PRACE NAUKOWE AKADEMII EKONOMICZNEJ WE WROCŁAWIU
Nr 955 2002
Nowoczesne technologie informacyjne w zarządzaniu

Lech Madeyski

Wydziałowy Zakład Informatyki
Wydział Informatyki i Zarządzania
Politechnika Wrocławska
e-mail: madeyski@ci.pwr.wroc.pl

NOWE KONCEPCJE TWORZENIA APLIKACJI INTERNETOWYCH NA PRZYKLADZIE PORTALU E-INFORMATYKA.PL

Streszczenie: W artykule zaprezentowano najważniejsze koncepcje i rozwiązania stosowane przez autora przy tworzeniu zaawansowanych internetowych aplikacji biznesu elektronicznego, a w szczególności przy projekcie portalu e-informatyka.pl. Szczególną uwagę poświęcono personalizacji, warstwie prezentacji, która zapewniałaby obsługę zarówno istniejących, jak i mogących się pojawić w przyszłości urządzeń (komunikatorów osobistych), aplikacji czy formatów prezentacji, a także zagadnieniom integracji systemów. Przedstawiono koncepcje integracji technologii platformy J2EE, XML i serwisów sieciowych, najnowsze metodyki tworzenia oprogramowania, ich mocne i słabe strony, a w rezultacie sposób odpowiedniego doboru oraz przykładowy podział kompetencji członków zespołu.

1. Wprowadzenie

W Polsce odczuwalny jest brak liczących się, profesjonalnych czasopism elektronicznych o charakterze naukowym i popularno-naukowym, poświęconych najnowszym trendom i technologiom informatycznym, w szczególności biznesowi elektronicznemu, tworzeniu aplikacji internetowych, zarządzaniu projektami informatycznymi, inżynierii oprogramowania, metodykom tworzenia oprogramowania, nowoczesnym technologiom i systemom informatycznym.

Istniejące portale i witryny internetowe nie wytworzyły prężnych społeczności ludzi zainteresowanych wymienionymi zagadnieniami, chcących pogłębiać swoją

wiedzę, wspólnie rozwijać się i tworzyć nowe, zaawansowane technologicznie rozwiązania.

Wychodząc naprzeciw powyższemu zapotrzebowaniu autor podjął próbę stworzenia portalu e-informatyka.pl. Portal jest tworzony pod egidą Polskiego Towarzystwa Informatycznego. Głównym filarem portalu ma być czasopismo elektroniczne „e-Informatyka" kontynuujące najlepsze tradycje ukazującej się do niedawna „Informatyki". Wydawanie czasopisma w formie tradycyjnej (papierowej) wiąże się z dużymi kosztami, długim cyklem publikacji (co uniemożliwia szybkie informowanie czytelników o nowościach na stale zmieniającym się rynku informatycznym), ścisłymi ograniczeniami na objętość każdego numeru, a co za tym idzie ograniczoną liczbą artykułów, ograniczeniami na nakład i zasięg artykułów i utrudnionym dostępem (także do materiałów archiwalnych).

W zamierzeniu autora cele projektu e-Informatyka.pl są następujące:

- Stworzenie społeczności zainteresowanej najnowszymi trendami i technologiami informatycznymi m.in. przy pomocy liczącego się, profesjonalnego, elektronicznego czasopisma.
- Podnoszenie kwalifikacji i ułatwienie wymiany informacji w środowisku zawodowym.
- Łatwy dostęp do ludzi profesjonalnie zajmujących się informatyką.
- Obniżenie kosztów wydawania czasopisma.
- Skrócenie czasu potrzebnego na opublikowanie artykułu.
- Brak ograniczeń na liczbę artykułów i rozmiar kolejnych numerów.
- Światowy zasięg czasopisma.
- Łatwy dostęp do opublikowanych materiałów.

Potencjalnymi użytkownikami portalu e-informatyka.pl są pracownicy naukowo-dydaktyczni wyższych uczelni na kierunkach informatycznych, doktoranci, magistranci i studenci kierunków informatycznych, pracownicy działów IT, wydawnictwa naukowe i popularno-naukowe, dostawcy sprzętu i oprogramowania, towarzystwa informatyczne (PTI, PIIT, NTIE), a nawet zewnętrzne serwisy korzystające z udostępnianych przez nas usług sieciowych (*Web services*) np. publikowanych w naszym systemie treści. Warto również wspomnieć o zewnętrznych serwisach udostępniających usługi sieciowe, z których nasz portal będzie mógł korzystać np. systemy wyszukiwawcze lub udostępniające treści związane z informatyką.

Główne założenia przyjęte podczas projektowania portalu e-Informatyka.pl to przenośność, otwartość i łatwość integracji z innymi systemami, a także z różnymi urządzeniami dostępowymi po stronie klienta. W kolejnych rozdziałach przedstawione zostaną, ważne zdaniem autora, aspekty funkcjonalny, architektoniczny i realizacyjny związane z tworzeniem zaawansowanych aplikacji internetowych.

2. Aspekt funkcjonalny

Jedną z najbardziej ekscytujących cech nowoczesnych aplikacji internetowych są możliwości personalizacji i dostosowania prezentacji (generacji widoków) do szeroko rozumianych preferencji użytkownika. Warto w tym miejscu zdefiniować różnicę pomiędzy dwoma kluczowymi pojęciami: personalizacją (*personalisation*) i dostosowaniem (*customization*) prezentacji. Różnica pomiędzy personalizacją a dostosowaniem różnych aspektów prezentacji (m.in. jej treści czy postaci) tkwi w sposobie interakcji. Dostosowanie bazuje na wyborach dokonanych bezpośrednio przez użytkownika np. „Interesuje mnie A, B i C, nie interesuje mnie X, Y i Z". Personalizacja bazuje natomiast na obserwacji użytkowników np. „Ten użytkownik często zagląda do artykułów na temat XML, a to nie znajduje odzwierciedlenia w jego profilu. Dlaczego więc nie zasugerować odpowiedniej modyfikacji profilu?". Prezentowaną informacje należy oczywiście dostosować do kontekstu użycia np. w sytuacji, gdy użytkownik zmienił urządzenie dostępowe to należy odpowiednio dostosować prezentacje. Bardziej precyzyjnie można powiedzieć, że personalizacja jest procesem gromadzenia i przechowywania informacji o osobach odwiedzających witrynę, a następnie analizy zgromadzonych informacji i w efekcie dostarczaniem właściwej treści we właściwym czasie i we właściwy sposób.

Inaczej mówiąc spersonalizowana i dostosowana prezentacja jest wypadkową elementów profilu użytkownika. Profil użytkownika (*user profile*) można podzielić na dwa główne komponenty:

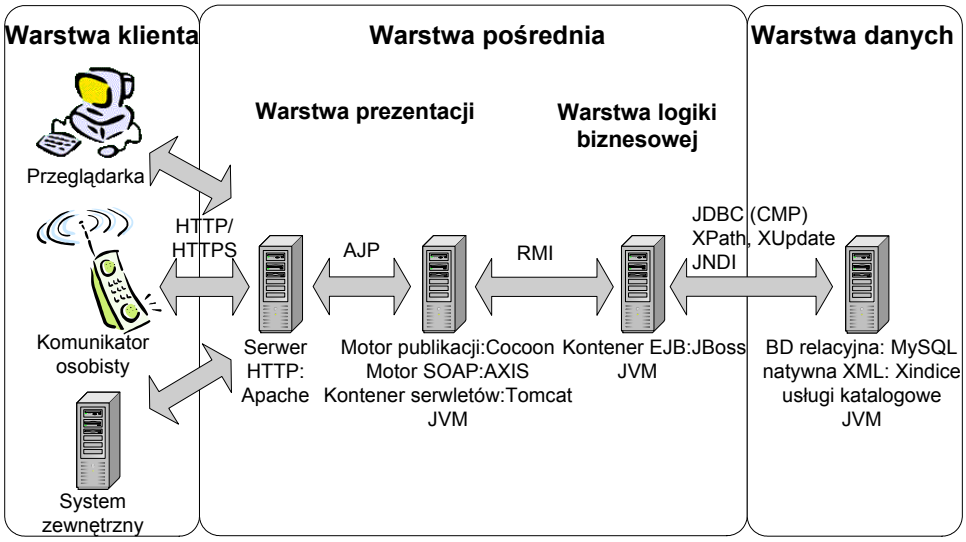
1. Profil preferencji użytkownika (*user preferences profile*) umożliwiający selekcję treści na podstawie aktualnych preferencji użytkownika (np. profil może wskazywać zainteresowanie użytkownika metajęzykiem XML i jego aplikacjami).
2. Profil urządzenia i oprogramowania systemowego, jak i aplikacyjnego, używanego przez użytkownika (*user agent profile*) umożliwiający formatowanie treści we właściwy sposób, dopasowany do możliwości prezentacyjnych posiadanego sprzętu i oprogramowania. Komponentami tego profilu mogą być również parametry nawiązanego połączenia sieciowego, mające wpływ na możliwości prezentacyjne urządzenia.

Szczegółowe omówienie zawartości profilów, stosownych rekomendacji, sposobów implementacji i wykorzystania odpowiednich mechanizmów jest bardzo interesującym, ale i obszernym tematem godnym osobnego artykułu. Zagadnienie to jest o tyle istotne, że odpowiednie wykorzystanie profilu użytkownika umożliwia nie tylko stworzenie portalu niezależnego od używanych urządzeń klienckich (*device independent site*), ale równocześnie uwzględniającego potrzeby i preferencje indywidualnych użytkowników (*user focused site*). Warto zaznaczyć,

że odpowiednia kombinacja technologii bazujących na XML (w szczególności wykorzystanie XSLT) wydaje się jak najbardziej stosownym rozwiązaniem technologicznym umożliwiającym realizacje powyższych celów.

3. Aspekt architektoniczny

Termin architektura systemu może mieć różne znaczenia i jest w różny sposób definiowany. Idee użycia różnych widoków do wyrażenia architektury systemu wyraził Kruchten [KRUC00] opisując „*The 4+1 View Model of Architecture*". Zwykle architektura jest wiązana z technologiami lub produktami. Warto jednak pamiętać, że jest to tylko jeden z wielu możliwych widoków, a technologia to nie architektura. W związku z tym proponuje się niekiedy podział na architekturę logiczną (opisującą główne cechy systemu niezależnie od implementowanych technologii) i architekturę fizyczną systemu (opisującą realizację architektury logicznej w postaci implementowanych technologii, które mogą zawierać standardy, języki programowania, technologie komponentowe i określone produkty). Używając powyższej terminologii rys. 1 przedstawia uproszczoną architekturę fizyczną portalu.



Rys. 1. Uproszczona architektura fizyczna portalu e-Informatyka.pl.
Źródło: opracowanie własne.

Architektura portalu e-Informatyka.pl bazuje w dużej mierze na platformie J2EE (*Java 2 Enterprise Edition*), która jest bardzo udaną, otwartą platformą używaną do budowy dużych, skalowalnych aplikacji internetowych. Pomimo

powszechnej akceptacji i wielu niewątpliwych zalet tej platformy może być ona postrzegana jako mało elastyczna lub nawet niekompletna. Wynika to z faktu, iż aplikacje internetowe J2EE bazują zwykle na języku HTML w warstwie prezentacji. Dzisiejsze systemy biznesu elektronicznego wymagają elastyczności i łatwego wsparcia dla szerokiego spektrum nowych urządzeń i aplikacji pojawiających się po stronie klienta. W przypadku systemów B2B (*Business to Business*) będą to aplikacje partnerów biznesowych z którymi chcemy się integrować. W przypadku systemów B2C (*Business to Customer*) będą to przeglądarki internetowe lub osobiste komunikatory w różnych postaciach (telefony komórkowe, palmtopy czy Pocket PC).

Kluczową kwestią staje się więc budowa systemów otwartych, wspierających nowe urządzenia po stronie klienta i dających się łatwo integrować z innymi systemami partnerów biznesowych. W tym zakresie duże nadzieje wiąże autor z metajęzykiem XML (*eXtensible Markup Language*), jego aplikacjami, jak również z zupełnie nową koncepcją integracji systemów na bazie XML za pomocą usług sieciowych (*Web services*). Typowym przykładem wykorzystania usług sieciowych jest wsparcie międzykorporacyjnych procesów biznesowych (np. zarządzania łańcuchem dostaw).

Uwzględniając powyższe założenia przy projektowaniu architektury portalu zdecydowano się na integrację technologii platformy J2EE takich, jak serwlety czy EJB (*Enterprise Java Beans*) z technologiami i aplikacjami XML, a w szczególności XSLT (*eXtensible Stylesheet Language Transformation*) i XSL FO (*eXtensible Stylesheet Language Formatting Objects*) oraz technologiami umożliwiającymi wykorzystanie usług sieciowych m.in. SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*) i UDDI (*Universal Description, Discovery and Integration*). Taka architektura to oczywiście wyzwanie technologiczne, zapewnia jednak realizację głównych założeń projektowych (otwartość systemu oraz przenośność danych i kodu). Trudno omawiać szczegółowo całe spektrum wykorzystywanych technologii i narzędzi czy przyjętych założeń architektonicznych. Najważniejsze z nich zostaną jednak pokrótce przedstawione w kolejnych rozdziałach.

3.1. Otwarta i modularna architektura

Istotnym zagadnieniem związanym z budową nowoczesnej aplikacji internetowej jest jej otwarta i modularna konstrukcja. W przypadku portalu e-informatyka.pl funkcjonalność głównej części portalu związanej z czasopismem musi być łatwo uzupełniana podprojektami rozszerzającymi funkcjonalność szkieletową.

Otwartość zapewnia nam w dużej mierze wykorzystanie XML i usług sieciowych. Bardzo dobrym przykładem może być tutaj wykorzystanie usług

sieciowych oferowanych przez bodaj najpotężniejszy system wyszukiwawczy Google (nawiasem mówiąc projekt akademicki) czy najbardziej znaną elektroniczną księgarnię, a właściwie sklep Amazon. Architektura portalu e-Informatyka.pl została tak zaprojektowana, że integracja z innymi systemami poprzez usługi sieciowe jest bardzo prosta, a rezultatem będzie funkcjonalność umożliwiająca w ramach portalu e-Informatyka.pl wyszukiwanie w ponad 2 miliardach dokumentów udostępnionych Internecie lub też wyszukiwanie, przeglądanie i zakup produktów znajdujących się w bazie Amazon. Ta druga usługa może być nie tylko atrakcyjna dla użytkowników portalu, ale również finansowo korzystna dla samego portalu. Jeżeli projekt zacznie mieć częściowo komercyjny charakter to takie usługi dostarczą środków finansowych potrzebnych na utrzymanie portalu. Warto nadmienić, że w najbliższym czasie należy się spodziewać bardzo szerokiej oferty usług sieciowych oferowanych przez wiele podmiotów. Jest to bowiem, zdobywająca sobie szybko zwolenników, atrakcyjna koncepcja integracji systemów.

Modularna konstrukcja aplikacji bazuje w dużej mierze na koncepcji portletów. Portlety to spełniające określone wymogi aplikacje zarejestrowane w repozytorium portletów naszego portalu. Wygląd tworzonych portletów powinien być zgodny z odpowiednimi zaleceniami tak, aby zapewnić jednolity wygląd i nawigacje w ramach całego środowiska (*look&feel*). Tworzone portlety powinny umożliwiać korzystanie ze wszystkich usług i zasobów (także zewnętrznych) do których użytkownik ma prawa dostępu poprzez jednokrotne logowanie się użytkownika do portalu (*single sign on*). Między innymi dzięki umiejętnemu wykorzystaniu możliwości XML, a w szczególności przestrzeni nazw i schematów możliwe jest zdecentralizowane i równoległe rozwijanie aplikacji portalu nawet przez zasoby zewnętrzne.

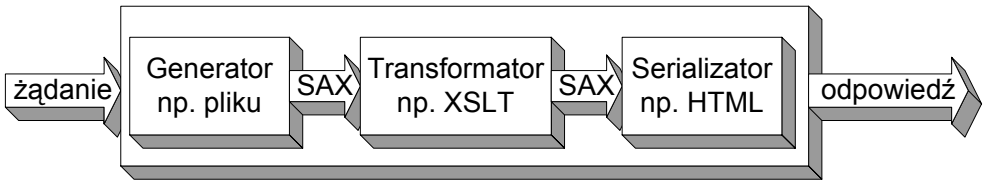
Autor ma nadzieje, ze zaprezentowana architektura i funkcjonalność portalu będzie punktem wyjścia do stworzenia kompletnego modelu nowoczesnej aplikacji internetowej nazwanego roboczo modelem X (ze względu na wielopoziomowe wykorzystanie XML).

3.2. Apache Cocoon 2

Niewątpliwie jednym z najważniejszych i wartym przedstawienia elementem portalu e-Informatyka.pl jest Apache Cocoon 2 - produkt organizacji Apache będący zaawansowanym motorem publikacji XML. W proponowanej przez autora architekturze portalu, Cocoon będzie mógł być wykorzystany zarówno jako szkielet publikacji, realizując zadania związane z prezentacją, jak również jako element wspierający budowę skomplikowanych serwisów sieciowych (np. takich, które wymagają zarządzania stanem pomiędzy kolejnymi wywołaniami) wspomagając procesy integracji systemów.

Warto zatem bliżej się przyjrzeć architekturze Cocoon. W istocie jest to serwlet działający w kontenerze serwletów, jakim jest Tomcat – kolejny produkt organizacji Apache wykorzystywany w portalu e-Informatyka.pl. Cocoon umożliwia publikacje dokumentów XML w różnych formatach (HTML, WML, PDF, SVG czy RTF by wymienić tylko niektóre), co jest realizowane przy użyciu transformacji XSLT.

W Cocoon mamy do czynienia z przetwarzaniem potokowym. Każdy krok transformacji bazuje na dokumencie wygenerowanym w poprzednim kroku. Transformacje powtarzane są, aż do momentu osiągnięcia końca potoku przetwarzania i utworzenia dokumentu wynikowego. Przykład potokowego przetwarzania żądania został przedstawiony na rys. 2.



Rys. 2. Przykład potokowego przetwarzania żądań w Apache Cocoon.
Źródło: Zaadoptowane z [COCO02]

W momencie przyjścia żądania jego przetwarzanie jest delegowane do odpowiedniego potoku. Przykładowa obsługa żądania, przedstawiona na rys. 2, polega na odczytaniu pliku XML z dowolnego URI (*Uniform Resource Identifier*) i przekształceniu go na strumień zdarzeń SAX (*Simple API for XML*) za pomocą generatora pliku. Sekwencja zdarzeń SAX jest następnie odczytana i przekształcona za pomocą transformatora XSLT z wykorzystaniem arkusza stylu XSLT, a wyniki tego przetwarzania są ponownie przekazane do potoku przetwarzania w postaci zdarzeń SAX, które są następnie przetworzone na odpowiedni format odpowiedzi (np. HTML) za pomocą serializatora. Oczywiście Cocoon posiada znacznie więcej komponentów niż te, które zostały zilustrowanane na powyższym przykładzie.

4. Aspekt realizacyjny

Bez względu na to, w jaki sposób próbujemy sobie radzić z problemem wytwarzania oprogramowania dwie rzeczy wydają się pewne. Wciąż nie potrafimy sobie radzić z tym w zadawalającym stopniu i jest mało prawdopodobne, by jedna metodyka, postrzegana jako produkt z pudełka, mogła pasować do każdego projektu. Trudno bowiem zgodzić się z tezą, że zastosowanie określonej metodyki diametralnie zmieniło sytuację i sprawiło, że projekty informatyczne zaczęły się

kończyć sukcesem, w terminie i bez przekroczenia budżetu przy jednoczesnym uzyskaniu wymaganej funkcjonalności.

4.1. Wybór metodyki tworzenia oprogramowania

W praktyce stosowane są różne podejścia począwszy od wciąż jeszcze bardzo popularnej, chaotycznej aktywności w stylu „koduj i poprawiaj" [FOWL00] poprzez nowy, niezmiernie dynamicznie rozwijający się nurt metodyk lekkich (*lightweight metodologies*), określanych od niedawna sprawnymi (*agile metodologies*) po metodyki ciężkie (*heavyweight metodologies*) dyscyplinujące w znacznym stopniu proces tworzenia oprogramowania.

Pierwsze podejście sprawdza się przy bardzo niewielkich projektach. Jak można przypuszczać w innych przypadkach błędy stają się zbyt liczne i trudne do naprawienia. W przypadku tak dużego projektu jak e-Informatyka.pl to podejście zdecydowanie odpada.

Zupełnie odmienne podejście zakłada skorzystanie z którejś z metodyk tworzenia oprogramowania. W tym momencie pojawia się jednak problem wyboru metodyki odpowiedniej do realizowanego projektu. Wybór wagi procesu wydaje się być szczególnie istotny. Nieformalnie można powiedzieć, że w inżynierii oprogramowania proces określa, w jaki sposób wymagania są zamieniane na oprogramowanie. Przyjrzyjmy się bliżej, jakie mamy możliwości wyboru w tym zakresie.

Nurt metodyk sprawnych, mimo iż historycznie najmłodszy, rozwija się wyjątkowo dynamicznie. Pojawiło się całe spektrum metodyk tworzenia oprogramowania począwszy od najbardziej znanych takich, jak XP (*eXtreme Programming*) [BECK01], poprzez metodyki Crystal [COCK02], SCRUM [SCHW02], FDD (*Feature Driven Development*) [COAD99] czy ASD (*Adaptive Software Development*) [HIGH99]. Nowe metodyki, często dobrze sprawdzające się przy niewielkich i średnich projektach, mają charakter adaptacyjny i są swego rodzaju kompromisem pomiędzy brakiem jakiegokolwiek procesu a zbyt ciężkim procesem. Adaptacyjność procesu rozumiana jest zarówno w kontekście dostosowywania się do zmiennych wymagań klientów jak również w kontekście samego procesu. Z biegiem czasu będzie on bowiem ulegał modyfikacjom np. wykorzystywał te techniki, które się sprawdziły.

Nie trzeba wielkiego doświadczenia by wiedzieć, że ze specyfikacją wymagań bywa różnie, a sytuacja, kiedy jest ona spójna i całkowicie określona przed podjęciem prac zdarza się niezmiernie rzadko. Skoro więc nie możemy uzyskać stabilnych wymagań to trudno również stworzyć przewidywalny plan. W związku z tym niezmiernie istotne jest posiadanie mechanizmu pozwalającego precyzyjnie określić aktualną sytuacje, a przez to postęp prac. Kluczową rolę odgrywa tutaj znana od dawna idea iteracyjnego rozwoju (*iterative development*). Istotą

iteracyjnego rozwoju jest częste produkowanie pracujących wersji finalnego systemu posiadających podzbiór wymaganych funkcjonalności. Powinny być one w pełni zintegrowane i przetestowane tak, jakby były ostatecznym produktem [FOWL00]. Wprowadza to pewną dozę realności potrzebną każdemu projektowi poprzez konfrontację z realnymi problemami. Preferowana długość iteracji w metodykach sprawnych waha się od tygodnia do około 2 miesięcy, przy czym jest tendencja, by iteracje były jak najkrótsze, gdyż umożliwia to uzyskanie dokładniejszych informacji o postępie prac.

Wszystkim tworzonym metodykom sprawnym przyświeca minimalistyczna zasada Musashiego by nie robić nic, co nie jest absolutnie konieczne: *„Do not do anything useless”*. Istotne jest bowiem dostarczenie poprawnie funkcjonującego, przetestowanego oprogramowania, a nie tomów dokumentacji.

Kolejnym istotnym czynnikiem wpływającym na sukces projektu są ludzie. Twórcy metodyk sprawnych zwracają szczególną uwagę na umiejętnie wykorzystywanie olbrzymiej wiedzy i indywidualnych zdolności oraz predyspozycji członków zespołu, a także na współdzielenie wiedzy, czego najlepszymi przykładami są programowanie w parach czy krótkie spotkania całego zespołu znane z metodyki XP.

Zasady promowane przez większość metodyk sprawnych to: krótkie cykle wydań, pracujemy tylko nad tym, co jest absolutnie niezbędne, nie marnujemy zbyt wiele czasu na analizę lub projekt, problem opisujemy w prosty sposób i z małych kawałków implementowanych w kolejnych iteracjach tworzymy niezawodny system poprzez ciągłą integrację i testowanie kodu wykorzystując natychmiastową reakcje ze strony użytkowników i programistów.

Niestety metodyki sprawne mają również słabe strony. Ograniczony rozmiar projektu jest często postrzegany jako główne ograniczenie metodyk sprawnych. Niestety nie każdy projekt może być w prosty sposób zdekompownowany na mniejsze części. Okazuje się również, że bardzo trudno jest przekroczyć barierę kilkunastu programistów pracujących nad projektem. Kolejnym ograniczeniem jest posiadanie wysoko wykwalifikowanych i jednocześnie wyjątkowo zdyscyplinowanych programistów gotowych zaakceptować praktyki narzucane przez metodyki sprawne, jak na przykład programowanie parami znane z XP.

Alternatywą zarówno dla zupełnego braku jakiegokolwiek procesu, jak i zbyt lekkiego procesu, jest wykorzystanie metodyki ciężkiej. Największą bodaj popularnością cieszy się obecnie metodyka Unified Process (UP) [JACO99] firmy Rational Software formalnie określana jako Unified Software Development Process (USDP). W tym miejscu warto wspomnieć o Rational Unified Process (RUP), który jest komercyjnym produktem firmy Rational Software rozszerzającym UP. Nie wchodząc w subtelne różnice terminologiczne i syntaktyczne pomiędzy UP a RUP wystarczy powiedzieć, że oba procesy są bardzo zbliżone z tym, że UP jest otwartym procesem generycznym, a nie

komercyjnym produktem. Musi on być uszczegółowiony przez zespół biorąc pod uwagę konkretny projekt. Obejmuje to opracowanie i wykorzystanie standardów, szablonów dokumentów, narzędzi (np. kompilatorów, narzędzi zarządzania konfiguracją), baz danych (np. śledzenie projektów, błędów) itp.

UP jest bardzo kompletnym i szczegółowo opisanym procesem. W szczególności identyfikuje on ponad sto artefaktów, które zespół projektowy może wyprodukować podczas realizacji projektu. Ta kompletność jest niestety często postrzegana jako wada, a sam proces jako zbyt ciężki. Nic jednak nie stoi na przeszkodzie by „odchudzić" proces dostosowując do konkretnego projektu wciąż korzystając z wielu jego zalet. Taka zresztą wydaje się intencja twórców UP, aby dostarczyć zespołowi realizującemu projekt kompletną kolekcję najlepszych praktyk, które mogą zostać dostosowane do specyfiki konkretnego projektu. Pewne projekty będą wymagały więcej artefaktów i większej liczby kroków w planie projektu niż inne. Zwykle jest tak, że im większe jest ryzyko związane z projektem, tym więcej uwagi trzeba poświęcić na ograniczenie ryzyka, co może wymagać bardziej szczegółowych opracowań czy większej liczby zadań do zrealizowania.

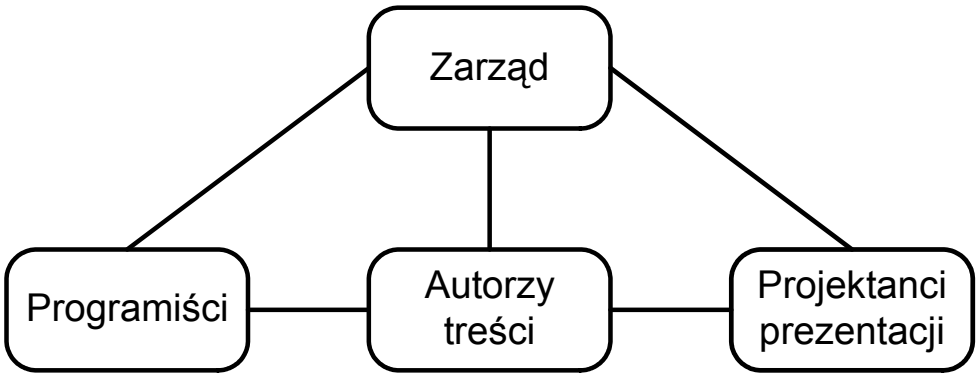
Trudno szczegółowo opisywać proces UP, warto jednak wspomnieć o najważniejszych aksjomatach. Przypadki użycia (*use cases*) są preferowanym sposobem specyfikacji wymagań. UP jest procesem iteracyjnym i inkrementacyjnym. Aspekt iteracyjny UP oznacza, że dzielimy projekt na małe podprojekty (iteracje), które dostarczają nam częściowej funkcjonalności prowadząc w kolejnych inkrementacjach do w pełni funkcjonalnego systemu. Jest to podejście zupełnie odmienne od znanego cyklu wodospadowego, gdzie analiza, projekt i budowa występują w określonej sekwencji. Warto również wspomnieć, że RUP znalazł komercyjne zastosowanie w wielu, szczególnie dużych, firmach tworzących oprogramowanie na całym świecie.

Biorąc pod uwagę specyfikę, rozmiar i w perspektywie ciągłą rozbudowę portalu e-Informatyka.pl, a z drugiej strony przedstawione powyżej ograniczenia metodyk sprawnych, autor zdecydował się na wybór i dostosowanie do własnych potrzeb procesu UP. Istotne znaczenie miał przy tym fakt, iż jest to kompletny proces, który można odpowiednio dostosować, a w szczególności „odchudzić". Równocześnie jednak wiele wartościowych, zdaniem autora, praktyk promowanych przez metodyki sprawne będzie także wykorzystywanych w realizowanym projekcie. Bodaj najlepszym przykładem jest tutaj ciągła integracja kodu będąca jedną z najważniejszych praktyk znanych z metodyki XP. Jest to zresztą praktyka wykorzystywana nie tylko w XP, ale również w wielu poważnych projektach m.in. nurtu Open Source.

Szczegółowy opis nawet tylko najpopularniejszych metodyk wytwarzania oprogramowania wykracza poza ramy tego artykułu. Autor ma jednak nadzieje, że przedstawione informacje staną się zaczynem do dyskusji nad problemem doboru właściwej metodyki do konkretnych projektów.

4.2. Podział kompetencji w zespole

Jedną z najbardziej wartościowych cech architektury Cocoon jest możliwość całkowitej separacji treści, prezentacji i logiki. To podejście promuje podział pracy i pozwala członkom zespołu współpracować w taki sposób, by nie niszczyć owoców pracy innych.



Rys. 3. Podział kompetencji członków zespołu.
Źródło: Zaadoptowane z [COCO02].

Zgodnie z przedstawionym na rys. 3. podziałem kompetencji członków zespołu można wyróżnić cztery grupy wchodzące w skład zespołu tworzącego aplikację internetową lub portal:

- Zarząd decyduje, co tworzona aplikacja internetowa powinna zawierać i jak się zachowywać. W zależności od stosowanej metodyki tworzenia oprogramowania tę grupę można podzielić dalej na podgrupy.
- Programiści logiki aplikacji są odpowiedzialni za logikę aplikacji i jej enkapsulację w spójny i intuicyjny zbiór znaczników umożliwiający dynamiczne generowanie treści. Ponieważ wymaga to integracji z technologiami dynamicznego generowania treści, zespół programistów może być dalej podzielony w zależności od wykorzystywanych technologii (np. programiści EJB czy serwletów) lub wręcz strategii organizacyjnej.
- Autorzy treści wykorzystując określone znaczniki produkują i zarządzają dokumentami XML.
- Projektanci prezentacji są odpowiedzialni za kwestie prezentacji (*look&feel*). Zespół projektantów może być dalej podzielony biorąc pod uwagę fakt, że projektowanie interfejsu użytkownika i grafiki to dwa odrębne zagadnienia, które wymagają różnych umiejętności. Projektanci interfejsu użytkownika są

odpowiedzialni za projekt interfejsu (jego użyteczność, przejrzystość, intuicyjność itd.), podczas gdy graficy mają za zadanie zaprezentować całość w atrakcyjny sposób.

Podział pracy ma oczywiście pozytywny wpływ na proces tworzenia oprogramowania. Przykładowo podczas prac nad sposobem prezentacji baza danych czy serwlety nie są potrzebne – arkusze XSLT mogą być tworzone niezależnie przy użyciu statycznych plików XML, a procesor XSLT może być w celach testowych wywoływany z linii komend. Inną interesującą opcją jest użycie narzędzi (np. takich jak XML Spy), które mogą w znacznym stopniu przyspieszyć postęp prac nad warstwą prezentacji.

5. Podsumowanie

Praktyczna realizacja portalu pozwoli na weryfikacje przyjętych koncepcji na dużym projekcie realizowanym w sposób rozproszony co ma niewątpliwe walory poznawcze. Szczególnie istotne będzie dekompozycja dużego projektu na podprojekty, jak również wykorzystanie, dostosowanie i przetestowanie procesu wytwarzania oprogramowania UP (z pewnymi praktykami znanymi np. z XP), najnowszych technologii, standardów i narzędzi informatycznych (m.in. natywna baza XML, serwer aplikacji J2EE, motor publikacji XML), koncepcji wykorzystania XML jako formatu bazowego portalu realizowanego na platformie J2EE oraz obsługi urządzeń mobilnych m.in. na platformie J2ME (*Java 2 Micro Edition*).

Autor zdaje sobie doskonale sprawę, że nie wyczerpał poruszanych tematów, a jedynie zasygnalizował wybrane zagadnienia składające się na rozległą problematykę tworzenia nowoczesnych aplikacji internetowych. Autor ma również nadzieje, że wiele zasygnalizowanych koncepcji i zagadnień stanie się inspiracją do szerokiej dyskusji i znajdzie swoje rozwinięcie w kolejnych artykułach, również na łamach e-informatyki.pl.

Literatura

[BECK01] Beck K., *Wydajne programowanie*, Mikom, 2001.

[COAD99] Coad P., Lefebvre E., De Luca J., *Java Modeling In Color With UML: Enterprise Components and Process*, Prentice Hall, 1999, rozdział 6.

[COCK02] Cockburn A., *Agile Software Development*, Addison-Wesley, 2002, rozdział 6.

[COCO02] *Understanding Apache Cocoon*, (wersja internetowa: http://xml.apache.org/cocoon/userdocs/concepts/index.html).

[FOWL00] Fowler M., *Put Your Process on a Diet*, Software Development magazine, December 2000.

[HIGH99] Highsmith J., *Adaptive Software Development: A Collaborate Approach to Managing Complex Systems*, Dorset House Publishing, 1999.

[JACO99] Jacobson I., Booch G., Rumbaugh J., *Unified Software Development Process*, Addison-Wesley, 1999.

[JEFF99] Jeffries R., Anderson A., Hendrickson C., Beck K., *Extreme Programming Installed*, Addison-Wesley, 2000.

[KRUC00] Kruchten P., *The Rational Unified Process, An Introduction*, Addison-Wesley, 2000.

[SCHW02] Schwaber K., Beedle M., *Agile Software Development with Scrum*, Prentice Hall, 2002.

NEW IDEAS OF INTERNET APPLICATIONS DEVELOPMENT ON THE EXAMPLE OF THE E-INFORMATYKA.PL WEB PORTAL

Abstract: The article presents new ideas and solutions used by the author during development of advanced e-Business Internet applications, particularly e-Informatyka.pl Web portal. The main attention was drawn to personalisation, customisation and the presentation layer which would be able to service all kinds of existing and possibly upcoming devices (personal communicators), applications and presentation formats as well as enterprise applications integration. The idea of J2EE, XML and Web services integration, latest software development methodologies, their weak and strong points and, as a result, the way of proper choice as well as the way of separating concerns of team members, was also presented.