

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Skalowalne witryny internetowe. Budowa, skalowanie i optymalizacja aplikacji internetowych nowej generacji

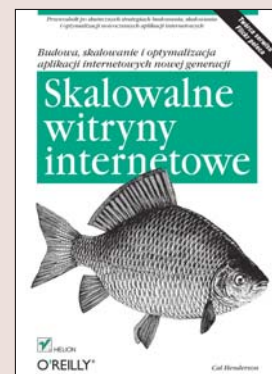
Autor: Cal Henderson

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 978-83-246-0723-5

Tytuł oryginału: [Building Scalable Web Sites:
Building, scaling, and optimizing the next
generation of web applications](#)

Format: B5, stron: 400



Naucz się tworzyć aplikacje internetowe nowej generacji i dołączyć do nurtu Web 2.0

- Chcesz tworzyć bardziej wydajne aplikacje internetowe?
- Chcesz poznać zasady projektowania skalowalnych architektur?
- Chcesz efektywnie zarządzać danymi w aplikacjach internetowych?

Oblicze internetu podlega nieustannym zmianom. Obecnie coraz częściej obok klasycznych witryn internetowych pojawiają się aplikacje internetowe, które charakteryzują się odseparowaniem warstwy danych od warstwy prezentacji. Zmiana modelu programowania wymaga przygotowania odpowiedniej platformy sprzętowej i programowej oraz zaprojektowania nowego systemu obsługi danych. Zastosowanie przy wykonywaniu tych zadań sprawdzonych strategii wykorzystywanych przez pionierów tworzących aplikacje internetowe nowej generacji pozwoli Ci zaoszczędzić czas i koszty.

Książka „Skalowalne witryny internetowe...” to zaawansowany i wszechstronny przegląd zagadnień związanych z budowaniem takich właśnie aplikacji internetowych. Pomoże Ci ona w rozwiązaniu problemów i uniknięciu pułapek czyhających na programistów witryn internetowych nowej generacji. Poznasz sprawdzone strategie projektowania architektury oprogramowania, przygotowywania środowiska programistycznego, zapewniania niezawodności aplikacji czy wydajnego zarządzania informacjami. Dowiesz się także, jak tworzyć skalowalne i łatwe w konserwacji witryny, które będą zapewniać komfort pracy niezależnie od upływu czasu i wzrostu liczby użytkowników.

- Projektowanie architektury aplikacji internetowych
- Przygotowywanie środowiska programistycznego
- Tworzenie aplikacji wielojęzycznych
- Zarządzanie bazami danych
- Integrowanie poczty elektronicznej z witrynami
- Stosowanie usług zdalnych
- Wykrywanie i rozwiązywanie problemów z wydajnością
- Skalowanie aplikacji internetowych
- Monitorowanie funkcjonowania aplikacji
- Korzystanie z interfejsów API

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl



Spis treści

Przedmowa	7
1. Wprowadzenie	15
Czym jest aplikacja internetowa?	15
Jak budujemy aplikacje internetowe?	16
Czym jest architektura?	17
Od czego należy zacząć?	18
2. Architektura aplikacji internetowej	21
Wielowarstwowa architektura oprogramowania	21
Technologie wielowarstwowe	24
Projektowanie interfejsów programowych	27
Droga od punktu A do punktu B	29
Podział na oprogramowanie i sprzęt	31
Platformy sprzętowe	31
Rozwój platformy sprzętowej	36
Nadmiarowość sprzętu	39
Sieć	40
Języki, technologie i bazy danych	43
3. Środowiska wytwarzania oprogramowania	45
Trzy naczelné zasady	45
Kontrola kodu źródłowego	46
Kompilacja w jednym kroku	66
Śledzenie błędów	77
Skalowanie modelu wytwarzania aplikacji	85
Standardy kodowania	86
Testowanie	89

4. i18n, L10n i Unicode.....	93
Umędzynarodowienie i lokalizacja oprogramowania	94
Unicode w pigułce	98
Schemat kodowania UTF-8	104
Schemat kodowania UTF-8 w aplikacjach internetowych	105
Stosowanie schematu kodowania UTF-8 w języku PHP	107
Stosowanie schematu kodowania UTF-8 w pozostałych językach programowania	108
Stosowanie schematu kodowania UTF-8 w bazach danych MySQL	109
Stosowanie schematu kodowania UTF-8 w wiadomościach poczty elektronicznej	111
Stosowanie schematu kodowania UTF-8 w skryptach języka JavaScript	113
Stosowanie schematu kodowania UTF-8 w interfejsach API	115
5. Integralność danych i bezpieczeństwo	117
Strategie zapewniania integralności danych	117
Dobre, prawidłowe i nieprawidłowe	119
Filtrowanie sekwencji UTF-8	120
Filtrowanie znaków sterujących	126
Filtrowanie kodu HTML	127
Ataki XSS	131
Wstrzykiwanie kodu języka SQL	140
6. Poczta elektroniczna	147
Otrzymywanie wiadomości poczty elektronicznej	147
Ryzyko wstrzykiwania wiadomości poczty elektronicznej do naszej aplikacji	150
Format MIME	152
Analiza składniowa prostych wiadomości MIME	154
Analiza składniowa załączników zakodowanych w trybie UU	156
Załączniki w formacie TNEF	157
Dlaczego technologie bezprzewodowe nie lubią programistów?	159
Zbiory znaków i schematy kodowania	162
Rozpoznawanie użytkowników	164
Testy jednostkowe	167
7. Usługi zdalne.....	169
Klub usług zdalnych	169
Gniazda	170
Stosowanie protokołu HTTP	173
Nadmiarowość usług zdalnych	179
Systemy asynchroniczne	182
Wymiana danych w formacie XML	187
Lekkie protokoły	192

8. Wąskie gardła	197
Identyfikowanie wąskich gardeł	197
Operacje wejścia-wyjścia	212
Usługi zewnętrzne i czarne skrzynki	225
9. Skalowanie aplikacji internetowych	241
Mit skalowania	241
Skalowanie sieci	253
Równoważenie obciążeń	256
Skalowanie bazy danych MySQL	272
Replikacja baz danych MySQL	278
Partycjonowanie bazy danych	287
Skalowanie wielkich baz danych	292
Skalowanie pamięci masowej	294
Pamięć podręczna	302
Skalowanie w pigułce	305
10. Statystyki, monitorowanie i wykrywanie usterek	307
Śledzenie statystyk aplikacji internetowej	307
Monitorowanie aplikacji	318
Alarmowanie	336
11. Interfejsy API.....	339
Kanały danych	339
Technologie mobilne	352
Usługi sieciowe	356
Warstwy transportowe interfejsów API	359
Nadużywanie interfejsów API	367
Uwierzytelnianie	371
Przyszłość	375
Skorowidz	377

Wprowadzenie

Zanim przystąpimy do projektowania i kodowania naszych przykładów, musimy zrobić krok wstecz i zdefiniować najważniejsze terminy. Warto się zastanowić, jaki jest cel naszych rozważań i czym ten cel się różni od tego, co robiliśmy do tej pory. Czytelnicy, którzy budowali już jakieś aplikacje internetowe, mogą pominąć kolejny rozdział, ponieważ prezentowany tam materiał będzie obejmował wiedzę podstawową. Z drugiej strony, czytelnicy zainteresowani ogólnym kontekstem naszych rozważań koniecznie powinni się zapoznać także ze wspomnianym rozdziałem 2.

Czym jest aplikacja internetowa?

Większość czytelników tej książki najprawdopodobniej doskonale zdaje sobie sprawę z tego, czym jest aplikacja internetowa, co wcale nie oznacza, że próba precyzyjnego zdefiniowania tego terminu jest bezcelowa, ponieważ etykieta aplikacji internetowej bardzo często jest przypisywana różnym rozwiązaniom w sposób błędny. Aplikacja internetowa nie jest ani witryną internetową, ani aplikacją w rozumieniu typowego użytkownika systemu operacyjnego. Aplikacja internetowa plasuje się gdzieś pomiędzy tymi dwoma rozwiązaniami i ma pewne cechy zarówno witryny internetowej, jak i autonomicznej aplikacji.

O ile witryna internetowa zawiera strony z danymi, o tyle aplikacja internetowa składa się zarówno z danych, jak i odrębnego mechanizmu ich dostarczania do przeglądarki. Podobnie jak entuzjaści dużej dostępności witryn internetowych zachwycają się możliwością oddzielania znaczników od stylów za pomocą plików CSS, również projektanci aplikacji internetowych szczerą się możliwością faktycznego oddzielania danych od warstwy prezentacji — dane aplikacji internetowej nie mają nic wspólnego ze znacznikami (choć same mogą tego rodzaju elementy zawierać). Komunikaty składające się na komponent aplikacji internetowej są odseparowane od znaczników. Oznacza to, że w razie konieczności wyświetlenia tych danych użytkownikowi należy wyodrębnić odpowiednie komunikaty z miejsca ich składowania (najczęściej z bazy danych), po czym dostarczyć użytkownikowi w określonym formacie i za pośrednictwem jakiegoś medium (najczęściej w formacie HTML i za pośrednictwem protokołu HTTP). Najciekawsze jest jednak to, że **nie musimy** dostarczać danych w formacie HTML — równie dobrze mogliśmy je wysłać w pliku PDF za pośrednictwem np. poczty elektronicznej.

Aplikacje internetowe nie zawierają stron w rozumieniu znanym z witryn internetowych, które z natury rzeczy składają się ze stron internetowych. O ile aplikacja internetowa może prezentować swoje dane np. za pośrednictwem dziesięciu stron internetowych, dodawanie kolejnych danych

może zwiększyć tę liczbę bez konieczności tworzenia nowych szablonów języka HTML ani dopisywania kodu źródłowego samej aplikacji. Ponieważ aplikacja internetowa może oferować takie funkcje jak przeszukiwanie (na podstawie słów wpisywanych przez użytkowników), liczba generowanych „stron internetowych” w praktyce może być nieograniczona, co oczywiście nie oznacza, że musimy w nieskończoność dopisywać niezbędny kod HTML-a. Stosunkowo niewielki zbiór szablonów i logika generowania dynamicznej zawartości pozwala nam na tworzenie niezbędnych stron na bieżąco, w odpowiedzi na takie parametry wejściowe jak adresy URL czy dane przekazywane w trybie POST.

Z perspektywy przeciętnego użytkownika aplikacja internetowa może się niczym nie różnić od witryny internetowej. Nawet analizując kod wyświetlanych stron internetowych trudno stwierdzić, czy były one generowane w sposób dynamiczny na podstawie danych składowanych np. w bazie danych, czy miały postać statycznych dokumentów HTML-a. Pewną wskazówką może być rozszerzenie pliku, co wcale nie oznacza, że nie można tego rozszerzenia sfałszować (w jedną lub drugą stronę). Aplikacja internetowa ujawnia swój prawdziwy charakter tylko tym użytkownikom, którzy modyfikują jej dane. Z reguły (choć nie zawsze) do takiej edycji wykorzystuje się interfejs HTML, ale równie dobrze można sobie wyobrazić autonomiczną aplikację zaprojektowaną z myślą o bezpośredniej lub zdalnej edycji tych danych.

Wraz z wprowadzeniem technologii AJAX (od ang. *Asynchronous JavaScript and XML*), znanej wcześniej jako zdalne wykonywanie skryptów (ang. *remote scripting*), znacznie rozbudowano model interakcji z aplikacjami internetowymi. W przeszłości interakcja użytkowników z aplikacjami internetowymi odbywała się wyłącznie za pośrednictwem stron internetowych. Użytkownik żąda od serwera zwrócenia strony internetowej, wprowadza i wysyła zmiany za pośrednictwem żądania POST protokołu HTTP i otrzymuje w odpowiedzi nową stronę potwierdzającą dokonanie zmian lub prezentującą zmodyfikowane dane. Model AJAX umożliwia nam wysyłanie modyfikacji danych w tle (bez konieczności ponownego wyświetlania strony w przeglądarce użytkownika), co zbliża aplikacje internetowe do modelu interakcji znanego z aplikacji autonomicznych.

Charakter aplikacji internetowych zmienia się dość powoli. Nie można oczywiście ignorować daleko idących różnic, które dzielą współczesne aplikacje internetowe od aplikacji implementowanych jeszcze kilka lat temu, jednak rozwoju tego rodzaju aplikacji w żadnym razie nie sposób uznać za zakończony. Projekty Gmail firmy Google i Office Live firmy Microsoft pokazują, że rynek aplikacji internetowych ewoluuje w kierunku produktów dostarczanych za pośrednictwem internetu, które łączą w sobie najlepsze cechy rozwiązań autonomicznych i internetowych. O ile tradycyjne aplikacje biurowe oferują bogate możliwości w zakresie interakcji i dużą efektywność (w szczególności krótkie czasy odpowiedzi), o tyle aplikacje internetowe mogą zapewniać swoim użytkownikom bezproblemowe i niezauważalne aktualizacje, naprawdę przenośne dane i mniejsze wymagania sprzętowe po stronie klienta. Niezależnie od stosowanego modelu interakcji z użytkownikiem, jeden aspekt pozostaje niezmienny — aplikacje internetowe są systemami, których dostęp do podstawowych danych uzyskujemy za pośrednictwem stron internetowych (z reguły strony internetowe umożliwiają także modyfikowanie tych danych), a pozostałe interfejsy mają raczej charakter uzupełniający.

Jak budujemy aplikacje internetowe?

Budowa aplikacji internetowej wymaga przygotowania co najmniej dwóch głównych komponentów: platformy sprzętowej i platformy programowej. W przypadku małych, prostych aplikacji platforma sprzętowa może się składać z pojedynczego serwera, na którym działa

serwer WWW i system zarządzania bazą danych. W tak małej skali platformy sprzętowej nie należy traktować jak istotnego komponentu naszej aplikacji, jednak wraz z rozbudową oprogramowania rola tego elementu w całym projekcie zyskuje coraz większe znaczenie. W tej książce poświęcimy wiele miejsca obu aspektom projektowania i implementowania aplikacji internetowych, wpływowi działań podejmowanych w jednym aspekcie na drugi aspekt oraz technikom wiązania obu rozwiązań (sprzętowych i programowych) celem tworzenia możliwie efektywnych architektur.

Programiści, którzy do tej pory mieli do czynienia wyłącznie z aplikacjami internetowymi małej skali, mogą się zastanawiać, po co w ogóle zawracać sobie głowę projektem platformy, skoro można z powodzeniem stosować gotowe rozwiązania. W przypadku niewielkich aplikacji takie podejście rzeczywiście zdaje egzamin. Oszczędzamy czas i pieniądze, by ostatecznie otrzymać działającą aplikację, które całkowicie spełnia nasze oczekiwania. Problem pojawia się dopiero w aplikacjach większej skali — gotowe rozwiązania umożliwiające budowę takich aplikacji jak *Amazon.com* czy *Friendster.com* po prostu nie istnieją. Chociaż implementacja zbliżonej funkcjonalności wcale nie musi być trudna, zagwarantowanie właściwego funkcjonowania tego rodzaju oprogramowania w środowisku obejmującym miliony produktów i miliony użytkowników bez zapewnienia odpowiedniej platformy sprzętowej wymagałoby daleko idącego dostosowania i optymalizacji naszych rozwiązań do konkretnych potrzeb. Okazuje się, że wszystkie największe aplikacje funkcjonujące obecnie w internecie są „szyte na miarę” z jednego istotnego powodu — żadne inne podejście nie gwarantowałoby odpowiedniej skalowalności w warunkach ograniczonych możliwości budżetowych.

Wspominaliśmy już, że rdzeniem każdej aplikacji internetowej jest jakiś zbiór danych, które są udostępniane użytkownikowi i które z reguły mogą być przez tego użytkownika modyfikowane. Projektując część oprogramowania naszej aplikacji musimy zdecydować o sposobie składowania tych danych (o schemacie ich reprezentacji np. w bazie danych), o sposobie uzyskiwania dostępu i modyfikowania danych (implementowanym przez logikę biznesową) oraz o sposobie ich prezentowania użytkownikom (implementowanym przez logikę interakcji). Wszystkie te komponenty omówimy w rozdziale 2., gdzie przeanalizujemy występujące pomiędzy nimi zależności oraz dalsze elementy, które składają się na każdy z tych trzech komponentów. Dobry projekt aplikacji zawsze schodzi w dół od najwyższego, najbardziej ogólnego poziomu do konkretnych definicji architektury oprogramowania i sprzętu, komponentów składających się na naszą platformę oraz funkcjonalności implementowanej w poszczególnych warstwach.

Niniejsza książka w założeniu ma być praktycznym przewodnikiem po projektowaniu i konstruowaniu aplikacji wielkiej skali. Po lekturze tej publikacji Czytelnik powinien wiedzieć, jak należy projektować aplikację i jej architekturę, jak należy skalować gotowe systemy i jak powinny przebiegać procesy implementowania i wdrażania w życie tych projektów.

Czym jest architektura?

Temat architektur aplikacji jest bardzo modny, co nie znaczy, że wszyscy, którzy posługują się tym terminem, wiedzą, co on oznacza. Kiedy architekt projektuje dom, jego zadanie jest dość precyzyjnie określone: musi zebrać wymagania, przeanalizować różne opcje i sporządzić stosowny szkic. Kiedy ekipa budowlana zapozna się z planami architekta, będzie oczekiwała kilku prostych rzeczy: dom musi przetrwać wichury, zapewniać ochronę przed deszczem i zimą oraz oferować swoim mieszkańcom właściwe oświetlenie. Na tym powinniśmy zakończyć nasze rozważania poświęcone budowie domów, ponieważ podobieństwa pomiędzy tymi architektu-rami są tylko iluzoryczne.

Po pierwsze, gdyby budynek był jak oprogramowanie, architekt musiałby się angażować w cały proces jego budowy, od wylania fundamentów po założenie instalacji elektrycznych i sanitarnych. Kiedy projektowanie i budowa domu dobiegnie końca, architekt musiałby przystąpić do wykańczania i urządzania kilku pomieszczeń w oczekiwaniu na przybycie części przyszłych mieszkańców, którzy zamieszkaliby w nowym domu przed jego ostatecznym wykończeniem. Bezpośrednio przed zakończeniem budowy do budynku wprowadziłoby się mnóstwo innych osób, które na własnej skórze sprawdzałyby, jak zastosowane rozwiązania sprawdzają się w praktyce. Nowi mieszkańcy stawialiby jednak dodatkowe wymagania — chcieliby mieć do dyspozycji większą liczbę sypialni, basen, piwnicę itd. Architekt musiałby niezwłocznie przystąpić do projektowania nowych pomieszczeń i atrakcji, modyfikując swój oryginalny projekt. W czasie realizacji nowych inwestycji wszyscy dotychczasowi mieszkańcy pozostawaliby jednak w przebudowywanym domu. Ciągłe korzystaliby z istniejących udogodnień, narzekając na hałas i pył powodowane przez prace ekip budowlanych. Co więcej, wbrew wszelkiej logice do wciąż rozbudowywanego budynku wprowadziliby się kolejni mieszkańcy. Po zakończeniu wprowadzania modyfikacji nowo przybyli mieszkańcy natychmiast zaczęliby zgłaszać następne wnioski o rozbudowę domu o kolejne pomieszczenia i dodawanie dalszych atrakcji.

Kluczową cechą dobrego architekta aplikacji jest zdolność przewidywania tego rodzaju problemów od samego początku. Gdyby architekt naszego hipotetycznego domu rozpoczął od budowy wielkiego gmachu z bardzo skomplikowanymi instalacjami, w żaden sposób nie mógłby sprostać dalszym żądaniom. W chwilę po zakończeniu budowy mieszkańcy wyprowadziliby się w inne miejsce, gdzie mogliby uczestniczyć w przebudowie zamieszkiwanego domu zgodnie ze swoimi potrzebami. Z utratą mieszkańców musielibyśmy się liczyć także wtedy, gdyby rozbudowa naszego domu zajmowała zbyt wiele czasu. Musimy wiedzieć, jak rozpocząć prace we właściwej skali, by w przyszłości można było w miarę bezproblemowo ten dom rozbudowywać.

Z powyższych rozważań w żadnym razie nie wynika, że architekt musi od samego początku nie popełniać żadnych błędów w przyjmowanych założeniach. Skalowanie typowej aplikacji polega między innymi na analizie i dostosowywaniu wszelkich aspektów pod kątem ich ewentualnego dopasowania do nowych wymagań. Nie ma w tym nic złego — zadaniem architekta aplikacji jest tylko minimalizowanie czasu potrzebnego do modyfikowania poszczególnych komponentów oraz możliwie precyzyjne przygotowywanie projektu początkowego i projektów niezbędnych zmian.

Od czego należy zacząć?

Aby przystąpić do projektowania i budowy pierwszej aplikacji internetowej wielkiej skali, musimy dysponować czterema niezbędnymi elementami. Po pierwsze, będziemy potrzebowali koncepcji. Z reguły jest to element najtrudniejszy, ponieważ nie ma wiele wspólnego z tradycyjnymi działaniami, do których przywykła większość inżynierów. Techniki i technologie prezentowane w tej książce można co prawda stosować podczas realizacji małych projektów, jednakże w przypadku większych projektów wymagających zaangażowania wielu programistów ich wykorzystywanie ma charakter opcjonalny. Jeśli dysponujemy aplikacją, która nie została jeszcze wdrożona w docelowym środowisku lub jest stosunkowo mała i wymaga skalowania, możemy przyjąć, że najtrudniejsze za nami — wówczas możemy od razu przystąpić do projektowania rozwiązań w większej skali. Jeśli dysponujemy aplikacją wielkiej skali, warto przestudiować tę książkę, by przekonać się, czy do tej pory postępowaliśmy właściwie, i przygotować się do dalszego rozwoju naszego produktu.

Kiedy już będziemy mieli koncepcję tego, co chcemy zbudować, będziemy musieli znaleźć ludzi, którzy tę koncepcję zrealizują. O ile małe i średnie aplikacje mogą być budowane przez pojedynczych inżynierów, o tyle konstruowanie wielkich aplikacji zawsze wymaga zaangażowania większych zespołów. W grudniu 2005 roku oprogramowanie *Flickr.com* obejmowało ponad 100 tys. linii kodu, 50 tys. linii kodu szablonów oraz 10 tys. linii kodu języka JavaScript. Utrzymywanie i właściwa konserwacja tak dużej ilości kodu przekracza możliwości pojedynczego inżyniera, zatem odpowiedzialność za poszczególne obszary aplikacji należy rozłożyć pomiędzy różne osoby lub zespoły. Techniki zarządzania procesami wytwarzania i rozwoju oprogramowania, które wymagają zaangażowania wielu programistów, omówimy w rozdziale 3. Budowa aplikacji w zespole dowolnej wielkości jest możliwe tylko w odpowiednim środowisku wytwarzania, które zapewnia mechanizmy oceny postępu prac — dysponowanie tym środowiskiem jest warunkiem koniecznym do osiągnięcia oczekiwanych rezultatów. Środowiskami wytwarzania oprogramowania i środowiskami zarządzającymi kolejnymi etapami prac, a także odpowiednimi narzędziami programowania zajmiemy się w rozdziale 3. tej książki. Na razie możemy przyjąć, że nasze potrzeby ograniczają się do komputera z działającym serwerem WWW i systemem zarządzania bazą danych.

Najważniejszym elementem, którego będziemy potrzebowali, jest właściwa metoda omawiania i rejestrowania procesu wytwarzania oprogramowania. Szczegółowa specyfikacja może być śmiertelnie nudna, ale brak jakiegokolwiek dokumentacji może doprowadzić do prawdziwej katastrofy. W przypadku bardzo niewielkich zespołów wystarczy dosłownie jeden zeszyt lub tablica (warto ją fotografować, aby w przyszłości dysponować trwałą kopią nanoszonych zapisów). Programiści, którzy nie mogą się rozstać z komputerem na tyle długo, by wziąć do ręki długopis lub flamaster, mogą z powodzeniem korzystać z takich narzędzi jak Wiki. W pracach większych zespołów Wiki doskonale sprawdza się w roli narzędzia do zarządzania specyfikacjami wytwarzanego oprogramowania i sporządzanymi notatkami — w ten sposób można umożliwić wszystkim zaangażowanym programistom dodawanie i edycję notatek o realizowanych zadaniach.

O ile metodyka kaskadowego wytwarzania oprogramowania może się okazać skuteczna w przypadku wielkich, ale jednolitych, monolitycznych aplikacji internetowych, wielu programistów tego rodzaju aplikacji preferuje szybkie i bardziej elastyczne podejście iteracyjne. Kiedy opracowujemy i rozwijamy projekt aplikacji internetowej, z natury rzeczy nie chcemy podejmować kroków, które w przyszłości ustawią nas w narożniku. Powinniśmy dbać o to, by każda podejmowaną decyzję można było stosunkowo szybko zmienić, jeśli okaże się, że nie zbliża nas do założonego celu — nowe elementy funkcjonalności można projektować na poziomie bardzo ogólnych rozwiązań, implementować i iteracyjnie udoskonalać jeszcze przed ostatecznym wdrożeniem w ramach większej aplikacji (choć takie doskonalenie wcale nie musi się kończyć wraz z wdrożeniem). Stosowanie takich lekkich narzędzi jak Wiki do zarządzania bieżącą dokumentacją gwarantuje nam niezbędną elastyczność — nie musimy poświęcać sześciu miesięcy na sporządzanie specyfikacji i kolejnego roku na implementowanie przyjętych założeń. Możemy opracować specyfikację w ciągu jednego dnia, po czym zaimplementować odpowiednie rozwiązania w kilka dni. W takim przypadku zyskamy całe miesiące, w ciągu których będziemy mieli mnóstwo czasu na iteracyjne usprawnianie tej wstępnej wersji. Im wcześniej będziemy dysponowali działającym kodem (choćby wstępną, ograniczoną wersją), tym szybciej będziemy mogli przystąpić do odkrywania i rozwiązywania ewentualnych problemów w naszym projekcie i tym mniej czasu stracimy, jeśli zrezygnujemy z dotychczasowej strategii na rzecz zupełnie innego podejścia. Ostatni czynnik jest szczególnie ważny — im mniej czasu poświęcimy na

implementację i rozwój pojedynczej jednostki funkcjonalności (które powinny być możliwie niewielkie i proste), tym mniejsze inwestycje taka jednostka pochłonie i tym łatwiej będzie ją odrzucić w razie przyjęcia innej koncepcji. Znacznie więcej informacji na temat metodyk i technik wytwarzania oprogramowania można znaleźć w książce Steve’a McConnella pt. *Rapid Development* (Microsoft Press).

Z długopisem w dłoni i narzędziem Wiki na ekranie komputera możemy przystąpić do projektowania architektury naszej aplikacji i rozpocząć implementowanie rozwiązań, które podbiją świat.