done. Zapisujemy zmianę.
13. Na koniec zamknijmy błąd – wchodzimy do błędu i zmieniamy jego status na Done. Oj, chyba coś popsuliśmy, bo nie ma dostępnego tego statusu jest tylko następny po Approved, czyli Commited! Dalej zajmiemy się tym w Laboratorium podstawowym.

Prosta refaktoryzacja

Przejdźmy teraz do przykładu prostej refaktoryzacji, z racji złożoności zagadnienia, nie będziemy wspominać o całej klasie wzorców programistycznych, anie projektowych, a ograniczymy się jedynie do pokazania jak za pomocą narzędzia VS 2010 można wspomóc proces refaktoryzacji.

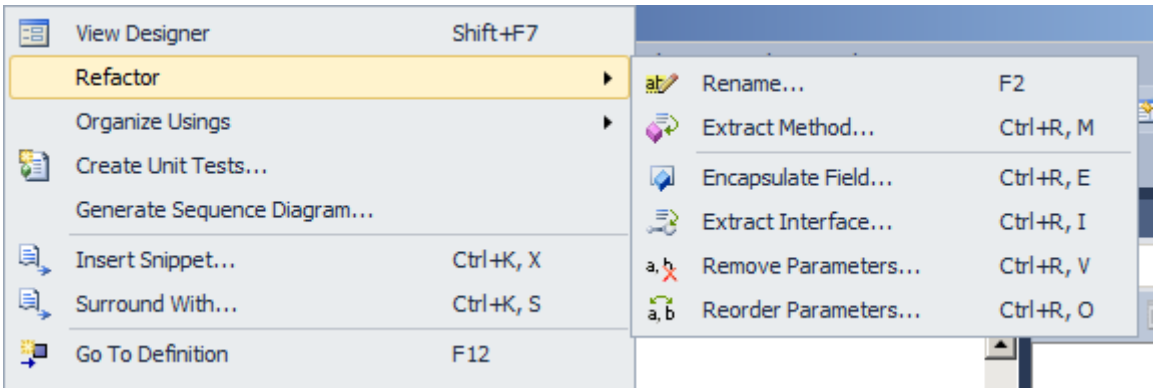
Zmiana nazwy

Tworzycie dalej aplikację kalkulatora i chcecie stworzyć funkcję, która zostanie wykonana po kliknięciu na przycisk czyszczenia „C”. W tym celu w oknie projektowania aplikacji dwa razy klikacie na ten przycisk, co spowoduje otwarcie okna z kodem programu i automatyczne wygenerowanie funkcji jak poniżej, lub podobnej, np. może być inny numer przy słowie button:

```
private void button1_Click(object sender, RoutedEventArgs e)
{


}
```

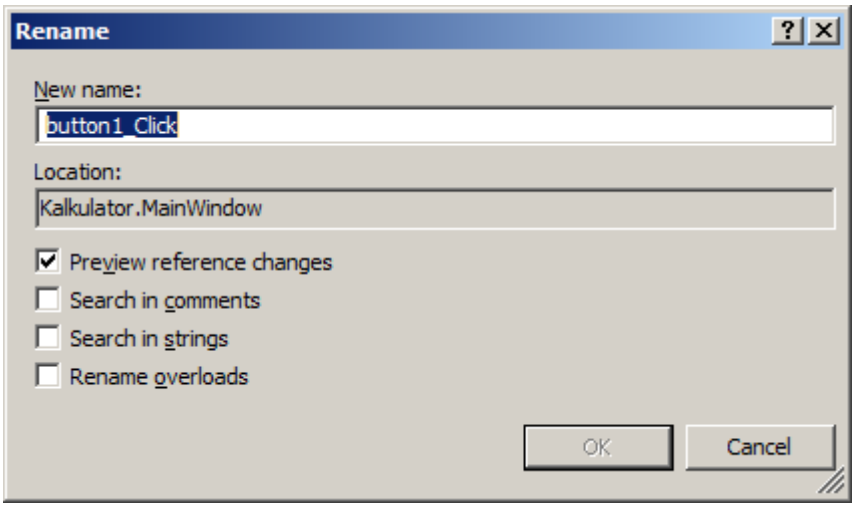
Widać wyraźnie, że nazwa metody, która ma obsługiwać tę akcję została nazwana poprzez sklejenie nazwy przycisku (w naszym przypadku button1) oraz słowa oznaczającego akcję, którą obsługuje ta metoda. Oczywiście sam fakt łączenia nazwy elementu GUI oraz akcji, która będzie na tym elemencie wykonana należy do bardzo dobrych praktyk, gdyż na pierwszy rzut oka jest jasne, do czego może służyć dana metoda. Jednak popełniliśmy błąd, gdyż zanim spowodowaliśmy wygenerowanie tego kodu powinniśmy zmienić nazwę przycisku na taką, która niesie więcej informacji np. buttonCzyszczenie – kwestia mieszania nazw polskich z angielskimi również często podlega dyskusji – zatem może lepiej buttonClear. Zmieńmy nazwę przycisku (Properties dla tego przycisku) na sugerowaną przed chwilą. Niestety zmiana nazwy przycisku nie spowodowała zmiany nazwy metody.

Musimy zastosować pierwszą z metod refaktoryzacji – zmianę nazwy. W tym celu otwieramy plik zawierający kod metody (MainWindow.xaml.cs) i najeżdżamy kursorem mysz na nazwę metody, naciskamy prawy klawisz i pojawia się duże menu kontekstowe, z którego nas interesuje podmenu Refactor, rozwińmy je (Rysunek 10.).



Rysunek 10. Menu refaktoryzacji.

Widzimy w tym menu różne metody refaktoryzacji, z których w tej chwili chcemy wybrać Rename, klikamy, lub naciskamy klawisz F2.



Rysunek 11. Okno dialogowe parametrów zmiany nazwy.

Widzimy, że pojawiło się nowe okno dialogowe (Rysunek 11.), pozwalające określić dodatkowe parametry dla tego procesu. Warto pozostawić opcję podglądu tego co zostanie zmienione, widać, że od razu możemy dokonać zmian w komentarza, napisach i wymusić zmianę nazwy. Z ta ostatnia opcją należy postępować ostrożnie, gdyż pochopna zmiana nazwy w niektórych miejscach może spowodować iż kod przestanie się kompilować. Wprowadzamy nową nazwę (buttonClear) i naciskamy przycisk OK. Niestety dostaliśmy komunikat o konflikcie z istniejącą nazwą (zmieniliśmy nazwę przycisku i środowisko nie wie co zrobić), zatem wróćmy do starej nazwy przycisku (button1). Powtórzmy operację zmiany nazwy. Tym razem powinno pojawić się okno pokazujące w których miejscach nastąpi zmiana, zatwierdźmy je i dopiero teraz zmieńmy nazwę przycisku na docelową.

Zbudujmy aplikację – wykonujmy zawsze tę operację po jakiejkolwiek refaktoryzacji. Dzięki temu unikniemy sytuacji w której nie wiemy co się stało, bo wykonaliśmy 3 refaktoryzacje z rzędu i zmiany się nałożyły na siebie. Jeśli budowa aplikacji przebiegła poprawnie, to nie ma kłopotu i możemy pracować dalej. To czy w tym momencie zapiszemy zmiany do repozytorium czy nie, zależy czy ta refaktoryzacja jest w całości zmianą, czy elementem większych zmian.

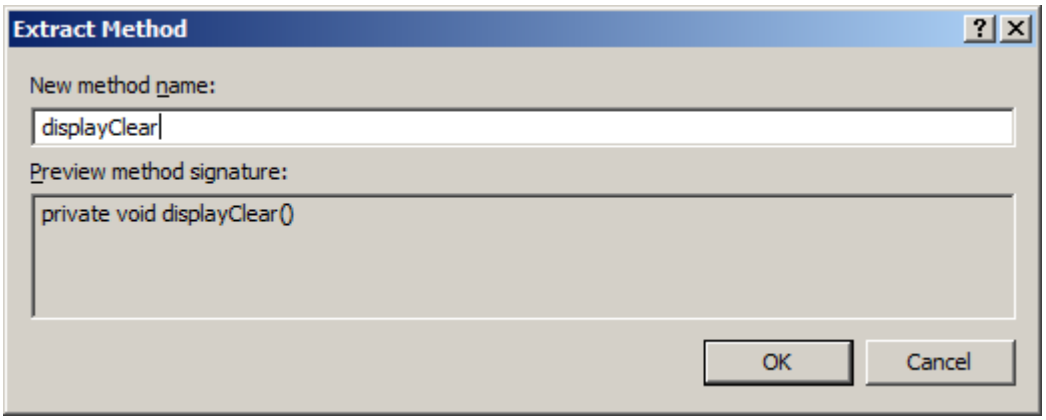
Omówiona metoda refaktoryzacji, choć bardzo prosta w idei, jest bardzo często wykorzystywana, a mechanizm dostarczany przez środowisko, który umożliwia automatyczną zamianę wszystkich odwołań jest niewątpliwie bardzo cenny.

Ekstrakcja metody

Zmieńmy nazwę komponentu textBlock1, który reprezentuje wyświetlacz w naszym Kalkulatorze na textBlockDisplay. Teraz idźmy do metody buutonClear w wpiszmy w niej następujący przykładowy kod:

```
private void buttonClear(object sender, RoutedEventArgs e)
{
  textBlockDisplay.Text = "Wyczyszczono zawortość";
}
```

Kod ten, dość naiwny, umieszcza komunikat o wyczyszczeniu kalkulatora w oknie wyświetlacza i nic więcej nie robi. Wyobraźmy sobie teraz, iż doszliśmy do wniosku, że sam fakt czyszczenie mógłby być osobną metodą, gdyż istniej potrzeba wywołania go w różnych miejscach. Zaznaczmy zatem linie kodu z których chcemy utworzyć nowa metodę (w naszym przypadku jest to jedna linia). Najedźmy na ten obszar kursorem myszy, naciśnijmy prawy klawisz i z menu wybierzmy kolejno Refactor, a potem Extract Method. Pojawi się okno (Rysunek 12.), w którym wpisujemy pożądaną nazwę nowej metody i naciskamy przycisk OK.



Rysunek 12. Okno dialogowe pozwalające wprowadzić nazwę dla ekstrahowanej metody.

Na skutek tych czynności, zostanie utworzona nowa metoda o podanej przez nas nazwie i kod źródłowy zostanie zamieniony na odwołanie do niej:

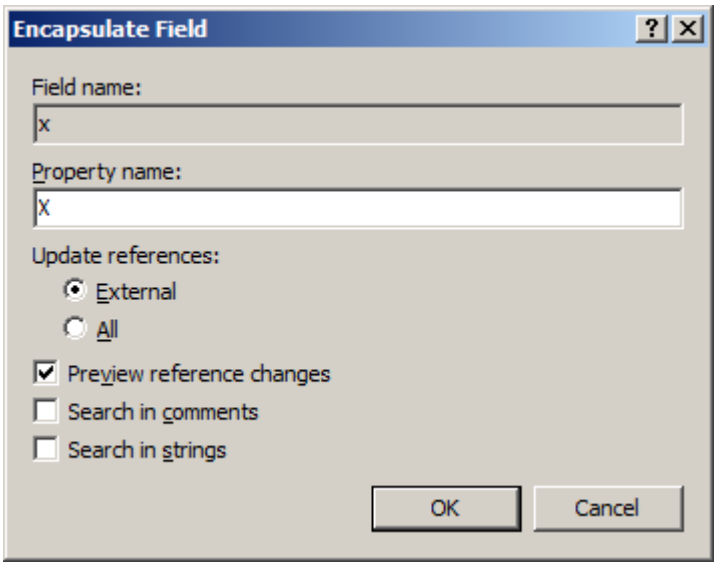
```
private void buttonClear(object sender, RoutedEventArgs e)
{
  displayClear();
}


private void displayClear()
{
  textBlockDisplay.Text = "Wyczyszczono zawortość";
}
```

Jeśli wydzielalibyśmy kod gdzie istniałyby odwołania do zmiennych, to środowisko podpowiedziałoby funkcję z odpowiednimi argumentami.

Proces ekstrakcji metody jest często wykorzystywany w praktyce codziennej programistów.

Enkapsulacja składowych klasy

Kolejną bardzo przydatną metodą refaktoryzacji jest enkapsulacja składowych klasy. Wpiszmy do klasy MainWidnow, składową typu całkowitego o nazwie x. Następnie najedźmy na nią kursorem myszy i dalej naciśnijmy prawy klawisz, następnie wybierzmy kolejno Refactor i Encapsulate Field, pojawi się okno dodatkowych parametrów (Rysunek 13.).



Rysunek 13. Okno parametrów dla enkapsulacji składowych klasy.

Zostaniemy zapytania o nazwę właściwości, która ma opakować naszą zmienną, to czy możemy podejrzeć zmiany zanim zostaną wykonane oraz podobnie jak w przypadku zmiany nazwy o przeszukiwanie komentarzy oraz napisów. Dodatkowo zostaniemy zapytani, czy mają zostać zmienione odwołanie zewnętrze czy wszystkie (zatem i wewnętrzne). Wybór pozostawiamy programiści. Naciskamy przycisk OK i zostaje wygenerowany kod jak poniżej:

```
public partial class MainWindow : Window
{
  int x;

  public int X
  {
    get { return x; }
    set { x = value; }
  }
  ...
```

Zatem otrzymaliśmy utworzone tzw. setery i gettery dla naszej składowej.

Pozostałe trzy sposoby refaktoryzacji pozostawiamy jako ćwiczenie indywidualne.

Analiza kodu

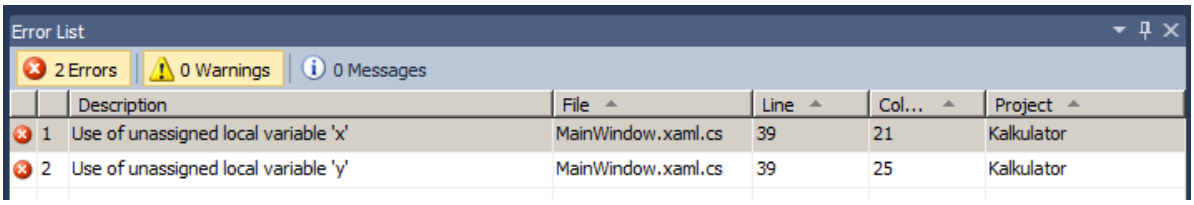
Zupełnie drugi sposób patrzenia na kod dają różnego rodzaju analizatory, które wspomagają śledzenie tego co potrzebne lub nie w kodzie. W zależności od stopnia zaawansowania tychże, mogą one wręcza same wynajdywać kawałki kodu, które będą kandydatami dla ekstrakcji metody, by wykorzystać ją w dwu miejscach. Inne wynajdują wspólne fragmenty występujące w kilku

miejscach programu i proponują zamianę na wywołania tej samej funkcji. Rozwiązań tych, szczególnie komercyjnych, jest obecnie dużo i możemy wybierać w zależności o potrzeb.

W przypadku Visual Studio również mamy pewne rozwiązania, które mogą wspomóc nas w tworzeniu oprogramowania. Wpierw zmodyfikujmy kod metody tak jak to podano niżej (oczywiście zdajemy sobie sprawę z naiwności przykładu).

```
private void buttonClear(object sender, RoutedEventArgs e)
{
  displayClear();
  int x;
  int y;
  int z = x + y;
}
```

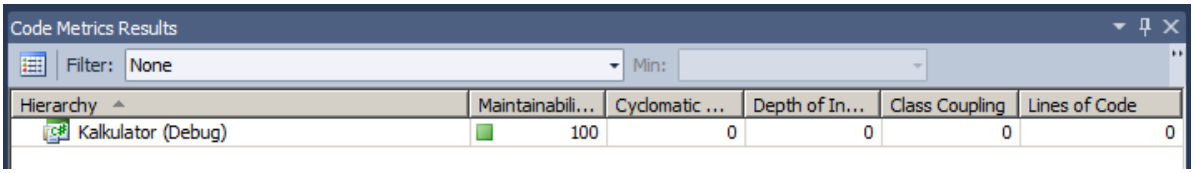
Następnie w oknie Solution Explorer w drzewie aplikacji, klikamy prawym klawiszem mysz na nazwie aplikacji (Kalkulator) i z menu kontekstowego wybieramy pozycję Run Code Analysis, w efekcie zostanie wykonana analiza kody całego programu pod kontem różnego rodzaju wątpliwych rozwiązań i w naszym przypadku pojawią się dwa błędy (Rysunek 14.).



Rysunek 14. Błędy po analizie kodu.

Co prawda te błędy, które wyskoczyły również umożliwiałyby budowę programu, ale na przykład z ostrzeżeniami już tak nie będzie, a warto je usuwać.

Drugim dobrym narzędziem wspomagającym analizę kody są metryki. Przy czym ograniczymy się jedynie do bardzo krótkiego omówienia tych metryk, które dostarcza VS 2010, gdyż temat metryk kodu jest sam z siebie ogromny. Najedźmy kursorem myszy na główny węzeł solucji (Solution 'Kalkulator'), naciśnijmy prawy klawisz i wybierzmy z menu Calculate Code Metrics. Po wykonaniu analizy przez środowiska zostanie wyświetlony raport (Rysunek 15.).



Rysunek 15. Rezultat analizy kodu pod kontem pomiaru różnego rodzaju metryk.

W tym oknie mamy dostępne następujące metryki:

- **Maintainability Index** – wskazuje na ile łatwo kod będzie poddawał się utrzymaniu. Im wyższa wartość tym lepiej.
- **Cyclomatic Complexity** – tzw. złożoność cyklomatyczna, wskaźnik ten mówi o ilości rozgałęzień kodu (np. funkcji). Im mniejsza wartość tym lepiej.
- **Depth of Inheritance** – wskaźnik określa ilości (głębokości) dziedziczenia. Im mniejsza wartość tym lepiej.
- **Class Coupling** – określa ilość klas z wzajemnymi odwołaniami, im mniejsza wartość tym lepiej.

- **Lines of Code** – wskaźnik ilości kodu. Im mniejsza wartość tym lepiej.

Oczywiście w naszym przypadku, gdy program jest banalnie prosty wskaźniki te są dość oczywiste i nie wnoszące wiele. Jednak w miarę wzrostu złożoności programu dostarczają one pożytecznych informacji.

Porady praktyczne

Nie sposób w tak krótkim omówienie przedstawić wszystkich zagadnień związanych z zarządzaniem jakością, ani nawet sporządzić solidny przegląd tych zagadnień. Stąd też następne moduły należy postrzegać niejako jako rozwinięcie bieżącego. Szczególnie zarządzanie ryzykiem (moduł 7) czy też omówienie procesów testowania (moduł ???).

Po pierwsze, pamiętajmy, że zapewnianie jakości, czy szerzej zarządzanie jakością jest procesem ciągłym, a nie pojedynczym zdarzeniem. Zatem najlepiej mieć wyznaczoną rolę (w większych firmach robi to nawet zespół osób), która ma wciąż na uwadze ten proces.

Po drugie, miejmy na uwadze, że o jakości nie wolno zapominać. Niestety praktyka pokazuje, że najczęściej jeśli gdzieś oszczędzamy to właśnie na jakości. Niestety jest to najszybsza droga do spowodowania poważnych problemów w projekcie.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- wiesz z jakimi zagadnieniami wiąże się zapewniania jakości,
- wiesz co to jest zapewnianie jakości kodu, programu czy też dokumentacji,
- potrafisz określić czym jest standaryzacja kodowania,
- potrafisz wskazać na źródła dotyczące standardów kodowania czy dokumentowania,
- potrafisz przeprowadzić prostą refaktoryzację kodu,
- potrafisz stworzyć prosty przypadek testowy,
- rozumiesz role zapewniania jakości w całym cyklu życia oprogramowania.

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. Zalecenia standardu kodowania wg. MSDN firmy Microsoft
   http://msdn.microsoft.com/en-us/library/aa291596(VS.71).aspx

   Na stronie tej znajdują się zalecenia standardu kodowania firmy Microsoft.

Laboratorium podstawowe

Zadanie 1

Pierwszym Twoim zadaniem będzie weryfikacja poprawności powyższego fragmentu przykładowego rozwiązania dotyczącego błędów.

Zadanie	Tok postępowania
1. Wykrycie pomyłki	• Cofnij się i przeczytaj jeszcze raz podpunkt 13 w punkcie Błąd w wykonaniu zadania. • Teraz przypomnij sobie, że Błąd jest zwykły elementem typy Work Item (patrz np. rysunek 7 w module 4 i opis z nim związany). • Przypomnij sobie cykl życia elementów typu Work Item (patrz rysunek 14 w module 4 i opis z nim związany). • Zatem dlaczego przyjęliśmy, że wystarczy zmienić status błędu na Approved, by móc wykonać zadanie (poprawić błąd)?
2. Poprawa pomyłki	• Jako programista przydzielony do poprawy błędu zmień jego satus na Commited. • Zmień zapytanie znane z rysunku 9 na takie by wybierało błędy ze statusem Commited.
3. Poprawa zadania	• Spróbuj wycofać się do kroku 7 w punkcie Błąd w wykonaniu zadania, a następnie spróbuje wykonać już wszystkie kroki poprawnie.

Zadanie 2

W drugim zadaniu masz dokończyć rozpoznanie pozostałych trzech metod refaktoryzacji (rysunek 10.), tzn.:

1. Extract Interface.
2. Remove parameters.
3. Reorder parametars.

Laboratorium rozszerzone

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 7

Wersja 1

Wstęp do zarządzenia ryzykiem

Spis treści

Informacje o module

Opis modułu

W module tym znajdziesz podstawowe informacje z zakresu zarządzania ryzykiem w projektach informatycznych. Dowiesz się co rozumiemy pod pojęcie ryzyka, jakie są czynniki mogące wpływać w negatywny i pozytywny sposób na projekt. Będziesz wiedział jakie procesy wchodzą w skład zarządzania ryzykiem, wiedział na czym polega identyfikacja ryzyk, ich analiza. Dowiesz się procesie zapobiegania i monitorowania ryzykiem.

Dalej zapoznasz się, na prostym przykładzie, jak podejść do identyfikacji ryzyk w projekcie i jak je można zapisać w środowisku Visual Studio 2010 oraz Team Foundation System 2010, przy wykorzystaniu szablonu procesu Visual Studio Scrum 1.0.

Cel modułu

Celem modułu jest przedstawienia podstawowych zagadnień w zakresie zarządzania ryzykiem w projekcie informatycznym.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- wiedział co rozumiemy pod pojęciem ryzyka,
- wiedział jakie procesy wchodzą w zakres zarządzanie ryzykiem,
- potrafił przeprowadzić identyfikację ryzyk oraz ich analizę,
- potrafił stworzyć prosty rejestr ryzyk, oraz zapisać niektóre z nim w ramach środowiska VS 2010,
- rozumiał rolę zarządzania ryzykiem w projekcie i powiązanie z procesem zapewniania jakości.

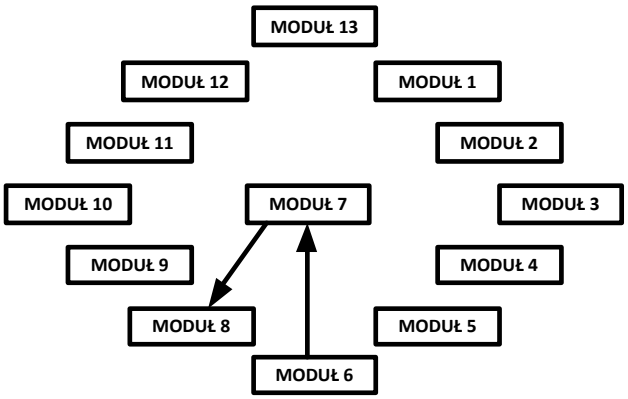
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawy obsługi środowiska Visual Studio 2010 oraz Team Foundation Studio 2010,
- znać podstawy pracy z szablonem Visual Studio Scrum 1.0,
- znać podstawy tworzenia aplikacji Windows (preferowany WPF) w środowisku Visual Studio 2010 przy użyciu języka C#,
- rozumieć czym jest cykl wytwarzania oprogramowania i jakie procesy w nim występują,
- rozumieć czym jest proces zapewniania jakości.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w module



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Pierwszy Sprint macie za sobą, jesteście zadowoleni, klient jest zadowolony, gdyż otrzymał to, co zostało ustalone w trakcie planowania sprintu. Jednym słowem pełny sukces. Zaplanowaliście kolejny sprint i już minęły 3 dni.

W między czasie Klient zaimportował dane ze starego systemu do aplikacji, którą otrzymał po pierwszej iteracji i po pół dnia pracy po prostu się wściekł. Właściciel produktu przyszedł z ogromnymi pretensjami, gdyż aplikacja, która zawiera kartotekę pracowników chodzi tak wolno, że nie daje się pracować. Po wstępnych oględzinach stwierdziliście, iż faktycznie przejście pomiędzy pracownikami w kartotece jest uciążliwe, gdyż trwa kilka sekund, jednak okazało się, że jeszcze gorzej jest w przypadku próby wydruku listy pracowników zwolnionych w pewnym miesiącu – generowanie takiego wydruku na bazie złożonej z 3400 pracowników trwa ok. 1,5 minuty. Właściciel produktu stwierdza, że jest to nieakceptowalne, a Wy zaczynacie się zastanawiać, co się właściwie stało?

Po jakimś czasie, i przeprowadzeniu szeregu analiz oraz testów okazało się, iż najnowsza wersja biblioteki dostępu do warstwy danych (dane przechowujecie w bazie SQL), źle współpracuje z tą wersją bazy, którą posiadacie. Właściwie rozwiązania są dwa, podnieść wersję bazy (upgrade), lub wrócić z biblioteką do wersji niższej (downgrade). Przy czym każda z tych dróg ma swoje wady, upgrade bazy wiąże się u klienta z wydatkiem pieniędzy, downgrade biblioteki jest poważnym zagrożeniem dla bezpieczeństwa aplikacji, gdyż poprzednia wersja miała znane dziury, a do tego wymaga sporych zmian w kodzie, zatem dla wiąże się z kosztami po Waszej stronie.

Niezależnie, jaką decyzję podejmiecie, wspólnie z Klientem, wiąże ona się z pewnymi kosztami i ryzykiem. Ale dużo ważniejsze jest to, jak to się stało, jak doszło do sytuacji, że administrator w waszej firmie dokonał aktualizacji biblioteki i nikt o tym nie wiedział? Co więcej, nikt nie uzgodnił tego z Klientem.

Podstawy teoretyczne

Zarządzanie ryzykiem w każdym projekcie, nie zależnie od jego wielkości, rangi, czy złożoności, jest procesem, którego nie wolno bagatelizować. Co rozumiemy pod pojęciem „ryzyka" w projekcie? Potocznie pod tym pojęciem rozumiemy zagrożenia, mówimy na przykład „ryzykownie jeździsz", „to jest ryzykowna gra". Zatem przyjmując tę potoczną, intuicyjną „definicję" rozumiemy ryzyko jako coś, co gdy się wydarzy, przyniesie nam jakieś „straty".

W projektach, definicja ryzyka jest rozszerzana również na szanse, zatem mówimy o zdarzeniach, które, jeśli wystąpią wiążą się ze stratą lub zyskiem. Oczywiście z ryzykiem związane jest prawdopodobieństwo wystąpienia danego zdarzenia, koszt w przypadku jego wystąpienia i kilka rzeczy, o których powiemy dalej. Można przyjąć, że celem zarządzania ryzykiem w projekcie jest zmniejszenie prawdopodobieństwa, uniknięcie lub zmniejszenie skutków ryzyk niosących zagrożenia, oraz zwiększenie prawdopodobieństwa i skutków szans, jakie się pojawiają.

W ramach zarządzania ryzykiem można wyróżnić kilka procesów, ich pogrupowanie i charakterystyka jest różna w zależności od metodyki zarządzania, niemniej jednak podział jest podobny, my będziemy posługiwali się poniższym:

1. **Identyfikacja ryzyka** – proces rozpoznawania, jakie ryzyka mogą wystąpić w projekcie.
2. **Analiza ryzyka** – proces oceny jakościowej i ilościowej ryzyk wcześniej zidentyfikowanych.
3. **Zapobieganie ryzykom** – proces opracowywania możliwych działań zwiększających szanse i zmniejszających zagrożenia.
4. **Monitorowanie ryzyka** – ciągły proces śledzenia wszystkich ryzyk zidentyfikowanych w projekcie, jak i rozpoznawania nowych. W procesie tym też często kontroluje się skuteczność całego procesu zarządzania ryzykiem.

Zanim zajmiemy się dokładniejszym omówieniem tych grup procesów, wpierw dokonamy pewnych obserwacji związanych z ryzykiem, a następnie podamy przykłady kilku ryzyk, aby łatwiej było zrozumieć cały proces zarządzania ryzykiem. Zauważmy, że:

- Ryzyko zawsze dotyczy przyszłości, jeśli już nastąpi nie mówimy o nim ryzyko, a wtedy nazywane jest problemem czy zagadnieniem (możliwe inne nazewnictwo).
- Jeśli się wydarzy to ma wpływ, na co najmniej jeden z wymiarów projektu: czas, zakres, budżet, jakość.
- Każde ryzyko ma zawsze jedną przyczynę bezpośrednią.
- Ryzyko może mieć wiele skutków.
- Pojedyncze ryzyko może być jednocześnie źródłem zagrożenia jak i szansy (patrz przykład poniżej).
- Ryzyka mogą być znane (zidentyfikowane) – w tym przypadku można mówić o zarządzaniu i działaniach zapobiegawczych i nieznane (niezidentyfikowane, których wystąpienia nie przewidzieliśmy) – dla nich nie ma mowy o konkretnych działaniach zapobiegawczych i możemy mówić jedynie o ogólnych planach awaryjnych.

Oto przykładowe ryzyka:

- Organizacja, w której realizowany jest projekt podlega pod ustawę o zamówieniach publicznych, a w ramach projektu ma być zakupiony zestaw serwerów i terminali. Ryzykiem będącym zagrożeniem jest opóźnienie procedury przetargowej wynikającej np.: z protestów czy unieważnienia przetargu i w efekcie nie dotrzymanie terminów wykonania projektu. Zaś szansą mogącą się pojawić jest zakup planowanego sprzętu w terminie za znacznie niższą kwotę niż planowano.
- Projekt jest realizowany przez 5-cio osobowy zespół, w którym jedna osoba posiada bardzo specyficzną wiedzę, osób znających ten obszar jest kilka w regionie i wszystkie są zaangażowane w projekty. Oczywistym zagrożeniem dla projektu jest „zniknięcie" tej osoby (przyczyn takiego faktu może być wiele: zmiana pracy, choroba a nawet śmierć).
- Serwerownia, w której stanęły komputery związane z projektem stoi w piwnicy. Zagrożeniem dla projektu będzie np. zalanie serwerowni i zniszczenie sprzętu.
- Projekt jest realizowany w oparciu o nową technologię (np. biblioteka dostępu do bazy danych). Zagrożeniem może być np. nieoczekiwany spadek wydajności aplikacji przy dużych operacjach na danych, a wynikających z błędu w bibliotece. Szansą może być np. wzrost prędkości tworzenia aplikacji wynikającej z łatwości użytkowania tej biblioteki przez programistów.
- Firma produkuje oprogramowania na zamówienie zagranicznej organizacji. W tej sytuacji jest narażona na ryzyko kursów walut, ale może ona być zarówno przyczyną zagrożenia (niekorzystny kurs dla firmy), jak i szansy (wyjątkowo korzystny kurs walut).

Identyfikacja ryzyka

Celem procesu identyfikacji ryzyka jest stworzenie możliwie kompletnej listy szans i zagrożeń dla projektu. To właśnie ta lista stanowi podstawę do zarządzanie ryzykiem i tylko, jak pisaliśmy, zidentyfikowanymi ryzykami można zarządzać. Stąd też należy dążyć, by była ona możliwie pełna. Należy jednak pamiętać, że nigdy lub prawie nigdy nie będzie ona kompletna, a gdyby nawet taka była, to my nie mamy co do tego faktu pewności. Zatem należy zapewnić wiele źródeł, z których pozyskujemy informacje o ryzykach, tak by lista była możliwie kompletna. W tym celu mamy do dyspozycji wiele technik pozyskiwania tych informacji, wymienimy kilka z nich:

- **Listy kontrolne** (ang. checklists) – jedna z podstawowych technik opierająca się na doświadczeniu zespołu. Listy takie tworzy się na podstawie danych z poprzednich projektów, jakie realizował zespół, lub inne zespoły w firmie, oraz na podstawie innych źródeł informacji

o ryzykach. Należy również pamiętać, że listy ryzyk należy aktualizować w trakcie realizacji projektu, dzięki czemu będą lepszym źródeł informacji dla nowych projektów.

- **Burza mózgów** (ang. Brainstorming) – jest to bardzo popularna technika, często wykonywana nawet nieświadomie w trakcie realizacji projektu. Technika ta sprowadza się do zgromadzenia zespołu, i ewentualnych ekspertów, w jednym pomieszczeniu w celu przeprowadzenia „burzliwej" rozmowy na temat projektu. By uporządkować dyskusję wyznacza się moderatora i zwykle to on nadaje rytm całemu spotkaniu. W trakcie tego spotkania uczestnicy zgłaszają swoje propozycję do rejestru ryzyk. Inną modyfikacją tej techniki jest ankietowanie określonej grupy osób i odpytywanie ich z zakresu ryzyk projektu. W jednym i drugim przypadku zwykle powstaje luźna lista ryzyk, którą należy uporządkować i pogrupować.
- **Metoda delficka** – celem tej techniki jest osiągnięcie konsensusu w gronie ekspertów, pomaga ona zminimalizować wpływ subiektywności, czy dominacji pojedynczej jednostki. Uczestnictwo w tej technice jest anonimowe, a osoba przeprowadzająca badanie zbiera opinie ekspertów poprzez ankietę (w dowolnej formie, np. papierowej czy elektronicznej). Zwykle otrzymane odpowiedzi gromadzi się, a następnie opracowuje poprzez zestawienie z innymi i odsyła do ekspertów w celu uzyskania komentarzy. Proces uzgadniania powtarza się kilkukrotnie w celu uzyskania konsensusu.
- **Analiza SWOT** – jest to bardzo znana technika, wykorzystywana nie tylko do analizy ryzyka, ale również np. w przypadku podejmowanie jakiegoś przedsięwzięcia. Opiera się ona na spojrzeniu na projekt z czterech stron: atutów, słabości, szans i zagrożeń. Technikę tę rozpoczyna się od rozpoznawania atutów i słabości, następnie określa się szanse oraz wszelkie zagrożenia. Same elementy wchodzące w skład tej analizy określa się zwykle za pomocą burzy mózgów. Stosując analizę SWOT pamiętajmy zwykle dotyczy całości projektu, a nie pojedynczego ryzyka.
- **Diagramy** – tego punktu nie będziemy opisywać, gdyż wymagałby on osobnego szerokiego omówienia. Niemniej jednak należy stwierdzić, że istnieje wiele technik analizy ryzyk opartych o tzw. diagramy, można tu wymienić: diagramy przyczynowo-skutkowe, schematy blokowe systemu, diagramy wpływów.

Analiza ryzyka

Analizę ryzyka przeprowadza się często w dwu ujęciach: jakościowym i ilościowym, my skupimy się w niniejszym opracowaniu na tym pierwszym. Analiza jakościowa jest etapem, w którym ryzyka są grupowane, kategoryzacja, przypisywane są im prawdopodobieństwa oraz skutki w przypadku, gdy się wydarzą. W celu przeprowadzenia jakościowej analizy ryzyk można stosować następujące techniki i narzędzia:

- **Ocena prawdopodobieństwa i skutków ryzyk** – jest to jedna z technik, bez której nie ma mowy o dalszej analizie. Sprowadza się ona do przypisania każdemu, wcześniej zidentyfikowanemu, ryzyku prawdopodobieństwa jego wystąpienia oraz skutków, jakie spowoduje, gdy się wydarzy. Wydaje się to pozornie proste, jednak należy zauważyć, że wielokrotnie nie jest to proste. Przykładowo jak ocenić prawdopodobieństwo zalania serwerowni? A jak wycenić odejście pracownika z firmy? Ocenę ryzyka można przeprowadzać np. poprzez burzę mózgów lub metodę delficką, straty wylicza się najczęściej modelami ekonomicznymi (podobnie z zyskami).
- **Macierz prawdopodobieństwa i skutków ryzyk** – jest to jeden ze sposobów klasyfikacji ryzyk, przy czym reguły tej klasyfikacji powinny być określone przez organizację przed samym projektem – w celu uniknięcia „dopasowywania" reguł klasyfikacji do oczekiwanych wyników. Istotność każdego ryzyka, a co za tym idzie, kolejność, w jakiej należy się nim zająć wygodnie jest określać za pomocą macierzy prawdopodobieństw i skutków ryzyk (Rysunek 2). Za pomocą tej macierzy w bardziej czytelny sposób możemy zobrazować te ryzyka, które są „mało istotne", „średnio istotne", „bardzo istotne" (na przykładowej macierzy zaznaczone

to jest stopniami szarości). Natomiast wartości, które są w komórkach zaprezentowanej macierzy mogą być wyliczane na różne sposoby i to od zasad przyjętych w danej organizacji zależy i od specyfiki projektu (dla jednego może być to inaczej niż dla drugiego). Najprostszym sposobem wyliczenia jest wymnożenie wartości prawdopodobieństwa przez wartość skutku (spływu na projekt). Taką macierz można zrobić dla każdego z ryzyk oddzielnie, albo można próbować stworzyć wspólny sposób klasyfikacji dla pewnej grupy ryzyk. Co więcej, można klasyfikować ryzyka oddzielnie dla każdego z wymiaru projektu (czas, zakres, budżet, jakość) (Rysunek 3).

P-stwo	**Zagrożenia**					**Szanse**				
0,9	0,18	0,36	0,54	0,72	0,9	0,9	0,72	0,54	0,36	0,18
0,7	0,14	0,28	0,42	0,56	0,7	0,7	0,56	0,42	0,28	0,14
0,5	0,1	0,2	0,3	0,4	0,5	0,5	0,4	0,3	0,2	0,1
0,3	0,06	0,12	0,18	0,24	0,3	0,3	0,24	0,18	0,12	0,06
0,1	0,02	0,04	0,06	0,08	0,1	0,1	0,08	0,06	0,04	0,02
Koszt /Zysk	**1**	**2**	**3**	**4**	**5**	**5**	**4**	**3**	**2**	**1**

Rysunek 2. Przykładowa macierz prawdopodobieństwa i skutków ryzyk. W lewej kolumnie znajduje się prawdopodobieństwo wystąpienia, w dolnym wierszu koszt lub zysk lub inaczej oddziaływanie (znormalizowany w zakresie do 5), jaki możemy ponieść lub otrzymać w przypadku wystąpienia danego ryzyka.

	Wpływ na projekt				
Wymiar projektu	**Bardzo łagodny (1)**	**Łagodny (2)**	**Średni (3)**	**Znaczący (4)**	**Bardzo znaczący (5)**
Zakres	Niezauważalne zawężenie zakresu	Wpływ na mniej istotne obszary zakresu	Wpływ na istotne obszary zakresu	Zawężenie zakresu nieakceptowane przez Klienta	Zagrożenie dla istnienia projektu i jego rezultatu
Czas	Nieznaczne wydłużenie czasu realizacji	Małe wydłużenie czasu realizacji (C<5%)	Średnie wydłużenie czasu realizacji (5%<C<10%)	Znaczenie wydłużenie czasu realizacji (10%<C<20%)	Zagrożenie terminu wykonania (20%<C)
Budżet	Nieznaczny wzrost kosztów	Małe zwiększenie kosztów realizacji (B<10%)	Średnie zwiększenie kosztów realizacji (10%<B<20%)	Znaczenie zwiększenie kosztów realizacji (20%<B<40%)	Zagrożenie klęską projektu (40%<B)
Jakość	Niezauważalny spadek jakości	Wpływ jedynie na najostrzejsze kryteria	Wpływ wymagający decyzji Klienta	Wpływ nieakceptowany przez Klienta	Produkt niezdatny do użycia

Rysunek 3. Oddziaływanie skutków ryzyk będących zagrożeniami w czterech obszarach projektu. Dla szans można przedstawić analogiczną tabelę.

- Następnie na podstawie tak opracowanych wyników można przeprowadzać następujące procesy: ocena jakości danych o ryzyku, grupowanie ryzyk, ocena pilności.

Należy pamiętać, że ilość działań, narzędzi, jakie będziemy używać oraz dokumentów, jakie będą generowane w związku z zarządzaniem ryzykiem trzeba dostosować do wymagań wobec projektu, jego wielkości i złożoności. Wynika to z faktu, że same działania związane z zarządzaniem ryzykiem również mają swój koszt i jako takie mogą być biznesowo nieuzasadnione, więcej na ten temat w następnym punkcie.

Zapobieganie ryzykom

W ramach tego procesu staramy się zaplanować wszelkie działania, które zmniejszają prawdopodobieństwo i straty dla zagrożeń oraz zwiększają prawdopodobieństwo i zyski dla szans. Podjęcie tych działań jest naturalną konsekwencją analizy ryzyk, kiedy mamy już listą wraz z przypisaną do nich oceną ryzyka, to wiemy tym samym, w jakiej kolejności tymi ryzykami musimy się zająć.

W pierwszej kolejności, każdemu ryzyku z listy ryzyk w projekcie należy przypisać właściciela, jest to osoba, która ma śledzić to ryzyko i informować w przypadku np. wzrostu zagrożenia lub zmniejszenia szansy. Jedna osoba może być właścicielem wielu ryzyk, ale nigdy odwrotnie – zawsze ryzyko musi mieć określnego jednego i tylko jednego właściciela.

Następnie dla każdego ryzyka planujemy działania, o których pisaliśmy na początku. Dla dowolnego ryzyka możemy określać więcej niż jedno działanie zapobiegawcze. Dla przykładu, mamy ryzyko „Zalanie serwerowni znajdującej się w piwnicy”, możemy zatem planować działanie „Wykupienie polisy ubezpieczeniowej od zalania” oraz „Budowa serwerowni zapasowej”.

Każde działanie zapobiegawcze ma swój koszt, zatem jeśli miałoby być wykonane, należy uwzględnić (policzyć), czy wykonanie tego działania jest uzasadnione. Dla przykładu z poprzedniego akapitu, koszty wykupu polisy będzie wielokrotnie tańszy od budowy dodatkowej serwerowni, ale z drugiej strony wykup polisy nie zabezpieczy nas przed: utratą danych (o ile nie mamy backupu poza serwerownią) czy szybkim uruchomieniem serwerowni po awarii. Stąd też nie ma uniwersalnej reguły, kiedy działania sią uzasadnione, a kiedy nie, zawsze zależy to od wielu czynników. Znów w powyższym przykładzie sam efekt ekonomiczny może być złym doradcą, należy jeszcze uwzględnić czynnik ciągłości pracy, który wynika z charakteru projektu czy Klienta. Zauważmy, że inne potrzeby ma spełniać serwerownia w szkole – gdzie wykorzystuje się ja do dydaktyki, a zupełnie inne na lotnisku – gdzie jest elementem systemu kontroli lotów.

W tym miejscu wymienimy standardowe strategie dla zagrożeń:

- **Unikanie** (ang. avoid) – polega ono na zmianie planu projektu, tak by całkowicie wyeliminować to ryzyko. Typowymi przykładami jest wydłużenie czasu realizacji kontraktu, zatrudnienie dodatkowej osoby, w skrajności będzie to rezygnacja z jakiejś funkcjonalności, a nawet z całego projektu.
- **Przeniesienie** (ang. transfer) – wszystkie lub część niekorzystnych skutków przenosimy na inny podmiot. Typowym przykładem będzie w tym przypadku właśnie umowa ubezpieczenia. Należy jednak pamiętać, że w przypadku przeniesienia w żaden sposób nie zmniejszamy prawdopodobieństwa wystąpienia ryzyka, a jedynie pracujemy na jego skutkach.
- **Łagodzenie** (ang. mitigate) – łagodzenie sprowadza się do zmniejszenia prawdopodobieństwa wystąpienia ryzyka lub jego skutków do poziomu, który akceptujemy. Działanie to przypomina starą zasadę lekarską, iż łatwiej zapobiegać niż leczyć. Przykładami takich działań mogą być: zmniejszenie złożoności projektu, wykonanie wcześniejszego prototypu, zwiększenie liczby testów.
- **Akceptacja** (ang. accept) – bardzo rzadko można wyeliminować wszystkie ryzyka, dlatego też dla części z nich przyjmuje się strategię akceptacji. Wyróżniamy dwie postawy: pasywną i czynną. Pasywna postawa to brak jakichkolwiek działań do momentu wystąpienia tego

ryzyka. Postawa czynna może sprowadzać się do stworzenia rezerwy finansowej czy czasowej na ewentualne przyszłe wykorzystanie.

Teraz przejdziemy do omówienia strategii dla szans:

- **Wykorzystanie** (ang. exploit) – strategia ta sprowadza się do urzeczywistnienia szansy, jaka się pojawia. Dla przykładu, w projekcie potrzebujemy osoby o określonych kompetencjach, a tu przypadkiem okazuje się, że taka osoba przysłała do nas CV i możemy ją zatrudnić do projektu – zauważmy, iż nie jest to tym samym co poszukiwanie pracownika na rynku pracy – wtedy takie nasze działanie miałby charakter wzmocnienia (patrz poniżej).
- **Udostępnienie** (ang. share) – charakter tej strategii jest podobny do przeniesienia w przypadku zagrożenia, otóż szanse przenosimy/udostępniamy innemu podmiotowy, który może ją lepiej wykorzystać, gdyż na przykład w tym zakresie ma większy potencjał od nas. Dobrymi przykładami jest tworzenie konsorcjów czy spółek joint venture na potrzeby realizacji jakiegoś przedsięwzięcia. Oczywiście nie oznacza to, że my musimy rezygnować z profitów, udostępnienia może być związane z umową udziału w zyskach.
- **Wzmocnienie** (ang. enhance) – strategia ta sprowadza się do zwiększenia prawdopodobieństwa wystąpienia szansy lub zwiększenia jej skutków. Dla przykładu by zapewnić oddanie projektu na czas, zatrudniamy więcej osób do realizacji.
- **Akceptacja** (ang. accept) – strategia ta jest postawą bierną. Wiemy, iż szansa jest, czekamy na nią nie podejmując żadnych dodatkowych wysiłków w celu jej urzeczywistnienia.

Monitorowanie ryzyka

Monitorowanie ryzyka, jako element zarządzania ryzykiem, jest procesem ciągłym, który rozpoczyna się w momencie uruchomienia projektu i kończy z momentem jego zakończenia. Poniżej wymieniamy zasadnicze działania związane z monitorowaniem:

- śledzenie i kontrola wszystkich zidentyfikowanych ryzyka,
- wdrażanie działań zapobiegawczych, które wcześniej zostały zaplanowane,
- ocena samego procesu zarządzania ryzykiem,
- obserwacja czy nie pojawiają się nowe, wcześniej nie rozpoznane ryzyka, a jeśli się pojawią, to przeprowadzanie dla nich wcześniej omówionych procesów,
- obserwacja i ocena, czy założenia projektu są wciąż aktualne,
- ocena czy dla wcześniej zidentyfikowanego ryzyka nie zmieniło się prawdopodobieństwo czy też jego skutki (lub w ogóle nie zniknęło).

Proces monitorowania ryzyka często umyka należytej uwadze w zespole, dzieję się tak gdyż mniej doświadczeni menadżerowi niejako „osiadają na laurach" w momencie, gdy zidentyfikowali początkowe ryzyka, poddali je analizie i zaplanowali działanie zapobiegawcze. Brak ciągłej świadomości, iż w projekcie wszystko podlega nieustannym zmianom może być bardzo zgubny. Przykładowo, projekt wystartował 6 miesięcy temu i wszystko było dobrze, do zeszłego miesiąca, kiedy to okazało się, iż tworzona aplikacja przestała czytać dokumenty z zewnętrznego programu. Stało się tak, gdyż zmieniono format pliku eksportu, a mu nie przewidzieliśmy takiego ryzyka.

Podsumowanie

W tym rozdziale przedstawione zostały podstawowe wiadomości z zakresu zarządzaniem ryzykiem w projektach. Stąd też dowiedzieliśmy się, że zarządzanie ryzykiem sprowadza się najczęściej z czterech podstawowych procesów: identyfikacji, analizy, zapobiegania i monitorowania ryzyka. Powiedziane zostało również, jakie procesy wchodzą w skład tych czterech wymienionych działań, przedstawione zostały również niektóre metody wspomagające poszczególne działania.

Na koniec, warto wspomnieć, że zarządzanie ryzykiem wiąże się ściśle z zapewnianiem jakości. Jest to do tego stopnia silny związek, że nie ma mowy o należytym zapewnieniu jakości programu, bez

zarządzania ryzykiem, a z drugiej strony, procesy wchodzące w skład zapewniania jakości często stają się źródłem informacji o nowych ryzykach.

Przykładowe rozwiązanie

Wróćmy do problemu z początku modułu. Jak sądzisz? Co było przyczyną tego, iż zapomnieliśmy jako zespół o Kliencie i jego środowisku? Oczywiście, można w banalny sposób odpowiedzieć, nasz brak doświadczenia czy tez nie udolność, ale przecież nie chodzi tutaj o piętnowanie kogokolwiek. Tak naprawdę zabrakło procedur związanych z ewidencją wymagań niefunkcjonalnych oraz brak planu zarządzania ryzykiem w projekcie. Niestety bardzo częsty błędem niedoświadczonych zespołów jest to, iż skupiają swoja uwagę na tym czego Klient „chce" (wymagania funkcjonalne), podczas, gdy zupełnie zapominają o tym czego Klient „nie chce" jak i „posiada" (wymagania nie funkcjonalne). Aby lepiej wyjaśnić te pojęcie rozważmy następujący przykład w nawiązaniu do problemu, który zasygnalizowaliśmy na początku modułu.

Przykładowy projekt

Klient oczekuje, że stworzymy na jego potrzeby oprogramowanie sieciowe, do ewidencji zamówień jego klientów, wraz z możliwością wysyłania poczty z zamówieniem i przechowywaniem historii operacji wykonywanych w systemie. Z wywiadu przeprowadzonego u Klienta w firmie wynika, iż posiada on:

1. zainstalowana i działająca od kilku lat sieć w firmie, z serwerem na systemie z rodziny Linux,
2. dla potrzeb firmy na serwerze tym została zainstalowana baza danych PostgreSQL, która jest używana do pewnych procesów w firmie (nie jest istotne jakich),
3. stacje klienckie pracujące pod kontrolą systemu Windows XP (5 szt.) oraz Windows 7 (3 szt.),
4. poczta firmowa jak i strona WWW jest obsługiwana na zewnętrznym serwerze.

Zarządzanie ryzykiem w przykładowym projekcie

Identyfikacja ryzyk

Z pierwszej części wynika, że znamy szkielet wymagań funkcjonalnych. Oczywiście opis w postaci jednego zdanie jest zbyt skromny by uznać go za wstępną analizę wymagań, ale nie to jest naszym celem w tej chwili. Skupimy się na drugiej, wypunktowanej części, która stanowi wstęp do analizy wymagań nie funkcjonalnych.

Zanim przejdziemy do dalszej dyskusji, chwila refleksji nad technologia w której my, jako firma, chcemy wykonać zlecenia dla Klienta. Ponieważ doskonale znamy Środowisko VS 2010 i dobrze programujemy w języku C#, a z drugiej strony Klient posiada stacje Klienckie pracujące pod kontrolą systemów rodziny Windows, to naturalny wyborem dla nas jest tworzenie aplikacji w C# i ideałem byłoby, gdybyśmy mogli podłączyć się do bazy MS SQL Server (chcemy oprogramować obsługę bazy w LINQ).

Zauważmy teraz, że to co Klient „posiada", staje się jednocześnie kandydatem do rozpatrywania, w kontekście czego Klient potencjalnie „nie chce". By poprzednie zdanie stało się bardziej jasne, przeanalizujmy kolejny powyższe punkty. Zanim to jednak uczynimy, pamiętajmy, iż wnioskowanie przedstawione poniżej opera się na założeniu, że domniemamy słusznie, zatem zawsze należy spytać Klienta czy się nie mylimy! Oto przykładowe wnioski wysnute z tego co Klient „posiada":

1. Zauważmy iż, z punktu 1 listy rzeczy posiadanych, możemy domniemać, że skoro Klient posiada sieć i jest z niej zadowolony (nie wspomniał o chęci wymiany serwera), to nie będzie chciał na postawienie drugiego serwera na systemie z rodziny Windows lub wymianę istniejącego na Windows. Wynika to choćby z faktu zwiększenia kosztów. Zatem pierwszy wniosek: **Klient nie chce wymieniać posiadanego serwera opartego o system z rodziny Linux**.

2. Z punktu 2 listy rzeczy posiadanych, wynika jasno, że baza PostgreSQL jest u Klienta wykorzystywana produkcyjnie, zatem Klient zdecydowanie nie ma ochoty jej wymieniać. Oprogramowania, które posiada, wykorzystuje ta bazę i przeniesienie na inną jest nie możliwe, gdyż po prostu firma, które je tworzyła już nie istnieje, a kodu źródłowego Klient nie ma. Co prawda, w kontekście poprzedniego punktu i rozważań związanych z wymianą serwera, można rozważać instalacje PostgreSQL na Windows, ale zawsze może to zrodzić problemy i dodatkowe koszty. Wniosek: **Klient zdecydowanie nie chce wymieniać obecnej użytkowanej bazy na inną**.
3. Ponieważ Klient posiada już łącznie 8 komputerów z zakupionym i legalnym oprogramowaniem, to z pewnością będzie miał niechęć do wymiany go na inne. Inna sprawa, że akurat to dla nas to dobra wiadomość, gdyż Klient posiada stacje z systemem operacyjnym dla nas odpowiednim. Wniosek: **Klient nie najprawdopodobniej nie zamierza wymieniać oprogramowania na stacjach roboczych**.
4. Zasadniczo, z naszego punktu widzenia, to gdzie znajduje się serwer pocztowy oraz strona WW ma drugorzędne znaczenie, zatem możemy uznać, że jest to dla nas informacja zbędna.

Jak widać z powyższego, jeśli tylko upewnimy się po stronie Klienta, że nasze przypuszczenia są słuszne, że fakt posiadania danej infrastruktury i rozwiązań implikuje określone ograniczenia. Te ograniczenia są dla nas źródłem ryzyk, zastanówmy się jakich, przy założeniu, że mamy rację co do powyższych stwierdzeń. Zanim jednak wymienimy potencjalne ryzyka, to spróbujmy wymienić wnioski jakie nasuwają się dla nas z powyższych stwierdzeń.

Wniosek 1: Z naszego punktu widzenia, niechęć Klienta do wymiany serwera, oraz niechęć do wymiany bazy danych jest z naszego punktu widzenia tym samym. Sprowadza się, do faktu iż nie możemy użyć bazy MS SQL, a trzeba będzie dostosować się do PostgreSQL. Zauważmy, że z wniosku 1, płynie następująca konkluzja: **ograniczeniem dla naszego systemu jest konieczność wykonania go w oparciu o bazę PostgreSQL**. Co znaczy ograniczenie? Najprościej mówiąc, każde dowolne wymaganie, które ogranicza nasz wybór. Z każdego ograniczenia płynie jakieś ryzyko.

Wniosek 2: Z naszego punktu widzenia, niechęć Klienta do wymiany oprogramowania na stacjach roboczych nie ma większego znaczenia, gdyż stosując Framework .NET uniezależniamy się od platformy (.NET 4.0 jest możliwy do zainstalowania zarówno na Windows XP, jak i Windows 7). Jednak to możemy przekłuć w ryzyko będące szansą: **Klient nie widzi konieczności wymiany oprogramowania na stacjach Klienckich**.

Możemy teraz przejść do identyfikacji ryzyk związanych z tym projektem:

1. **Konieczność użycie bazy, z którą nie mieliśmy do tej pory doświadczenia** – ryzyko to jest zdecydowanie zagrożeniem dla realizacji projektu, które wymaga analizy, oraz dalszych działań. Na chwilę obecną je tylko identyfikujemy.
2. **Klient nie widzi konieczności wymiany oprogramowania na stacjach Klienckich** – ryzyko to, będące ewidentnie szansą, o czym pisaliśmy powyżej, jest czymś bardzo dobrym dla naszego projektu, nie musimy wszak nic w tym zakresie robić.

Analiza ryzyk

Jak wcześniej sygnalizowaliśmy, analiza powinna sprowadzić się do oceny prawdopodobieństwa wystąpienia danego ryzyka, oraz oceny kosztu lub zysku w przypadku jego wystąpienia. Niestety w przypadku obu powyżej zidentyfikowanych ryzyk dość łatwo stwierdzić prawdopodobieństwo wystąpienia, gdyż można je po prostu przyjąć jako pewne. Dużo gorzej z oceną kosztu czy zysku wystąpienia. Niestety w przypadku używania nieznanej technologii, bardzo trudno jest oszacować czas jaki jest potrzebny na jej rozpoznanie i użycie w projekcie. Z drugiej strony jak oszacować, oszczędności (zysk) wypływający z faktu, że nie musimy tworzyć aplikacji np. dla systemu Windows 98. W tej sytuacji należy w miarę możliwości poczytać dostępne badania, czy użyć metody delfickiej pozyskania informacji. Można jednak nie zrobić nic, uznając, że skoro i tak mamy pewność co do faktu wystąpienia tych ryzyk, że nie warto liczyć ich kosztów czy zysków.

Zapobieganie ryzykom

Jeśli posłużymy się określeniami z części teoretycznej w zakresie zapobiegania tym dwu zidentyfikowanym i przeanalizowanym ryzykom, to należy przyjąć, że w przypadku pierwszego z nich (zagrożenia) nie pozostaje nam nic innego jak akceptacja, podczas gdy w przypadku drugiego (szansa) należy zastosować również akceptację, gdyż nie mamy nic innego do zrobienia.

Monitorowanie ryzyka

Oczywistym jest, że należy cały czas monitorować ryzyka, jakie mogą wystąpić w projekcie. Może natomiast pojawić się pytanie, czy warto monitorować dwa wymienione powyżej ryzyka, skoro są one pewne. Odpowiedź jest oczywiście TAK! Kazde ryzyko w projekcie musi być monitorowane, nawet te które są pewne. Wynika to z faktu, że wszystko w projekcie się zmienia. I to, ze dane ryzyko ma prawdopodobieństwo równe 1 w danym momencie nie jest powiedziane, że za 3 tygodnie się to nie zmieni.

Stąd też pamiętajmy o nadrzędnej zasadzie, **monitorujemy wszystkie zidentyfikowane ryzyka i obserwujemy, czy nie pojawiają się nowe**.

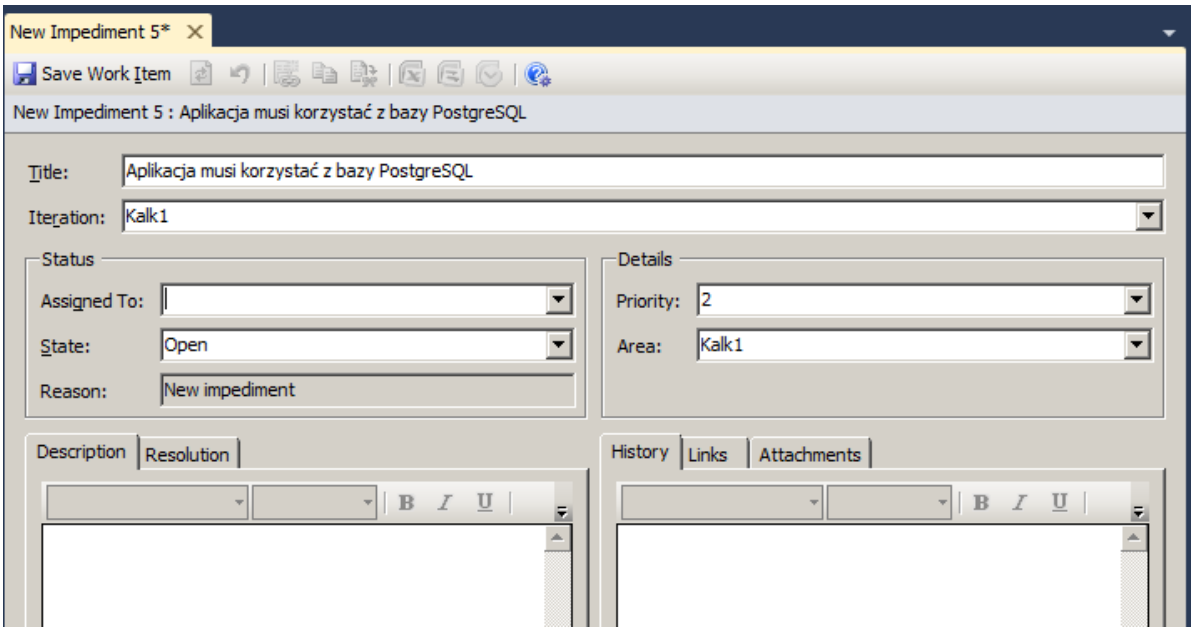
Narzędzia wspomagające zarządzanie ryzykiem

Ja widać z powyższych rozważań teoretycznych oraz zaprezentowanego przykładu, należy ewidencjonować wszelkie ryzyka jakie mogą pojawić się w projekcie. Tu oczywiście pojawi się naturalne pytanie, czy istniej narzędzi o tego dostosowane? Niestety odpowiedź nie jest jednoznaczna. Z jednej strony tak, istnieje doskonałe narzędzie, które pozwala ewidencjonować wszelkie zdarzenia (Excel), ale z drugiej nie jest to narzędzie ściśle związane ze środowiskiem programistycznym, zatem to co napiszemy w Excelu nie przenosi się wprost do środowiska TFS, a o to nam chodzi.

Na szczęście, cześć ryzyk będziemy mogli ewidencjonować w ramach rodziny Vs2010+TFS2010 oraz szablony Visual Stduio Scrum 1.0. Dzięki temu będziemy mogli mieć ich ewidencje w projekcie, a ponieważ , jak się za moment okaże, będą one elementami typu Work Items, to będziemy mieli dostępne dla nich wszystkie narzędzia jak dla zadań czy błędów – pokażemy to zaraz na przykładzie.

Zacznijmy wpierw od ewidencji ograniczenia w ramach środowiska w którym pracujemy. Aby zapisać ograniczenie (Inpediment) korzystamy z analogicznego sposobu jak w przypadku dodawania innych elementów typu Work Items.

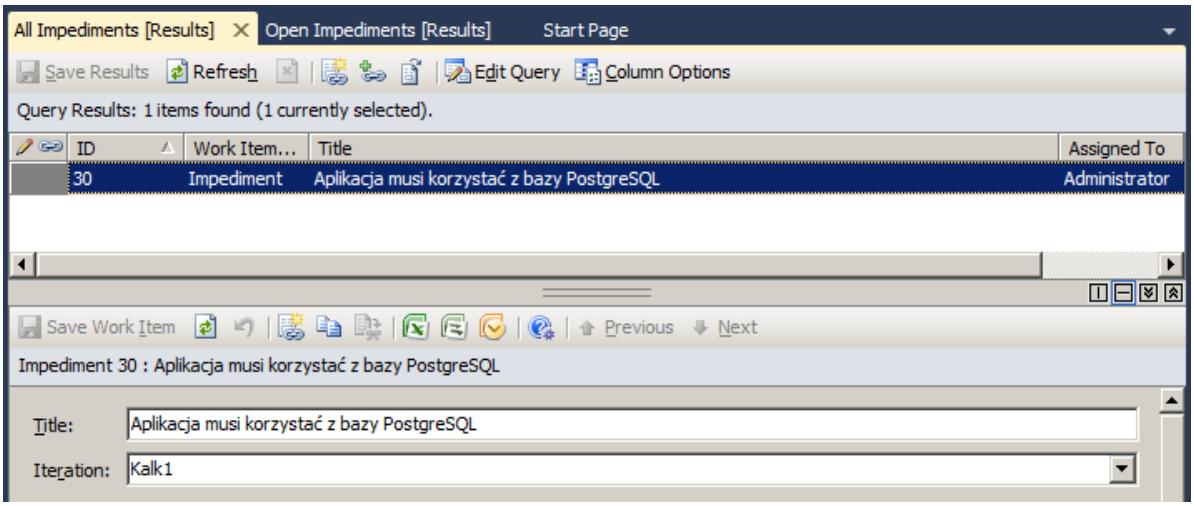
1. W oknie Team Explorer odszukujemy węzeł Work Items, klikamy na nim prawym klawiszem i z menu wybieramy New Work Item, a następnie Inpediment, pojawi się okno jak poniżej (Rysunek 5).
2. W powyższym oknie wpisujemy/wybieramy odpowiednio w polach (porównaj jak to było w przypadku pracy czy zadania):
   a) **Title** – tytuł ograniczenia. W naszym przypadku jak powyżej.
   b) **Iteration** – iteracja której to ograniczenie dotyczy. Pozostawmy na razie bez zmian.
   c) **Assigned To** – osoba odpowiedzialna. Wybierzmy preferowaną osobę.
   d) **State** – stan ograniczenia, przy czym rozróżniamy dwa: **Open** oraz **Closed**. Pozostawiamy Open.
   e) **Priority** – priorytet tego ograniczenia. Dopuszczalne wartości to 1,2,3,4, przy czym 1 oznacza najwyższy priorytet. Wybierzmy 1.
   f) **Area** – obszar. Pozostawmy bez zmien.
   g) **Description**, **History**, **Links**, **Attachements** – mają analogiczne znaczenie jak w przypadku prac i zadań (porównaj moduł 5).
   h) **Resolution** – docelowo ma zawierać informację w jaki sposób zespół prowadził sobie z tym ograniczeniem.
3. Po wpisaniu lub wybraniu odpowiednich wartości naciskamy przycisk Save Work Item.



Rysunek 5. Okno dialogowe pozwalające dodać nowe ograniczenie.

Widać wyraźnie, że wszystkie informacje, które wpisujemy w ramach definiowania ograniczenia, choć są przydatne, to nie odzwierciedlają przedstawionej teorii. W szczególności, nie możemy wpisać ani prawdopodobieństwa, ani też kosztu w przypadku gdy to ryzyko wystąpi. Oczywiście, to co umożliwia VS2010+TFS2010 jest uproszczoną formą ewidencji, jednak to nie jest bardzo duży problem, gdyż jak to było widać na macierzy ryzyk (Rysunek 2.). Zatem wystarczy zdefiniować odpowiednią dla nas macierz ryzyk, taką, by ostatecznie otrzymać skalę 1..4 i dzięki temu możemy już posługiwać się powyższym mechanizmem (patrz ćwiczenie).

Po zapisaniu ograniczenia, chcemy przejrzeć listę wszystkich ryzyk w projekcie i okazuje się, że nie mamy takiego zapytania predefiniowanego. Owszem mamy możliwość przejrzenia wszystkich ograniczeń danego sprintu (spójrz, że jest takie pytanie w węźle Current Sprint), ale ponieważ nasze ograniczenie miało w polu Iteration tylko Kalk1, to nie należy do bieżącego, ani też innego sprintu, więc go nie widać. Zatem, by móc zobaczyć listę wszystkich ograniczeń musimy zdefiniować nowe zapytanie. Klikamy prawym klawiszem w węzeł My Queries i menu kontekstowego wybieramy New Query. W oknie New Query wiersz zawierający Team Project pozostawiamy bez zmian, typ Work Item ograniczamy tylko do Inpediment, a stan pozostawiamy dowolny, zapisujemy nasze pytanie. Po zapisaniu może my już je wykonać, jeśli nie popełniliśmy błędu, to zobaczymy wynik podobny do zaprezentowanego na poniższym rysunku (Rysunek 6.).



Rysunek 6. Wynik wykonania zapytania wybierającego wszystkie ograniczenia w projekcie.

Przykładowa analiza ryzyk dla projektu Kalkulatora

W poprzednim przykładzie może wydać się dziwne, że dodaliśmy to ograniczenie do projektu Kalkulatora, dla którego ono nie ma sensu, ale nie chcieliśmy tracić czasu dla tworzenia nowego projektu. Zatem teraz, by nie zaśmiecać projektu Kalkulatora, usuńmy to niepotrzebne ograniczenie i tu spotyka nas zaskoczenie … nie możemy znaleźć opcji usunięcia dla ograniczenia, jak się okaże również dla pozostałych elementów typu Work Item. Jedyne co możemy zrobić to zamknąć o ograniczenie. Pamiętajmy o tym, że wszystko co wpiszemy do naszego projektu jest przechowywane „na wieczność". Stąd też unikajmy wpisywanie niepotrzebnych rzeczy, czy tylko czegoś na próbę. Im większa czytelność projektu, tym mniejsze ryzyko popełnienia przypadkowego błędu. Stąd zdecydowanie do takich eksperymentów warto założyć jakiś testowy projekt. Na szczęście projekt Kalkulatora spełnia u nas taką funkcję, stąd nie jest to wielki problem.

Dobrze, teraz możemy zastanowić się jakie ryzyka mogą się zdarzyć w przypadku naszego, wydawałoby się banalnego projektu, Kalkulatora. Przy czym, tym razem zastanówmy się nad tym w kontekście prac już wypisanych i zapisanych najpierw w Wykazie prac produktu, a obecnie w Wykazie prac sprintu. Przypomnijmy jakie to są prace oraz zadania:

1. Szkielet kalkulatora (praca)
   a) Wykonanie głównego okna aplikacji (zadanie)
2. Ustalić z Klientem wygląd przycisków (praca)
   a) Ustalenie wyglądu przycisków (zadanie)

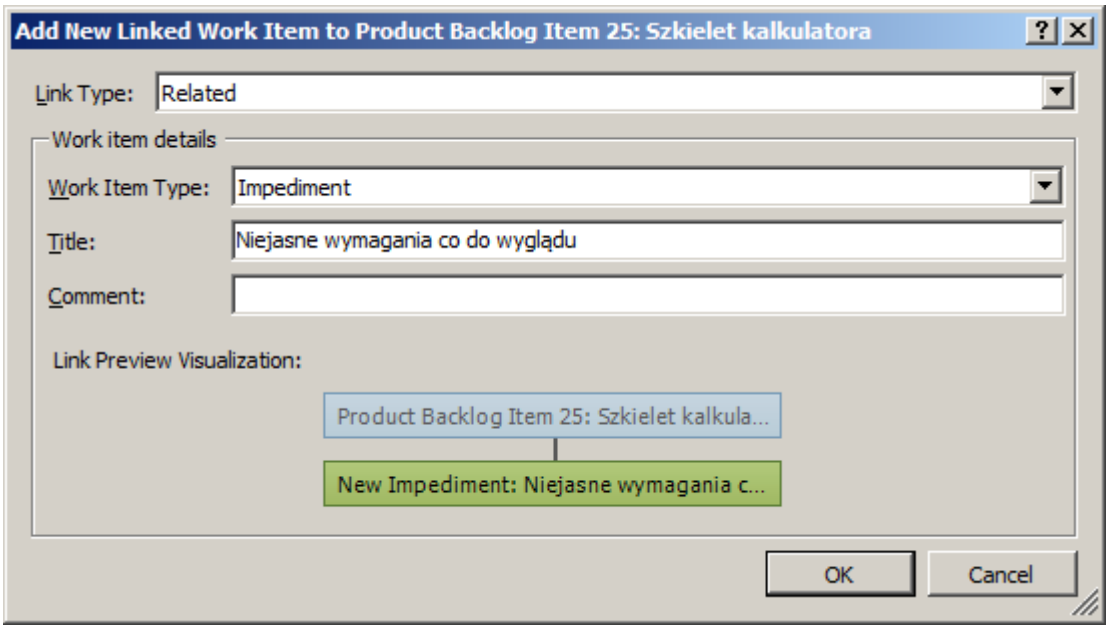
Zauważmy, że w poprzednich modułach już popełniliśmy błąd polegającym na tym, iż zapomnieliśmy wpisać kryteria akceptacji dla pracy „Szkielet kalkulatora". Pytanie jest, czy ten fakt jest ryzykiem w naszym projekcie? Nie, gdyż w rejestrze ryzyk dla naszego projektu wyszczególniamy te które przynależą do niego, jakby bezpośrednią przyczyną ich powstania jest ten projekt. Podczas, gdy fakt naszej nieudolności, czy braku doświadczenia jest ryzykiem firmy jako takiej i nie możemy dopisywać go do każdego projektu, w przeciwnym razie będziemy mieli listę wszystkich możliwych ryzyk, włączając w to wyłączenia prądu i zalania biura, a to z kolei spowoduje brak czytelności i rozproszenie na rzeczy nie będące w bezpośrednim kręgu zainteresowań projektu.

W takim razie czy istnieją ryzyka w tym projekcie, bezpośrednio należące do niego. W świetle tego, co napisaliśmy w części teoretycznej, oczywiście tak, najwyżej nie umiemy je zauważyć. Zastanówmy się zatem jakie ryzyko może ze sobą nieść pierwsza praca „Szkielet kalkulatora", otóż największym zagrożeniem związanym z tą praca, jest złe zrozumienie wymagań. Może wydawać się dziwne, dlaczego tak może się stać, przecież kolega dokładnie nam powiedział czego chce?

Zastanówmy się, czy na pewno kolega jest tutaj końcowym odbiorcą? Oczywiście nie! Końcowym odbiorcą jest jego nauczyciel, który polecił wykonać ten program w ramach zajęć z programowania. Zatem jaką mamy gwarancję, że kolega prawidłowo zrozumiał polecenia, oraz dobrze je nam przekazał? Niestety pewności nie mamy. Zidentyfikowaliśmy pierwsze ryzyko: **Niejasne wymagania co do wyglądu**. Teraz musimy ocenić prawdopodobieństwo oraz koszty jeśli to ryzyko wystąpi. W przypadku prawdopodobieństwa, trzeba coś założyć w oparciu o wiedze o solidności czy rzetelności kolegi. Koszt należy oszacować na podstawie ilości godzin jakie potencjalnie spędzimy na przerabianiu projektu. Pozostaje pytanie jak zapobiegać temu ryzyku? Najprostszy sposób to poprosić o wymagania od prowadzącego na piśmie. Monitorowanie sprowadzi się, do obserwacji czy cos w trakcie realizacji nie dzieje się dziwnego z wymaganiami.

Powyższa analiza może wydawać się dziwna, gdyż pozornie układ nauczyciel-uczeń jest nietypowy w realizacji projektów komercyjnych. To prawda, ale niestety dokładnie taki sam problem wystąpi w tych komercyjnych zleceniach, a wynika on z bardzo prostej rzeczy – mianowicie w korporacji czy firmie, osoba, która zleca wykonanie projektu bardzo rzadko jest końcowym użytkownikiem, będzie to np. Dyrektor IT, podczas gdy końcowymi użytkownikami będą pracownicy działu księgowości. W tej sytuacji niejasność czy niepełność przekazu wymagań będzie jeszcze większa niż w zademonstrowanym przykładzie. Dlatego też już w module 2 pisaliśmy, że zbierając wymagania należy domagać się rozmów z jak największa ilości tzw. kluczowych użytkowników systemu.

Na dobrze, mamy zidentyfikowane ryzyko, chcemy je wprowadzić do systemu. W tej sytuacji, gdy ryzyko przynależy do jednej konkretnej pracy dobrze byłoby skojarzyć je właśnie z nią. Zatem otwórzmy Wykaz prac sprintu. Pojawią się tam prace i zadania. Klikamy prawym klawiszem myszy, gdy kursor znajduje się nad pracą „Szkielet kalkulatora”. Następnie z menu wybieramy pozycję New Linked Work Item (skrót Shift+Alt+L), pojawi się okno znane już z wcześniejszych przykładów (Rysunek 7.).



Rysunek 7. Dodawanie ograniczenia powiązanego z pracą, którą realizujemy w bieżącym sprincie.

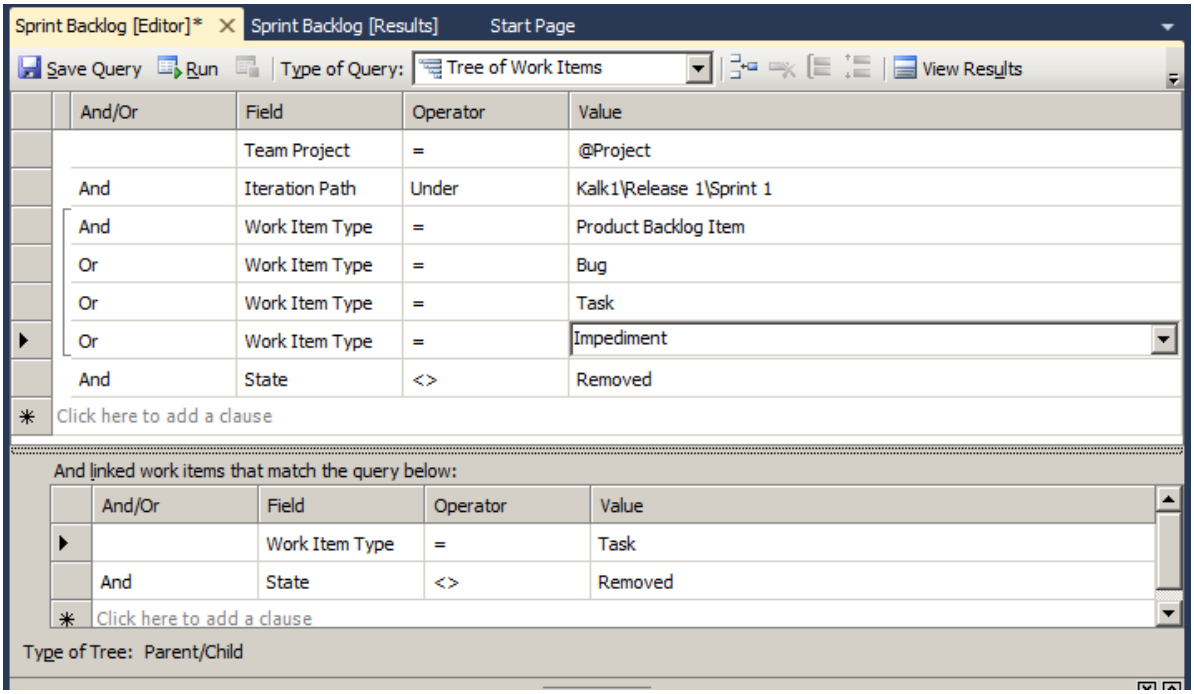
Ale tym razem zatrzymajmy się chwilę i rozwińmy listę Link Type. Do tej pory wszystkie elementy podłączaliśmy jako Child, ale czy w tym wypadku jest to właściwe? Zobaczmy ile mamy tam relacji, do tej pory wykorzystywaliśmy jedną, relację rodzic-dziecko, za jej pomocą można łatwo zobrazować, zależność, gdy jedna rzecz podlega pod drugą. Ale czy nasze ograniczenie ma charakter dziecka w stosunku do wymagania? Zdecydowanie nie, również nie można byłoby powiedzie, że jest jego rodzicem. Ale zauważmy, że istnieje na liście Link Typer relacja, która nosi nazwę Related i oznacza ona powiązanie elementów, ale bez ścisłej hierarchii pomiędzy nimi.

Wybierzmy zatem ten typ relacji i pozostałe elementy jak rysunku 7. Naciskamy OK i pojawia się okno jak na rysunku 5. Wypełniamy pozostałe potrzebne pola i naciskamy Save Work Item – dodaliśmy ograniczenie związane z konkretną pracą.

O monitorowaniu raz jeszcze

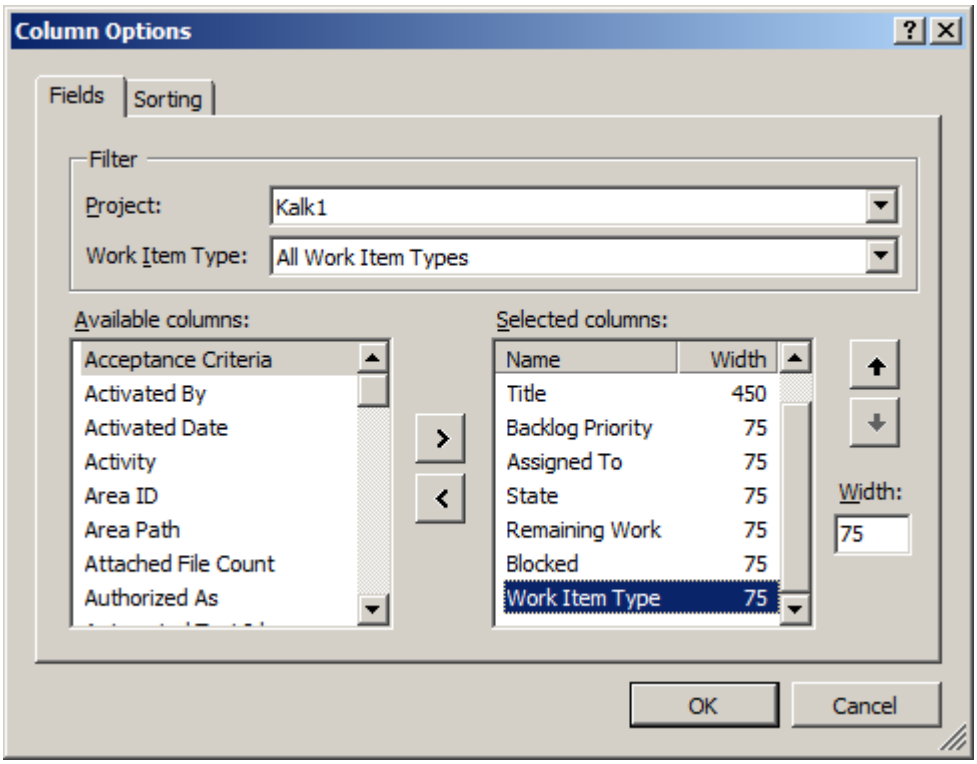
Jak wiemy z teorii im więcej ryzyk zidentyfikujemy tym lepiej dla projektu. Jednak jak praktyka pokazuje im jest ich więcej, tym trudniej nimi zarządzać. Stąd też monitorowanie jako proces również musi być wspomagana jakimiś narzędziami, by nie było problemu z zagubieniem jakiegoś ryzyka, czy pominięciem jego kontroli.

Wcześniej pokazaliśmy, jak zdefiniować własne zapytanie, które pozwoli wyświetlić wszystkie ograniczenia (Rysunek 6.), jednak zauważmy, że Szef Scrum zwykle będzie śledził Sprint Backlog. Im mniej „okienek" będzie musiał obserwować, ty lepiej. Ale gdy zajrzymy do wykazu prac sprintu, to okaże się, że on nie zawiera informacji o ograniczeniach, pomimo faktu, iż dodane przez nas ograniczenie powiązaliśmy z pracą. Wszystko się wyjaśni, gdy zajrzymy do edycji zapytania Sprint Backlog (prawy klawisz myszy na tym zapytaniu i z menu wybieramy Edit Query). Jak widać, zapytanie to pozwala wybrać zarówno prace, zadania i błędy, ale nie ma tu ograniczeń. W takim razie zmodyfikujmy to pytanie i zapiszmy (Rysunek 8.).

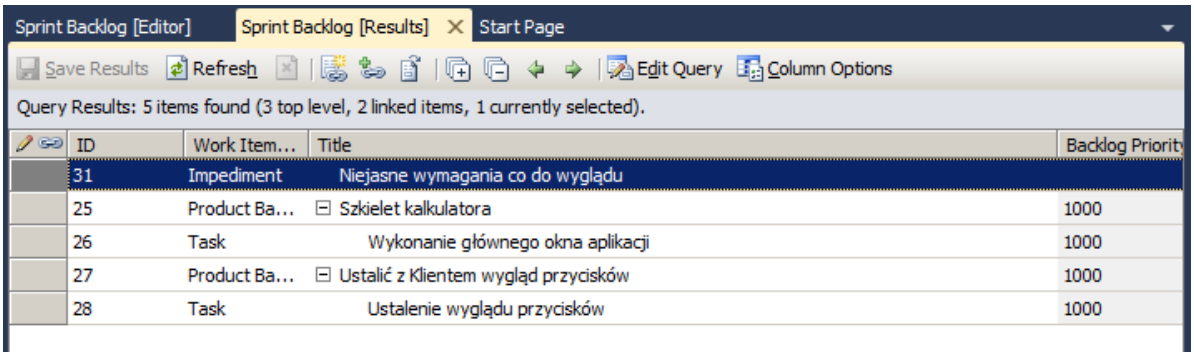


Rysunek 8. Zapytanie wykazu prac sprintu zmodyfikowane tak by pokazywało również ograniczenia.

Wykonajmy teraz to zapytanie, jak widać, na liście wyników znalazło się również nasze ograniczenie, z tym, że straciliśmy trochę na czytelności, gdyż to ograniczenie występuje na tym samym poziomie zagłębienia, co prace i można przez to się zgubić. Stad też przydałaby się przy nazwie kolumna informująca o typie Work Item każdego z elementów na liście. Kliknijmy zatem na górnym pasku w przycisk Column Options, w wyniku tego pojawi się okno dialogowe pozwalające na dodatkowe operacji (Rysunek 9.). W oknie tym mamy możliwość wyboru projektu, którego te operacje dotyczą (lista Project), listy filtrów elementów, które mają być wyświetlane (lista Work Item Type), listę wszystkich dostępnych kolumn (Available columns), listę kolumn aktywnych w tym momencie (Selected columns) i długość pola. W liście aktywnych kolumn odnajdźmy kolumnę Work Item Type a następnie znajdującą się obok strzałką przesuńmy ja na oczekiwaną pozycję, na koniec naciśnijmy OK. Jeśli wszystko się powiodło to możemy obejrzeć rezultat podobny do rysunku 10.



Rysunek 9. Okno opcji dla kolumn wyświetlanych w wykazie prac sprintu.



Rysunek 10. Wynik działania zapytania Sprint Backlog z dodanymi ograniczeniami i przesuniętą kolumną typu Work Item.

Porady praktyczne

Podobnie jak w przypadku zapewniania jakości, tak i w przypadku zarządzania ryzykiem należy podkreślić, że tego procesu nie wolno bagatelizować. Niestety jak każdy proces, który występuje w realizacji projektu, tak i ten ma swój koszt. W większych firmach czy zespołach wyznacza się tylko osobne role, ale czasem wręcz grupy osób, które specjalizują się w zarządzaniu ryzykiem i jego monitorowaniem w kilku projektach. Jednak nawet gdy jesteśmy niewielką firmą, i realizujemy niewielkie projektu, to starajmy się nie podchodzić nonszalancko do zarządzania ryzykiem. Proces ten choć żmudny, a czasem i nurzący, przekłada się w bezpośredni sposób na komfort pracy i jakość produktu finalnego.

Na koniec warto zaznaczyć, że z początku bardzo duży problem może nam sprawiać zarówno identyfikacja czy analiza, wynika to głównie z braku doświadczenia. W tej sytuacji można posłużyć się opublikowanymi listami kontrolnymi lub tez można zatrudnić osobę z doświadczeniem na zlecenie w naszym projekcie. Niezależnie od wyboru drogi, nie bójmy się w ramach zespołu powiedzieć, że nie umiemy czegoś wykonać, być może wspólnie znajdziemy rozwiązanie problemów które się pojawiają.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- wiesz co rozumiemy pod pojęciem ryzyka,
- wiesz jakie procesy wchodzą w zakres zarządzanie ryzykiem,
- potrafisz przeprowadzić identyfikację ryzyk oraz ich analizę,
- potrafisz stworzyć prosty rejestr ryzyk, oraz zapisać niektóre z nim w ramach środowiska VS 2010,
- rozumiesz rolę zarządzania ryzykiem w projekcie i powiązanie z procesem zapewniania jakości.

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. PMI – A Guide to the Project Management Body of Knowledge (PMBOK GUIDE)

   Książka ta omawia metodykę PMI.

2. PRINCE2™ – Skuteczne Zarządzanie Projektami, Drugie wydanie polskie Crown Copyright 2010

   Książka ta omawia metodykę PRINCE2™.

Laboratorium podstawowe

Zadanie 1

W sekcji Przykładowe rozwiązanie sygnalizowaliśmy, że w przypadku dodawania ograniczenia (Impediment) mamy możliwość określenia tylko i wyłącznie priorytetu wykonania. Nie mamy zaś możliwości, zapamiętania prawdopodobieństwa ani skutków wystąpienia. Spójrz ponownie w część teoretyczną, przyjrzyj się macierzy z rysunku 2, w niej wartości w środku są niczym innym jak wyliczonymi „priorytetami", a dokładniej wartościami, które poprzez dyskretyzację zostały zamienione na priorytety (stopnie szarości w tabeli).

Jak to widzieliśmy w przykładach powyżej ograniczenia w ramach szablonu Visual Studio Scrum 1.0 mogą mieć cztery wartości priorytetu z zakresu 1..4.

Twoim zadaniem jest opracowanie macierzy prawdopodobieństwa i skutków ryzyk (tylko dla zagrożeń), która pozwalałaby na dyskretyzację nie na trzy wartości priorytetu (rysunek 2), a na cztery, czego od nas oczekuje omawiany szablon.

Zadanie 2

Stwórz przykładowy arkusz kalkulacyjny, w dowolnym programie do tego służącym, który pozowali na notowanie ryzyk projektowych. Pamiętaj, że musisz dla każdego ryzyka przechowywać co najmniej następujące informacje:

1. Nazwę
2. Prawdopodobieństwo
3. Skutek (koszt/zysk, o ile możliwy do wyliczenia, to najlepiej wartość liczbowa).
4. Właściciela ryzyka (osobę za nie odpowiadającą).
5. Data wygaśnięcia (data po której ryzyko przestaje istnieć, może być to koniec projektu).
6. Proponowany typ reakcji na to ryzyko.

Po stworzeniu tego arkusza wróć pamięcią do Laboratorium podstawowego z modułu 2, może masz notatki? Dla projektów, które wtedy były analizowane przeprowadź identyfikacje, analizę oraz zapobieganie ryzyk tam występujących.

Laboratorium rozszerzone

Zadanie 1

W oparciu o arkusz sporządzony w Zadaniu 2 w Laboratorium podstawowym niniejszego modułu, sporządź wykaz ryzyk dla projektu z zadania 1 w Laboratorium rozszerzonym w module 2.

Zadanie 2

W oparciu o arkusz sporządzony w Zadaniu 2 w Laboratorium podstawowym niniejszego modułu, sporządź wykaz ryzyk dla projektu z zadania 2 w Laboratorium rozszerzonym w module 2.

ITA-111 Programowanie zespołowe

Ścibór Sobieski

Moduł 8

Wersja 1

Testy i proces testowania

Spis treści

Informacje o module

Opis modułu

Celem niniejszego modułu jest zapoznanie z zagadnieniami związanymi z testami i procesem testowania. Stąd też zostaną tu zaprezentowane pojęcia błędu oraz pochodnych, testów i procesu testowania. Wyjaśniona zostanie różnica pomiędzy walidacją, a weryfikacją, oraz podziałem koncepcyjnym z tego wynikającym. Wymienione też będą podstawowe sposoby testowania jak i pojęcia związana z tym procesem (precyzja i jakość). Następnie zostanie wskazane różne sposoby testowania oraz ich implementacja w ramach środowisk Visual Studio 2010, Team Foundation Studio 2010 oraz Microsoft Test Manager.

Cel modułu

Celem modułu jest przedstawienie ogólnych zagadnień związanych z błędami, testami je wykrywającymi i procesem testowania.

Uzyskane kompetencje

Po zrealizowaniu modułu będziesz:

- wiedział czym różnią się pojęcia błędu, problemu, usterki,
- wiedział jaka jest rola testów oraz procesu testowania,
- potrafił wprowadzić do środowiska przypadek testowy, oraz test jednostkowy,
- rozumiał role procesu testowania oraz wpływ tego procesu na zapewnienie jakości w projekcie.

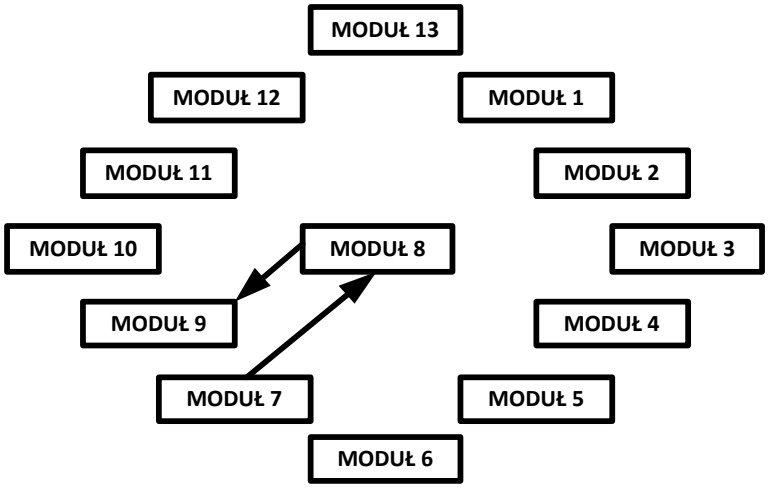
Wymagania wstępne

Przed przystąpieniem do pracy z tym modułem powinieneś:

- znać podstawy obsługi środowiska Visual Studio 2010 oraz Team Foundation Studio 2010,
- znać podstawy pracy z szablonem Visual Studio Scrum 1.0,
- znać podstawy tworzenia aplikacji Windows (preferowany WPF) w środowisku Visual Studio 2010 przy użyciu języka C#,
- rozumieć czym jest cykl wytwarzania oprogramowania i jakie procesy w nim występują,
- rozumieć czym jest proces zapewniania jakości.

Mapa zależności modułu

Zgodnie z mapą zależności przedstawioną na Rys. 1, przed przystąpieniem do realizacji tego modułu należy zapoznać się z materiałem zawartym w module



Rys. 1 Mapa zależności modułu

Przygotowanie teoretyczne

Przykładowy problem

Poprzednie Wasze doświadczenia pozwoliły Wam na zdobycie podstawowej wiedzy o rodzinie produktów VS 2010 i TFS 2010 oraz o tym, jak używając tych narzędzi i szablon Visual Studio Scrum 1.0, rozpocząć efektywną prace nad projektem w oparciu o metodykę Scrum. Wiecie już co rozumiemy pod pojęciem jakości, oraz znacie podstawowe fakty za zakresu zarządzania ryzykiem. Jednak już kilkukrotnie sygnalizowany był problem testowania aplikacji. Zaczynacie się zastanawiać czy to naprawdę, aż tak ważne? Co prawda w trakcie różnych przedmiotów z zakresu inżynierii oprogramowania wspominano o metodykach w mniejszym lub większym stopniu opartych o testy (jak np. TDD czy XP).

Z jednej strony zadajecie sobie pytanie, czy testy są aż tak istotne? Przecież to oczywiste, iż każdy programista, który pisze aplikacje sprawdza czy się kompiluje czy tez uruchamia. Skoro tak, to znaczy, że program jest wstępnie przetestowany, a resztę to przecież Klient sobie sprawdzi To on zlecał, więc on najlepiej wie, czy program działa poprawnie czy też nie. Z drugiej jednak pamiętacie, że przestrzegano Was, by absolutnie nie bagatelizować procesu testowania i raczej poświęcić mu więcej czasu niż mniej.

Ostatecznie postanawiacie, że warto spróbować sobie przypomnieć przynajmniej ogólny zarys wiedzy o testach i procesie testowania. Spróbujmy nakreślić najważniejsze fakty i pokazać, oraz to jak do tego procesu można wykorzystać poznane już narzędzia.

Podstawy teoretyczne

W module tym będziemy mówić o procesie testowania, jak i różnego rodzajach testów. Jednak zanim do tego przejdziemy musimy odnieść się do przyczyn dla których testujemy nasz produkt. Oczywiście intuicyjnie rozumiemy, iż proces testowania ma za zadanie wykryć różnego rodzaju nieprawidłowości w działaniu, który wynikają z błędów w procesie produkcji. Ale zauważmy, iż już w tym stwierdzeniu odwołujemy się do kolejnych pojęć, które wymagają rozumienia intuicyjnego, bo co to jest błąd? Albo czym jest nieprawidłowość w działaniu? Zdecydowanie wymaga to od nas chwili refleksji nad tymi pojęciami.

Zanim jednak do tego przejdziemy do definiowania pojęć związanych z błędami, należy podkreślić, że w ramach procesu testowania wyróżnia się dwa zasadnicze obszary, które różnią się co do sposoby testowania. A podział ten wyraża się w literaturze określeniami walidacja i weryfikacja oprogramowania. Niestety istniej dużo nieporozumień co do ich rozumienia, aby stały się one jaśniejsze, zacznijmy od bardzo ładnej, opisowej, quasidefinicji, którą podał Bohem w 1979 r., otóż:

- **Validation**: Are we building the right product?
- **Verification**: Are we building the product right?

Specjalnie pozostawiliśmy oryginalną pisownię, gdy zawiera ona nieprzetłumaczalną grę słów, która to właśnie doskonale oddaje różnicę pomiędzy tymi procesami. Wynika z niej jasno, że weryfikacja jest procesem (grupą procesów), który utwierdza nas, ze zbudowaliśmy produkt właściwy, a zatem zgodny ze specyfikacją wymagań Klienta. Podczas, gdy weryfikacja pomaga stwierdzić, na ile należycie wyprodukowaliśmy to oprogramowania, czyli czy na przykład program się nie zawiesza.

Zauważmy na koniec, że weryfikacja i walidacja nie są czymś co wynika jedno z drugiego, a są to rzeczy, które się uzupełniają. Dla przykładu, bez problemu możemy wyobrazić sobie pogram, który spełnia oczekiwania klienta w zakresie funkcji, ale od czasu do czasu się „zawiesza”. Z drugiej strony program może działać bezbłędnie, ale tylko dlatego, że jest aplikacją posiadającą jedynie menu, bez logiki aplikacji, a jako taki z pewnością nie będzie spełniał oczekiwań Klienta.

Co to jest błąd i do czego służy test?

Ponieważ w literaturze jak i praktyce można znaleźć tyle definicji błędów ilu autorów, to postaramy się wytłumaczyć, i w miarę możliwości uściślić, czym jest błąd w oparciu o opis sytuacji, która są przyczyną problemów mogących skutkować błędami. Wpierw jednak należy podkreślić, że bardzo często spotyka się różne określenia na problem, który wystąpił, w zależności od sytuacji lub firmy, która stosuje dane nazewnictwo, możemy spotkać się z określeniami, błąd, problem, defekt, pomyłka, nieprawidłowość, awaria, i wiele innych. Dla przykładu, brytyjski standard BS 7295-1 nadaje następujące znaczenia dla poniższych określeń:

- **Error** – jest interpretowany jako pomyłka, której przyczyną jest błąd w kodzie programu, ale wynikający z winy człowieka (programisty).
- **Fault** – dla odmiany służy do identyfikowania błędu jako takiego np. źle napisanej funkcji.
- **Failure** – jest niejako skutkiem, zatem będzie to błędny wynik, którego przyczyny mogą być różne (np. powyższe).

Nie będziemy tutaj toczyć dyskusji nt. tego, czy jest to dobry podział, czy nie, ale już sam ten przykład i to pochodzący ze standardu, pokazuje, ze nie ma tu dobrych i uniwersalnych definicji.

Dalej będziemy starali się po prostu mówić o błędzie, który z punktu widzenia Klienta powoduje problem z oprogramowaniem przez nas stworzonym. Oto najczęstsze sytuacje, w których Klient informuje, że wystąpił problem czy błąd (w nawiasie podano obszar, którego ten problem dotyczy):

1. Program nie wykonuje funkcjonalności, która jest zawarta w wymaganiach (walidacja).
2. Program posiada funkcje, których nie ma w wymaganiach (walidacja).
3. Program wykonuje funkcjonalność, która występuje w wymaganiach, ale nie tak jak oczekuje tego Klient (walidacja).
4. Program nie spełnia kryteriów nie funkcjonalnych np. działa zbyt wolno (walidacja).
5. Program generuje błędne wyniki (weryfikacja).
6. Program się „zawiesza” (weryfikacja).

Należy pamiętać, że niestety każdy Klient może mieć różne oczekiwania co do tego stwierdzenia „program ma działać dobrze”. Dlatego tak ważne jest dobre wyspecyfikowanie oczekiwań. Jak pokazują badania, najwięcej błędów powstaje na poziomie specyfikacji, potem projektu a dopiero na końcu programowania. Zatem warto precyzyjnie określać specyfikację i upraszczać projekt. Oczywiste jest również, że im później wykryjemy błąd, tym koszt jego usunięcia będzie większy. To jest główny problem modelu kaskadowego wytwarzania oprogramowani i jeden z naczelnych argumentów za metodykami lekkimi, które iteracyjnie podchodzą do wytwarzania i pozyskiwania wymagań. Dzięki czemu znaczenie wcześniej widać efekty pracy i ponosimy mniejszy koszt w przypadku konieczności zmiany.

Ostatecznie możemy spróbować powiedzieć o co chodzi w testach? Opierając się na powyższych rozważaniach możemy powiedzieć, że:

1. Celem testów jest wykrycie błędów.
2. Celem testów jest znajdowanie błędów jak najwcześniej.

Jakość kodu vs. jakość programu

Jak sygnalizowaliśmy już wcześniej (moduł 6 – zapewnianie jakości), znaczna część jakości ostatecznego produktu w projekcie informatycznym zależy od jakości kodu programu. I choć mówiliśmy też, że to nie wystarcza, to jest to na tyle ważne, że obecnie przykłada się ogromną rangę do procesu testowania. Zajmiemy się tym jak przeprowadzać oraz usprawnić testowanie, którego celem będzie podnoszenie prawdopodobieństwa niezawodności kodu, a w konsekwencji zwiększenie jakości programu.

Wiemy już, iż nawet w sytuacji gdy stosujemy standardy kodowania oraz dobrze zarządzamy projektem, nie wynika, że program będzie działał poprawnie. Stąd też potrzebne są dodatkowe

czynności pozwalające wykryć nieprawidłowości działania programu, za zatem wykryć błędy w kodzie programu. Wymienialiśmy w module 6, proste przykłady typów błędów jakie mogą pojawić się w programie, takie jak: błędy implementacyjne, błędy przepełnienia bufora (prawdę powiedziawszy tez można je rozumieć jako błędy implementacyjne), błędy złej interpretacji danych. Teraz przyszedł czas na bardziej szczegółowe przyjrzenie się błędom i testom pozwalającym je wykryć.

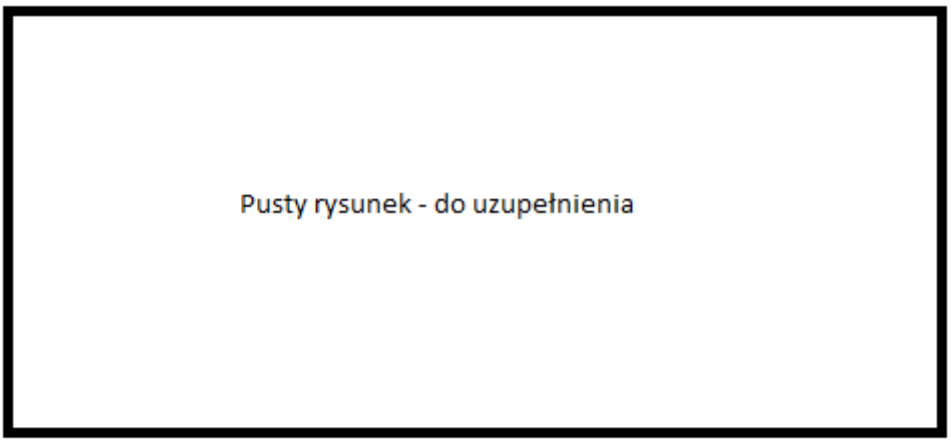
Testy i proces testowania

Proces testowania jak każda inna czynność posiada swoje ograniczenia, w przypadku testów często nazywa się je aksjomatami testowania, które należy znać i pamiętać w trakcie przeprowadzania tego procesu.

Aksjomaty testowania

Poniższa lista ma charakter wyboru najważniejszych ze spotkanych w literaturze, nie zamierzamy wymieniać tu wszystkich jakie można spotkać.

1. **Żadnego realnego oprogramowanie nie da się przetestować całkowicie**. Piszemy tu specjalnie, realnego, gdyż można wymyślić banalny program, który dałoby się przetestować. Co to znaczy, że programu nie da się przetestować całkowicie? Oznacza to tyle, że niezależnie od ilości przeprowadzonych testów oraz ich jakości, nigdy nie mamy pewności, czy programie nie zawiera jeszcze jakiegoś błędu. Czyli mówiąc inaczej, **test nie udowadnia braku błędów, a udowadnia jedynie to, ze ich nie znaleźliśmy**.
2. **Testy są czasochłonne**. Jak każdy inny proces, tak i testy zajmują czas, stąd planując je w cyklu wytwórczym danego oprogramowania należy uwzględnić w harmonogramie czas na ich wykonanie.
3. **Testowanie jest kosztowne**. Jak wiemy, każda czynność ma swój koszt, zatem i testy, stąd też z jednej strony warto testować jak najwięcej, ale drugiej strony w pewnym momencie przestaje się to opłacać.
4. **Testowanie jest ryzykowne**. Wybierając ten, a nie inny sposób testowania, sporządzając taką a nie inna listę testów, podejmujemy decyzje obarczone ryzykiem. Ponieważ jak pisaliśmy nie ma możliwości przeprowadzenie wszystkich testów, ze względu na czas i koszty, więc zawsze musimy dokonać pewnego wyboru, a ten może okazać się gorszy, od innego, zatem ryzykujemy. Spójrzmy na rysunek 2, który stanowi ciekawą ilustrację zależności, ilość, kosztu, oraz pośrednio czasu realizacji testów. Widać na nim zilustrowaną obrazowo regułę, że dla każdej aplikacji istnieje optymalna liczba testów. Niestety wyznaczenie tej liczy jest trudne.



Rysunek 2. Ilustracja kosztu i ilości testów oraz optymalnej liczby testów.

5. **Paradoks pestycydów**. To oryginalne i jednocześnie ciekawe określenie użył po raz pierwszy Boris Beizer w swojej książce pt. Software Testing Techniques. Autor użył tego określenia, które jest analogią, do faktu, że różnego rodzaju insekty uodporniają się na pestycydy i podobnie jest z błędami. Zauważył on, że jeśli cały czas powtarzamy te same testy, to co prawda, żaden z nich nie wystąpi, jednak nie oznacza to, że w „innym miejscu" one się nie rozmnażają. Czyli co jakiś czas trzeba dodawać nowe techniki i obszary testowania.
6. **Nie wszystkie znalezione błędy zostaną naprawione**. Ten paradoks, ze wszystkich do tej pory wymienionych może budzić najwięcej kontrowersji. Z pozoru wydaj się, że przecież naszym obowiązkiem jest naprawić każdy błąd. W praktyce tak się nie dzieje i podamy kilka przyczyn takiego stanu rzeczy:
   a) **Brak uzasadnienia ekonomicznego** – brutalna prawda, w ostatecznym rozrachunku liczy się rachunek ekonomiczny, a oprogramowanie nawet, jeśli posiada gwarancję, to nie dożywotnią. Więc jeśli Klient znajdzie błąd i będzie musiał dużo za jego naprawę zapłacić, to nie zdecyduje się tego zrobić, jeśli koszt przekroczy wartość płynącą z tej naprawy.
   b) **Brak czasu** – znów bardzo życiowy argument, nawet jeśli błąd został znaleziony, to może okazać się, że łatwiej nauczyć się z nim żyć, niż czekać na oddanie produktu finalnego.
   c) **Za duże ryzyko naprawy** – należy pamiętać, że zawsze poprawiając błąd modyfikujemy program, co za tym idzie istnieje szansa wprowadzenie inne błędu. A jeśli dana funkcjonalność jest bardzo „delikatna"? Może lepiej żyć z tym błędem, niż niepotrzebnie ryzykować?
7. **Testy są nużące** – testy i sam proces testowanie jest nużący. Jeśli ten sam test wykonywany jest w koło przez jedna osobę, to w pewnym momencie będzie on na tyle nudny, że osoba go wykonująca może zacząć popełniać błędy. Stąd też powszechny trend w stronę automatyzacji testów.
8. **Testowanie wymaga wyobraźni i złośliwości**. To z pozoru dziwne stwierdzenie ma głęboki sens. Otóż pisaliśmy już, że testowanie np. funkcji sinus dla argumentów co 180 stopni ma nikły sens. Ale przecież większość z nas z początku przetestuje ją dla wartości znanych ze szkoły, prawda? A niestety należy użyć wyobraźni i testować dowolną funkcję dla „złośliwych" przypadków, takie, których najmniej można się spodziewać. Do tego trzeba użyć wyobraźni, a czasami mieć również wiedzę dziedzinową związaną z tą funkcją. Dlatego tak ważna jest współpraca z Klientem, jako specjalistą w tej dziedzinie.

Precyzja vs. trafność

Zanim przejdziemy dalej musimy jeszcze uściślić dwa pojęcia: precyzję i trafność, gdyż są one istotne z punktu widzenia procesu testowania. Spróbujmy je wyjaśnić w poniższym przykładzie trafiania w cel:

- Trafienia niecelne i nieprecyzyjne – to takie, w których strzelamy dowolnie losowo po całej tarczy.
- Trafienia celne, lecz nieprecyzyjne – to takie, które są blisko środka, ale porozrzucanie (nie skupione).
- Trafienia niecelne, za to precyzyjne – to takie, kiedy udało się trafić w jeden punkt tarczy, ale nie jest to środek.
- Trafienie celnie i precyzyjne – chyba nie wymagają komentarza.

Zatem z naszego punktu widzenia, należy prowadzić testy w sposób i celny i precyzyjny, ale jak to zrobić?

Techniki testowania

Dość oczywiste jest, że niezależnie od rodzaju błędów, część z nich jest łatwiej znaleźć, inne trudniej. Niewątpliwie jednym z najlepszych sposób wykrywania błędów jest planowanie testów już w trakcie tworzenia oprogramowania, a nawet jego specyfikowania. Jak pisaliśmy, są metodyki

tworzenie oprogramowania w których testy muszą powstać zanim zacznie się kodować np. TDD (Test Driven Development) czy XP (eXtreme Programming). Przy podejściu jakie prezentują te metodyki, testy są tworzone już na poziomie specyfikowania wymagań. Jeśli zastanowimy się, to ma to ogromny sen, bo przecież, skoro Klient oczekuje policzenia płac, to niech poda konkretny przykład na którym możemy to sprawdzić. Taki test zwykle nosi nazwę testu jednostkowego (unit test).

Testy jednostkowe są jednym z najprostszych, a jednocześnie bardzo przydatnych, sposobów przeprowadzania testów. Celem tych testów jest sprawdzenie poprawności pewnej części kodu, zwykle dotyczy to konkretnej funkcji, procedury czy metody. Dążymy w nich, zatem do takiej separacji, by każdy test był niezależny i dawał jednoznaczna odpowiedź, czy dany fragment kodu działa poprawnie. Typowym przykładem takiego testu jest fragment programu, w którym wywołujemy daną funkcję z określonym argumentem i porównujemy wynik ze spodziewanym rezultatem. Przykładowo, łatwo stworzyć test jednostkowy dla funkcji sinus, w tym celu wywołujemy tę funkcję z argumentem o zadanej wartości, a następnie porównujemy rezultat z wartością tej funkcji dla tego argumentu, wartość tego rezultatu jest doskonale znana. Dużo gorzej jest testować złożone funkcje, np. trójmian kwadratowy – tak naprawdę należałoby napisać testy dla każdego z pierwiastków, delty i ostatecznie całej funkcji. Oczywiście im bardzie złożony kod, tym bardziej złożony zbiór testów należy przygotować. Natomiast przy tworzeniu testów jednostkowych należy kierować się, co najmniej poniższymi zasadami:

- Test jednostkowy powinien obejmować jak najmniejszą jednostkę funkcyjną, dzięki temu będzie możliwość sprawdzenia poprawności działania poszczególnych operacji, a nie ich ciągu. Choć sprawdzanie sekwencji działań też bywa konieczne.
- Dla każdej testowanej jednostki należy sporządzić jak najwięcej testów. Dzięki temu mogą zostać wykryte błędy na większym obszarze argumentów. Przykładowo dla wspomnianej funkcji sinus, testowanie jej dla pojedynczego argumenty traci zupełnie sens.
- Testy powinny być możliwie „złośliwe”, lub inaczej, należy testować nie „tendencyjnie”. Dzięki temu możemy znaleźć nietypowe przypadki. Przykładowo testowanie funkcji sinus dla 0 a potem, co 180 stopni, nie ma najmniejszego sensu ze względu na jej okresowość.
- Testy powinna przygotowywać osoba niebędąca autorem kodu, przy czym przygotowywać oznacza tu określać, jaki test powinien być przeprowadzony. Dzięki temu nie powstanie podświadoma chęć udowodnienia, że wszystko wykonaliśmy prawidłowo.
- Każdy przypadek błędnie działającej funkcji (znalezionego błędu), musi obowiązkowo trafić, jako przypadek testowy, do repozytorium. W ten sposób unikamy niespodziewanego pojawiania się błędów, które już były usunięte.

Testowanie a Scrum

Ponieważ Scrum nie określa żadnych surowych zasad związanych z testowaniem, to my w tym module wspomnimy jedynie o tym procesie, oraz zaprezentujemy w części ćwiczeniowej sposoby wykonania testów, jednak bez narzucania samego cyklu testowania.

Podsumowanie

W tym rozdziale przedstawione zostały ogólne pojęcia związane z testami, zatem powiedzieliśmy co to jest błąd, problem oraz inne określenia nieodpowiedniego działania programu. Dalej przedstawiliśmy pojęcia związane z testami i ich obszarami, oraz nakreśliliśmy ogólny zarys dlaczego testowanie jest tak trudne i kłopotliwe. Powiedzieliśmy w jaki sposób testy i proces testowania wpływa na jakość kodu i ostatecznie program. Dalej też wymieniliśmy problemy związanie z testowaniem, oraz wskazaliśmy, ze nie każdy błąd będzie poprawiony wskazując jednocześnie potencjalne przyczyny.

Musisz pamiętać, że testowanie jest niejako elementem zapewniania jakości tworzonego oprogramowania, a zatem podobnie jak tamten proces nie wolno go ignorować, ani też

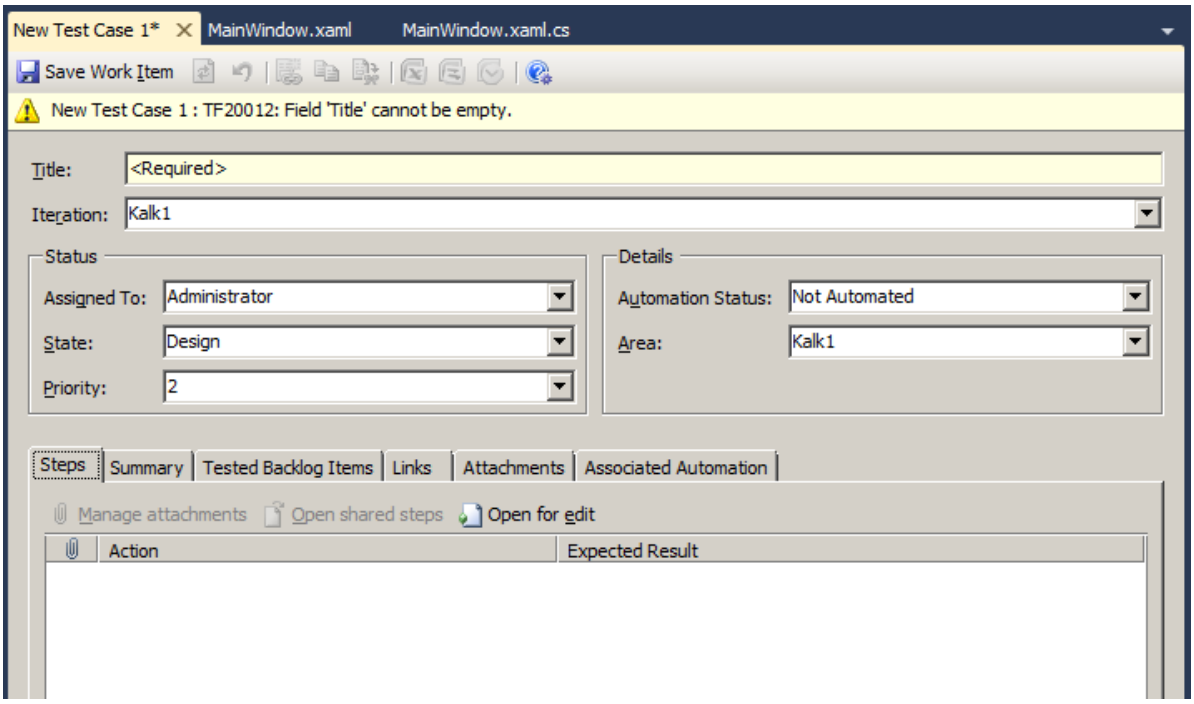
bagatelizować. Jednak by otrzymać właściwe efekty niezbędna jest automatyzacja testów, dzięki czemu unikniemy problemu, iż testy są nużące.

Przykładowe rozwiązanie

Zademonstrujemy w jaki sposób można tworzyć przypadki testowe w ramach środowiska Visual Studio 2010 oraz za pomocą portalu Team Web Access, a następnie pokażemy w jakiś sposób można nimi zarządzać przy wykorzystaniu narzędzia Microsoft Test Manager. To ostatnie wymienione narzędzie pozwala nie tylko na definicję pojedynczych przypadków testowych, ale również na łączenie ich w pakiety, a także definiowanie konfiguracji dla wykonywanie testów i ich pakietów. Co więcej, to właśnie to ostatnie wymienione narzędzie jest dedykowane do tworzenia testów, zatem warto się z nim zapoznać. Zacznijmy jednak od definicji przypadku testowego w ramach środowiska Visual Studio 2010.

Ogólna definicja przypadku testowego

Otwórzmy projekt Kalkulatora, czyli podłączamy się do TFS, oraz otwieramy projekt, tak byśmy widzieli kod źródłowy. Teraz w oknie Team Explorer na drzewie Kalk1 klikamy w węzeł Work Items, w menu kontekstowym wybieramy New Work Item, a następnie klikamy na Test Case. W konsekwencji otworzy się okno dialogowe jak poniżej.

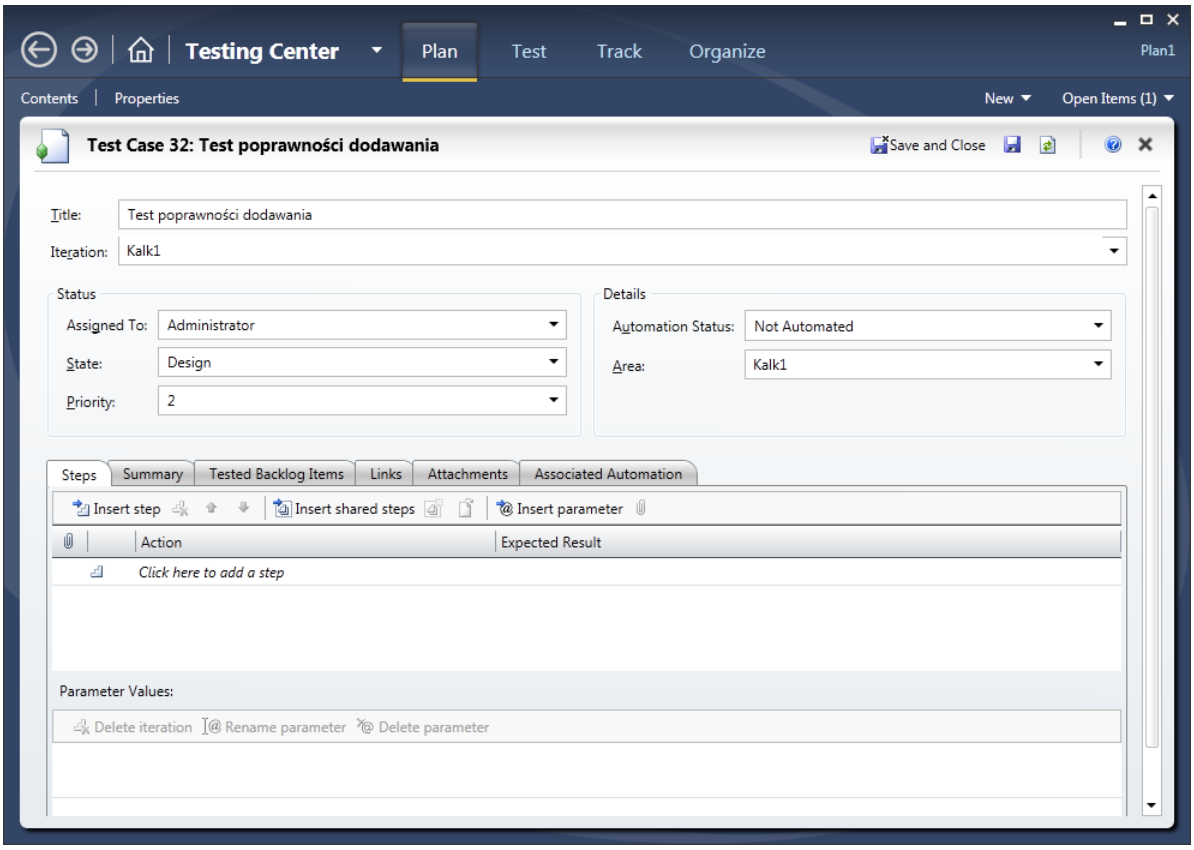


Rysunek 3. Okno dialogowe pozwalające definiować przypadek testowy.

W oknie tym mamy następujące pola:

- **Title** – tytuł przypadku testowego (jedyne obowiązkowe pole).
- **Iteration**, **Assignet To**, **Area, Links, Attachments** – mają identyczne przeznaczenie jak we wcześniej omawianych elementach typu Work Item, zatem pomijamy ich omówienie. Podobnie w zakładce **Summary** znajdują się znane już pola, jak: **Description** i **History**.
- **State** – oznacza jak zwykle stan, ale diagram stanów jest inny od wcześniej znanych i będzie szerzej omówiony w dalszej części modułu. Na chwilę obecną najważniejszą informacją jest, że można wykonać test, którego stan jest ustawiony na Design.
- **Priority** – oznacza priorytet testu, dopuszczalne wartości to 1..4, przy czym 1 oznacza najważniejszą grupę testów.
- **Associated Automation** – pozwala na skojarzenie testu z automatem.

- **Steps** – pozwala na określenie kroków wykonania testu. Do tego służy przycisk Open for edit wywołujący Microsoft Test Manager, ale uwaga, można dodać kroki dopiero po zapisaniu testu, w przeciwnym razie środowisko MTM nie zadziała prawidłowo. W przypadku wywołania okna MTM będziemy mogli wprowadzić kolejne kroki testu (Rysunek 4.).



Rysunek 4. Okno Testing Center w ramach Microsoft Test Manager.

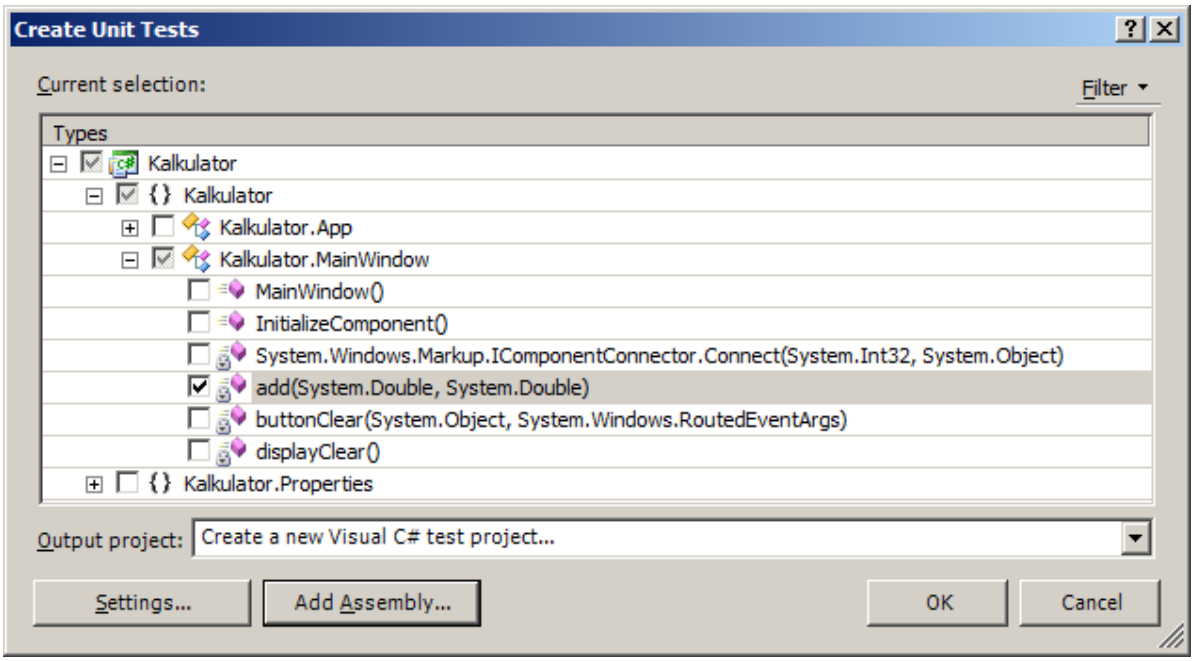
Powyższe okno jest ogólnym sposobem wprowadzania nowego przypadku testowego, pokażemy teraz jak w łatwiejszy sposób można zdefiniować test jednostkowy dla konkretnej funkcji czy metody.

Definiowanie testu jednostkowego dla metody

Zacznijmy od wprowadzenie w naszej głównej klasie (MainWindow) dodatkowej prywatnej metody, która będzie wykonywać prosta operacje dodawania i zwracać wynik. Wprowadzamy ją jedynie po to by pokazać na konkretnym i prostym przykładzie jak stworzyć przypadek testowy. Można oczywiście testować np. wywołania metod naciśnięcia przycisku, ale jest to trudniejsze. Kod tej metody może mieć taką postać:

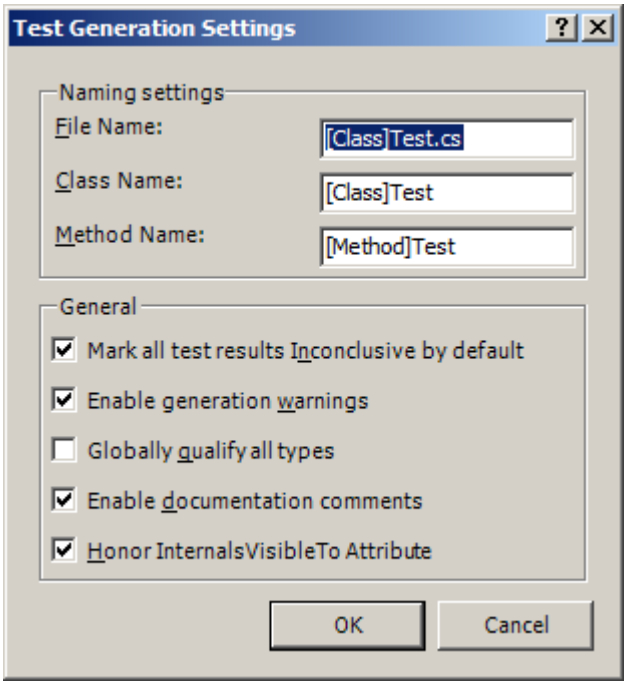
```
public partial class MainWindow : Window
{
…
  private double add(double a, double b)
  {
     return a + b;
  }
}
```

Najeżdżamy teraz kursorem myszy na nazwę tej metody i naciskamy prawy przycisk myszy. Pojawia się menu kontekstowe, z którego wybieramy pozycję Create Unit Tests, w efekcie pojawi się okno dialogowe z dodatkowymi opcjami (Rysunek 5.).



Rysunek 5. Okno dialogowe tworzenia testów jednostkowego.

W oknie dialogowym tworzenia testów jednostkowego mamy możliwość wybrania metod dla których chcemy utworzyć testy. Pozostawmy zaznaczoną tylko metodę add. W liście Output project możemy wybrać jaki typ projektu ma być utworzony do testowania naszej aplikacji, jak widać różnica polega jedynie na wyborze języka programowania. Natomiast dużo więcej możemy ustawić w oknie dialogowym, które ukarze się po naciśnięciu przycisku Settings (Rysunek 6.).



Rysunek 6. Okno dialogowe ustawień dla tworzenia testów jednostkowych.

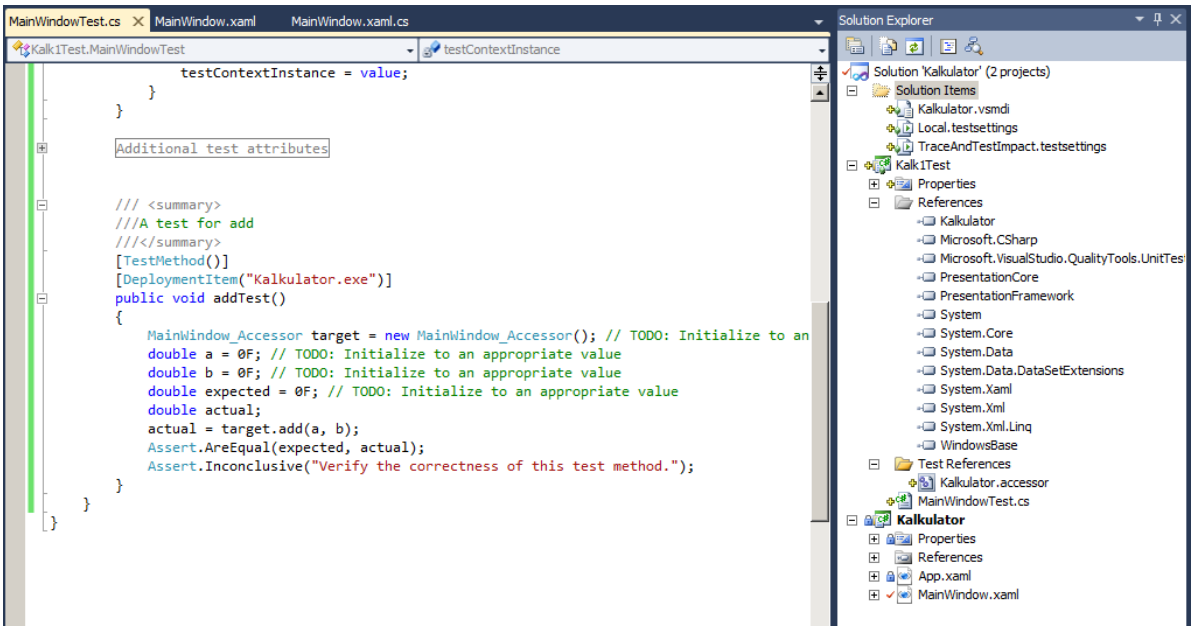
W oknie tym mamy możliwość określenia:

- Sposobu tworzenie nazwy: pliku zawierającego kod klasy testującej, nazwy klasy testującej oraz nazwy metody testującej, na podstawie nazw istniejących. Jak widać domyślnym sposobem jest doklejanie do istniejących nazw słowa Test.
- Włączenia lub wyłączenia tworzenia w ramach metody testowej instrukcji, która będzie informowała, iż ta metoda wymaga jeszcze wypełnienia treścią. Domyślny kod dodawany będzie miał postać jak poniżej lub podobną:

```
Assert.Inconclusive("TODO: Implement code to verify target");
```

- Włączenia lub wyłączenia pojawiania się ostrzeżeń, jeśli takie by wystąpiły w trakcie generowania kodu.
- Włącza globalną kwalifikację wszystkich typów. Oznacza to, że do deklaracji każdej zmiennej zostanie dodany kwalifikator **global::**. Ma to zastosowanie w przypadku podobnych nazw w różnych przestrzeniach nazw.
- Włącza lub wyłącza komentarze dla potrzeb dokumentacji.
- Włącza lub wyłącza traktowanie metod wewnętrznych lub zaprzyjaźnionych jako publiczne.

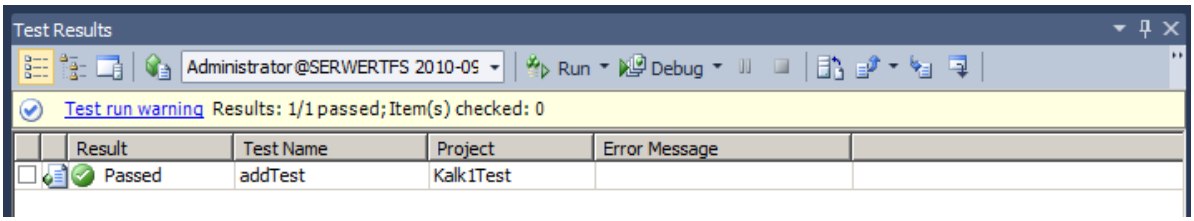
Pozostawmy domyślne wartości w tym oknie i naciśnijmy OK. W poprzednim oknie dialogowym również naciśnijmy OK. Pojawi się okno dialogowe New Test Project, w którym jesteśmy poproszenie o podanie nazwy projektu, który będzie zawierał te testowe klasy i metody. Wpisujemy własną nazwę np. Kalk1Test i naciskamy Create. W tym momencie środowisko VisualStudio 2010 tworzy nowy projekt. Po utworzeniu w oknie Solution Explorer widzimy drzewo projektu, które jak widać zawiera mnóstwo metod, oraz obok okno zawierające kod klasy MainWindowTest z metodą addTest() (Rysunek 7.).



Rysunek 7. Częściowy widok środowiska VS2010 po utworzeniu projektu testów dla projektu Kalkulator.

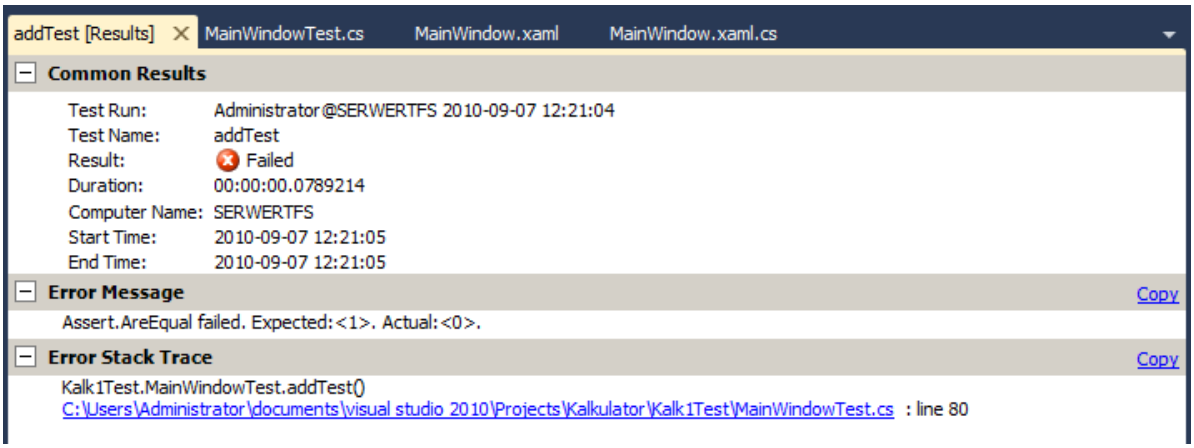
W tej chwili nie będziemy zajmować się elementami z drzewa projektu wyświetlanego w ramach Solution Explorer, przyjmując, że ona są potrzebne. Natomiast spójrzmy w okno zawierające kod źródłowy i przyjrzyjmy się jak wygląda kod metody testującej (addTest) naszą metodę (add). Jak widać, wpierw tworzony jest obiekt naszej klasy (MainWindow), następnie definiowane są dwie zmienne dla argumentów i przypisywane im domyślne wartości. Po czym dwie zmienne dla spodziewanego wyniku i obliczonego wyniku. Kolejna linia wykonuje testowaną funkcję, a następnie wartość ta jest porównywane z wartością spodziewaną. Spróbujmy wykonać ten test, w tym celu np. kliknijmy prawym klawiszem w oknie programu i z menu kontekstowego wybierzmy Run Tests. Na dole pojawi się okno wyniku testu i otrzymamy informację o tym, że metoda

testująca jest nie wypełniona. Oj! Zapomnieliśmy skasować ostatnio linię kodu w metodzie testującej. Zróbmy to i jeszcze raz uruchommy test (np. poprzez naciśnięcie Run w oknie Test Results), zostanie on wykonany i dostaniemy komunikat, iż się powiódł (Rysunek 8.).



Rysunek 8. Okno wyniku wykonania testów z informacją, że test zakończył się powodzeniem.

Przekonajmy się co się stanie, jeśli zrobimy wartości argumentów bardziej „złośliwe” – przypiszmy a=1/3 i b=2/3, oczywiście wartość spodziewana będzie wynosić 1. Po tych modyfikacjach uruchommy test. Ups… nie działa, aby dowiedzieć się więcej, kliknijmy prawym klawiszem na wierszu rezultatu test i wybierzmy z menu View Test Results Details, spróbujmy przeanalizować wynik (Rysunek 9.).



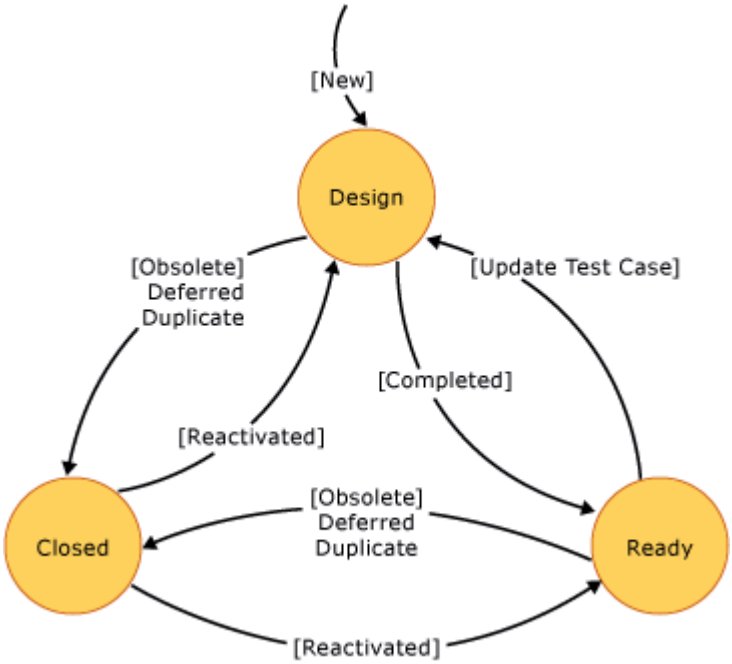
Rysunek 9. Okno zawierające szczegółowe informacje o wyniku testu (w tym przypadku niepowodzeniu).

A co to się stało? Przecież 1/3 + 2/3 to powinno być 1, lub 1 z pewną dokładnością (pamiętajmy, że operujemy na liczbach typu double, więc obowiązuje np. epsilon maszynowy). Zatem dlaczego w tym raporcie mamy informację, że aktualną (obliczoną) wartością jest 0? Zapomnieliśmy, że jeśli napiszemy tak po prostu 1/3, to środowisko przyjmie, że jest to dzielenie liczb całkowitych i wyniku otrzymamy tylko część całkowitą operacji, zatem 0, podobnie z 2/3 znów będzie 0, zatem ile jest 0+0? Co ilustruje ten przykład? Co najmniej dwie bardzo ważne rzeczy:

1. **Testowanie metod wymaga wiedzy** – musimy rozumieć co dzieje się w kodzie, dlatego tester również musi znać elementy języka programowania.
2. **Testowanie może być źródłem błędów** – bardzo ważna obserwacja, pisząc test musimy być pewni, że sam z siebie nie zawiera błędów! Inaczej ten błąd przeniesiemy na program i skutki jego działania mogą być nieprzewidywalne.

Cykl życia przypadku testowego

Kiedy przypadek testowy jest definiowany, jego stan automatycznie jest ustawiamy na Design, zmiana stanu na Ready, powinna odbyć się, gdy zespół zdefiniuje wszystkie akcje z nim związane. Graficzne zobrazowanie zmian stanów przypadku testowego widać na rysunku 10, a podsumowanie zmian stanów znajduje się w tabeli 1.



Rysunek 10. Cykl życia przypadku testowego wyrażony za pomocą zmian jego stanu. Linia ciągła to zmiana typowa, przerywana to zmiany nietypowe.

Zmiana stanu przypadku testowego	Przyczyna
Design >> Ready	Następuje, gdy członek z zespołu uznał, że przypadek testowy jest gotowy jako przypadek test akceptacyjny wymagań użytkownika.
Ready >> Closed	Następuję, gdy zostanie uznane, że ten przypadek testowy nie będzie więcej używany.
Design >> Closed	Oznacza, że dany przypadek testowy nie jest odpowiedni, lub na przykład, że dubluje już istniejący.
Ready >> Design	Oznacza, że został odkryty dodatkowy zakres dla przypadku testowego i musi on zostać ponownie zaprojektowany.
Closed >> Design	Oznacza, że przypadek testowy został zamknięty przez pomyłkę, albo, że jego zakres został z powrotem uznany za ważny.

Tabela 1. Zmiana stanów przypadku testowego.

Porady praktyczne

Testy, jak i poprzednio wymieniane elementy, jak zarządzanie ryzykiem czy zapewnianie jakością, bywają niedocenianym obszarem procesu wytwarzania oprogramowania. Dzieję się tak, gdyż większość programistów wychodzi z założenia, że skoro kod, który stworzyli skompilował się i dało się wprowadzić schematyczne dane, to znaczy, że działa! Nie warto w tym miejscu przytaczać wielu sytuacji, w których to doprowadziło do zgubnych sytuacji, ale z drugiej strony warto pracować w zespole nad świadomością, że testy są potrzebne. Oczywiście, testy muszą wykonywać testerzy a nie programiści, gdyż Ci drudzy za wszelką cenę będą próbowali udowodnić, że napisali oprogramowanie bezbłędne. Jednak, by testy były naprawdę skuteczne należy je wpleść w cała metodykę wytwarzania oprogramowania i proces testowania podobnie jak zarządzanie ryzykiem, czy też zapewniania jakością. Zatem proces testowania powinno planować się już od sameg początku projektu, aż do samego jego końca.

Zdecydowanie proces testowania, w obecnych czasach, przy tak złożonych systemach informatycznych, jest również problemem bardzo złożonym i uważa się go za istotny element

zapewnienia jakości. Co więcej nie byłoby szans na dobrą realizację tego procesu bez automatyzacji tych testów, stąd też, jeśli mówimy o dobrym i profesjonalnym wykonywaniu testów należy użyć środowisk programistycznych, które to wspierają.

Uwagi dla studenta

Jesteś przygotowany do realizacji laboratorium, jeśli:

- wiesz czym różnią się pojęcia błędu, problemu, usterki,
- wiesz jaka jest rola testów oraz procesu testowania,
- potrafisz wprowadzić do środowiska przypadek testowy, oraz test jednostkowy,
- rozumiesz role procesu testowania oraz wpływ tego procesu na zapewnienie jakości w projekcie.

Pamiętaj o zapoznaniu się z uwagami i poradami zawartymi w tym module. Upewnij się, że rozumiesz omawiane w nich zagadnienia. Jeśli masz trudności ze zrozumieniem tematu zawartego w uwagach, przeczytaj ponownie informacje z tego rozdziału i zajrzyj do notatek z wykładów.

Dodatkowe źródła informacji

1. Ron Patton, *Testowanie oprogramowania*, MIKOM 2002

   W książce Autor porusza zagadnienia związane z testowaniem oprogramowania. Książka stanowi dobry przegląd po różnego rodzaju obszarach i technikach testowania.

2. Boris Beizer, *Software Testing Techniques*, Van Nostrand Reinhold Co., USA 1990

   W książce Autor opisuje wiele różny technik testowania.

Laboratorium podstawowe

Zadanie 1

Czy pamiętasz, że w poprzednim module, gdy zdefiniowaliśmy ograniczenie, to nie mogliśmy je wybrać z bazy? Stało się tak dlatego, że nie było zdefiniowane domyśle zapytanie, które pozwala wybrać z bazy wszystkie ograniczenia, niezależnie od ich przynależności do konkretnego sprintu, czy iteracji. Twoim zadaniem jest zdefiniowanie zapytania All Tet Case, które pozwoli wyświetlić wszystkie przypadki testowe w projekcie.

Zadanie 2

Stwórz testy jednostkowego dla następujących funkcji prostego kalkulatora:

- Mnożenie
- Dzielenie – tu zastanów się, czy należy uwzględnić dodatkowe podprzypadki.

Laboratorium rozszerzone