Co każdy kierownik projektu powinien wiedzieć o programistach i ich pracy?

Niezbędnik wiedzy technicznej dla kierowników projektów programistycznych

bns it

Co każdy kierownik projektu powinien wiedzieć o programistach i ich pracy?

Niezbędnik wiedzy technicznej dla kierowników projektów programistycznych

Wstęp dla kierowników projektów

Drogi Kierowniku Projektu!

Wierzymy, że pomagając zespołom projektowym umiejętnie łączyć wiedzę techniczna z kompetencjami miękkimi, przyczyniamy się do wzrostu ich efektywności i poprawy komfortu pracy.

Jesteś kluczowym członkiem takiego zespołu. Chcemy, aby ten podręcznik pomógł Ci dowiedzieć się więcej o specyfice pracy programistów, a przez to podniósł jakość komunikacji i współpracy w Twoim zespole.

Życzymy wspaniałych projektów!

Zespół BNS IT

Wstęp dla programistów, którzy tu przypadkowo zajrzą

Celem tego podręcznika jest usprawnienie komunikacji i zwiększenie jakości współpracy w zespole projektowym. Nie jest nim więc stworzenie kompendium wiedzy na temat inżynierii oprogramowania. Wiele zagadnień zostało pominiętych, uproszczonych lub wręcz mocno nadwyrężono ich definicję. Wiemy jednak, że działania te były konieczne, aby zbudować pomost wiedzy technologicznej łączącej dwie bardzo różne grupy ludzi – programistów i kierowników projektów.

Jeśli uważasz, że któryś fragment wymaga zmiany lub masz pomysły na uzupełnienie podręcznika – napisz do nas na adres bnsit@bnsit.pl z tematem *Uwagi do niezbędnika dla PM.* Napisz, co chciałbyś, aby PM, z którym współpracujesz wiedział o Tobie?

Życzymy samych grinbarów!

Zespół BNS IT

Architektura systemów informatycznych

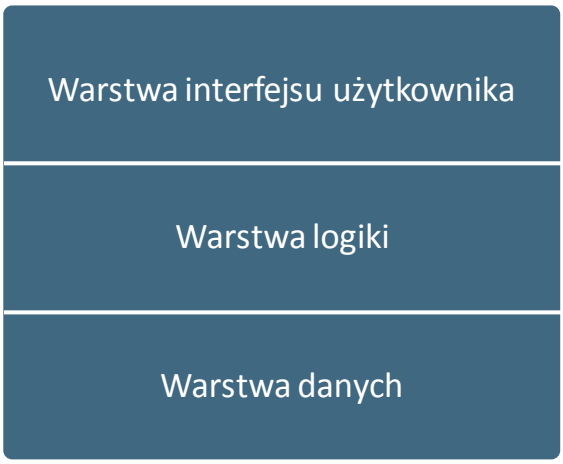
Słowo architektura nie przypadkowo pojawia się w kontekście systemów informatycznych. Inżynieria oprogramowania czerpie bardzo wiele z budownictwa. W dość odległej, jak na IT, przeszłości Christopher Alexander opublikował książkę[1], w której zastanawiał się, co sprawia, że jedne budynki są uznawane za ładniejsze i bardziej funkcjonalne, a inne za mniej i czy istnieją jakieś powtarzalne wzorce na ładne i funkcjonalne budynki. Książka spotkała się z zaskakująco dużym odzewem ze strony

[1] Alexander, Christopher (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press

środowiska programistów. Bez przesady można stwierdzić, że była ona jedną z bezpośrednich przyczyn dynamicznego rozwoju stosunkowo młodej dziedziny wiedzy – inżynierii oprogramowania.

Trzy części systemu

Z jakiś powodów liczba *3* nieustannie przewija się w historii ludzkości (trzech muszkieterów, trójpolówka, trzej amigos, trzy korony, itd.). Dlaczego w programowaniu miałoby być inaczej?

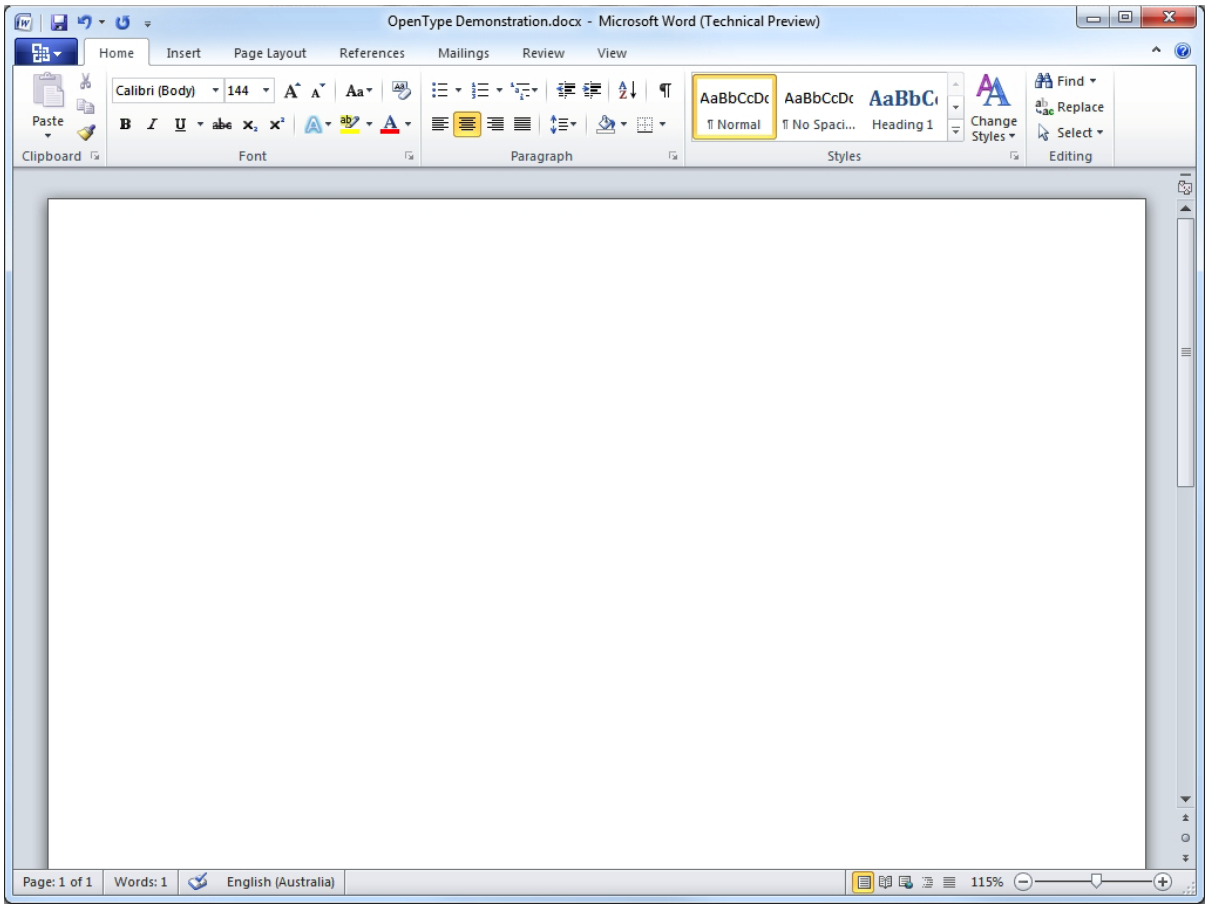


Rys. 1 System trójwarstwowy

Przyjęło się uważać system złożony z trzech części, nazywanych warstwami, za ideał piękna i elastyczności. Od tamtego momentu większość programistów jest święcie przekonana, że każdy system powinien być zbudowany z trzech części (warstw) (rys. 1):

- Warstwy interfejsu użytkownika, która ładnie wygląda
- Warstwy logiki, która reaguje na to, co zrobi użytkownik
- Warstwy danych, która przechowuje wprowadzone przez użytkownika

Spójrzmy na program Microsoft Word z perspektywy warstw.



Rys. 2 Microsoft Word

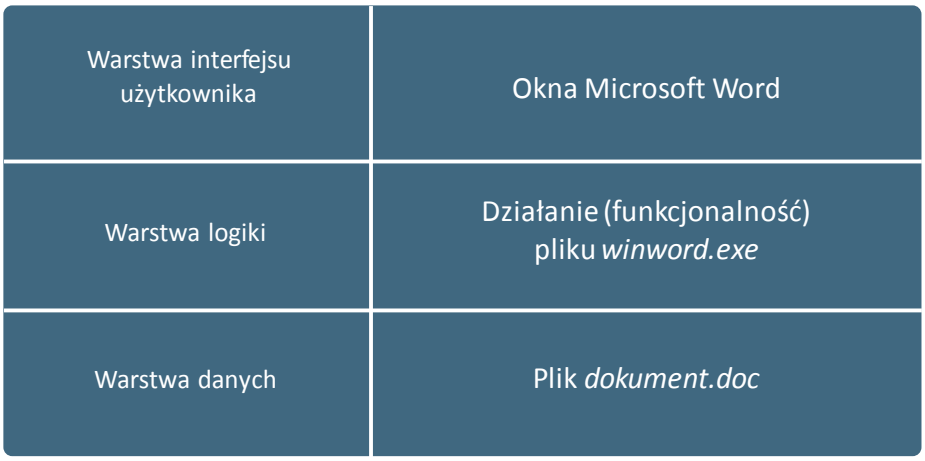
Program MS Word jest niczym innym jak plikiem na dysku o nazwie *winword.exe,* mieszczącym się w katalogu C:*Program Files**Microsoft Office**Office12*:

WINWORD.EXE

Rys. 3 Plik winword.exe

Po uruchomieniu programu oczom ukazuje znajomy widok popularnego „łorda”.

Okienka i wszystko to, na co można kliknąć należą do *warstwy interfejsu użytkownika*. Za to, co robi program w odpowiedzi na kliknięcie (coś robi prawda? powiększa czcionkę, wstawia obrazek albo tabelkę) odpowiada *warstwa logiki*. A gdzie warstwa danych? Warstwą danych jest plik, który edytujesz np. *raport.doc*. Jak widzisz, warstwa danych jest oddzielona od poprzednich dwóch warstw, które programiści upakowali w jednym pliku *winword.exe*.



Rys. 4 MS Word w podziale na warstwy

Być może zapytasz: czy warstwy mogą być od siebie oddzielone tak jak w programie MS Word? Czy nie powinny być wszystkie razem w jednym pliku? Oczywiście mogą być w jednym pliku, ale programiści bardzo lubią je oddzielać i trzymać osobno. Głównym powodem jest zwiększenie elastyczności oprogramowania. W końcu lepiej mieć edytor tekstu, który obsługuje wiele różnych plików niż tylko jeden, prawda?

Jak wygląda warstwa w środku?

Na najbardziej elementarnym poziomie warstwy są dosłownie *napisane*. Programiści piszą warstwy używając *języka programowania* (np. *Java, PHP, C#, C++,* i wiele innych).

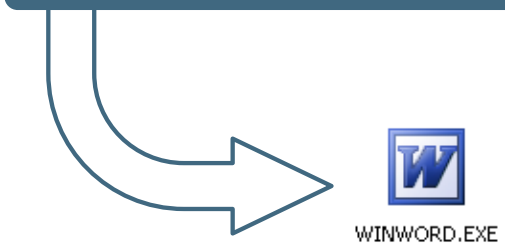
```
private boolean running = false;
private Map<String,Command> commands = new HashMap<String,Command>();
private ApplicationModel model = new ApplicationModel();


public void initialize() {
        commands.put("hello", new HelloCommand());
        commands.put("help", new HelpCommand(commands));
        commands.put("exit", new ExitCommand(this));
        commands.put("save", new SaveModelCommand(model));
        commands.put("load", new LoadModelCommand(model));
        commands.put("building_report", new BuildingReportCommand(model));
        commands.put("add_building", new AddBuildingCommand(model));
        commands.put("add_equipment", new AddEquipmentCommand(model));
}

public void run() {
        running = true;

        Scanner scanner = new Scanner(System.in);

        new HelloCommand().execute("");
```

Rys. 5 Fragment kodu źródłowego w języku Java

Języki programowania są konstruowane na wzór języka ludzkiego (naturalnego). Mają swoją składnię (słowa, których wolno używać) oraz gramatykę (reguły, według których należy konstruować złożone zdania). Taka pisanina programistów nazywana jest kodem źródłowym. Nie jest to jeszcze gotowy

program, jeszcze nie można go używać. Gdy już programiści zakończą etap pisania kodu źródłowego, to rozpoczyna się proces przekształcania kodu w plik, który użytkownik może uruchomić.

```
box_for_display(t_accno) ;
int row=7, flag ;
fstream file ;
file.open(„data.DAT", ios::in) ;
while (file.read((char *) this, sizeof(account)))
{
        if (accno == t_accno)
        {
                flag = 0 ;
                delay(10) ;
                cout <<balance ;
                row++ ;
                if (row == 23)
                {
                        flag = 1 ;
                        row = 7 ;
                        gotoxy(4,24) ;
                        getch() ;
                        clrscr() ;
                        box_for_display(t_accno) ;
                }
        }
}
```



Rys.6 Przekształcanie kodu źródłowego do pliku, który można uruchomić

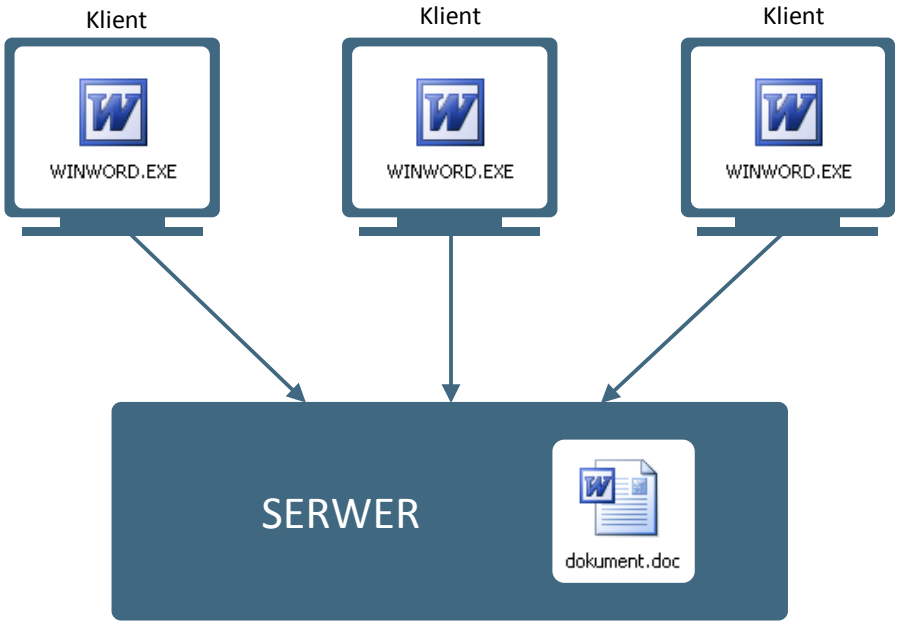
Jak wygląda plik, który można uruchomić?

W zależności od użytej technologii może wyglądać bardzo różnie. W przykładzie jest to plik typu *exe* (od *executable)*, który jest dedykowany dla systemów operacyjnych Windows. Inne rodzaje plików to np.: *bin, jar, war, ear, air*. O niektórych z nich będzie jeszcze mowa w tym opracowaniu.
Najistotniejsze jest to, że postać pliku do uruchomienia może być różna dla różnych technologii, np.: plik dla Windows nie będzie działał na komputerze Macintosh, natomiast aplikacja lokalizacyjna z telefonu prawdopodobnie nie będzie działać na laptopie.

Dalsze rozwarstwienie

Pierwszy kłopot z programem MS Word pojawia się wtedy, gdy kilku użytkownikom przychodzi na myśl, aby wspólnie pracować nad jednym plikiem. Częstą praktyką jest przesyłanie sobie pliku mailem oraz korzystanie z wbudowanych funkcjonalności zapisywania wersji pliku oraz recenzji treści. O ile przy dwóch lub trzech użytkownikach nad zmianami w pliku można jeszcze zapanować, to przy dziesięciu taka praca staje się niezmiernie uciążliwa.

Idealnym rozwiązaniem byłoby, aby użytkownicy rzeczywiście pracowali na jednym pliku. Każdy z użytkowników mógłby mieć na swoim komputerze zainstalowany program MS Word, a wspólny plik umieszczony by był w jakimś konkretnym miejscu. To konkretne miejsce zazwyczaj nazywane jest serwerem.



Rys. 7 Architektura klient-serwer

W ten sposób działa *architektura klient-serwer*. Programy MS Word zainstalowane na komputerach użytkowników nazywane są *klientami* (w mianowniku liczby mnogiej: *klienty* – dla odróżnienia od klientów-ludzi). Natomiast miejsce, na którym leżą dokumenty nazywane jest *serwerem*[2].

Tak jak wspominaliśmy wcześniej warstwy znów się nieco rozdzieliły.

Warstwa interfejsu użytkownika	Okna programu Microsoft Word w pliku winword.exe
Warstwa logiki	Działanie (funkcjonalność) pliku *winword.exe* przeniesiona na serwer
Warstwa danych	Plik *dokument.doc*

Rys. 8 MS Word w architekturze klient-serwer w podziale na warstwy

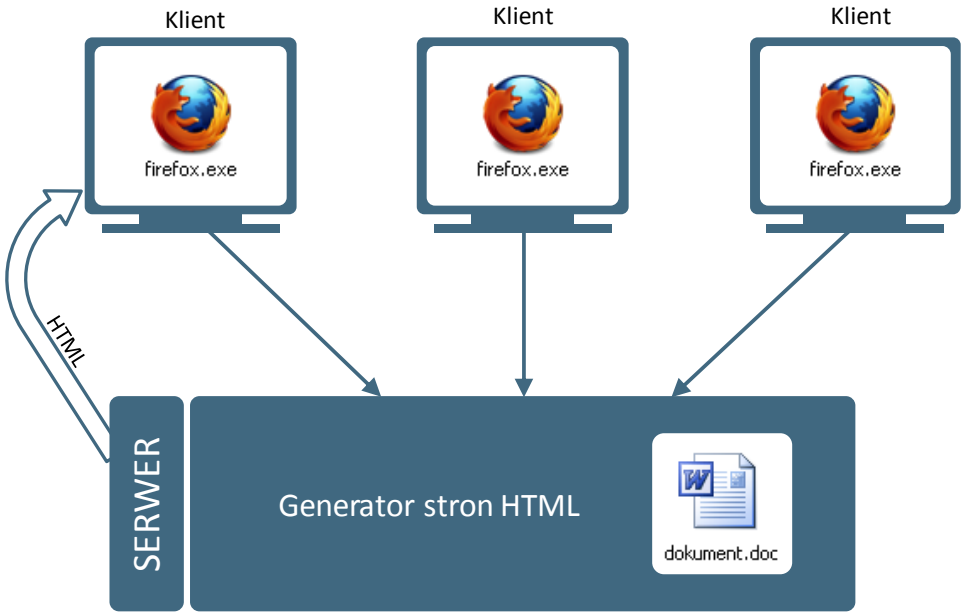
Interfejs użytkownika jest w pliku uruchomieniowym MS Word (albo bardziej fachowo *po stronie klienta)*. Warstwa logiki podzieliła się na dwie części: część po stronie klienta, która odpowiada za obsługę podstawowych „kliknięć" użytkownika oraz część po stronie serwera, której zadaniem jest zarządzanie dostępem do dokumentów. Warstwa danych powędrowała na serwer (tu już chodzi o fizyczny komputer; patrz: przypis 2).

[2] Słowo *serwer* jest używane w dla określenia dwóch rzeczy: komputera (maszyny) na której leżą dane udostępniane klientom oraz oprogramowania zainstalowanego na tym komputerze, które obsługuje to udostępnianie (np.: kontroluje prawa dostępu, zapisuje i odczytuje pliki itd.). W tym wypadku mamy na myśli oprogramowanie.

Czy można bez instalowania?

Wspólną cechą omówionych rozwiązań jest to, że trzeba instalować program MS Word na komputerze użytkownika. Jest to uciążliwe zadanie, zwłaszcza gdy trzeba to zrobić dla wszystkich pracowników firmy?

Najlepiej, aby użytkownik nie musiał nic instalować na komputerach użytkowników, a jedynie na serwerze. Łatwiej zmienić coś na jednym komputerze niż na stu. Co zatem powinno pełnić rolę klienta na komputerze użytkownika? Z pewnością coś co prawdopodobnie znajduje się na każdym komputerze. Wybór padł na przeglądarkę internetową, a tworzenia interfejsu użytkownika wykorzystano strony internetowe. W ten sposób narodziła się *architektura aplikacji webowej/łebowej* (od *web*).



Rys. 9 Architektura aplikacji webowej

Podstawową zaletą aplikacji webowych jest łatwość zarządzania nią: instalowania, aktualizacji, poprawiania błędów, ponieważ wystarczy to zrobić tylko w jednym miejscu – na serwerze.

Dlaczego akurat strony internetowe i HTML?

Ponieważ za sprawą Internetu strony WWW stały się bardzo popularne, każdy komputer i większość telefonów są wyposażone w przeglądarkę stron, język do tworzenia stron internetowych (HTML) został ustandaryzowany przez ogólnoświatowe konsorcjum W3C, a każdy producent przeglądarki stara się przestrzegać tego standardu, aby nie wypaść z gry. Twórcy systemów informatycznych postanowili zatem wykorzystać coś, co już istnieje, zamiast tworzyć i popularyzować zupełnie nowe rozwiązanie.

Kolejny raz mamy do czynienia z przetasowaniem warstw w systemie. Tym razem podziałowi uległa warstwa interfejsu użytkownika. Zauważ, że strony internetowe (albo strony HTML) są statyczne, to znaczy mają tylko wygląd, nie mają zachowania[3]. Za aplikacją desktopową było inaczej – okienka zarówno miały wygląd jak i zachowanie – potrafiły wykonać jakieś złożone zadanie w odpowiedzi na kliknięcie. Strony internetowe tylko wyglądają.

[3] Obecnie technologie takie jak JavaScript, Ajax pozwalają dodawać pewne zachowanie do stron HTML, aby oszczędzić na długotrwałym przesyłaniu danych pomiędzy przeglądarką a serwerem. Historycznie strona HTML jest jednak statyczna, nie posiada zachowania.

Jak wygląda strona HTML w środku?

Możesz to łatwo sprawdzić. Kliknij prawym przyciskiem myszy na stronie i wybierz pozycję *Pokaż źródło*. Zobaczysz wtedy tekst podobny do tego z rysunku nr 10. Tekst ten opisuje wyłącznie wygląd strony, brak tam zachowania.

```
<html>
<head>
<title>div element example</title>
</head>
<body>
        <div id="Layer1" style="position:absolute;
                width:300px;
                 height:294px;
                 background-color:#FFFFCC;
                 layer-backgroundcolor:#FFFFCC;
                 z-index:1">
        <p><img src="yourImage.jpg" alt="" width="300"  height="225"></p>
        <p align="center">This div element contains an image plus a center aligned text.</p>
        </div>
        <div id="Layer2" style="position:absolute;
                width:315px;
                height:174px;
                background-color:#EEEEEE;
                z-index:2">
         <p>Text.</p>
         </div>
</body>
</html>
```

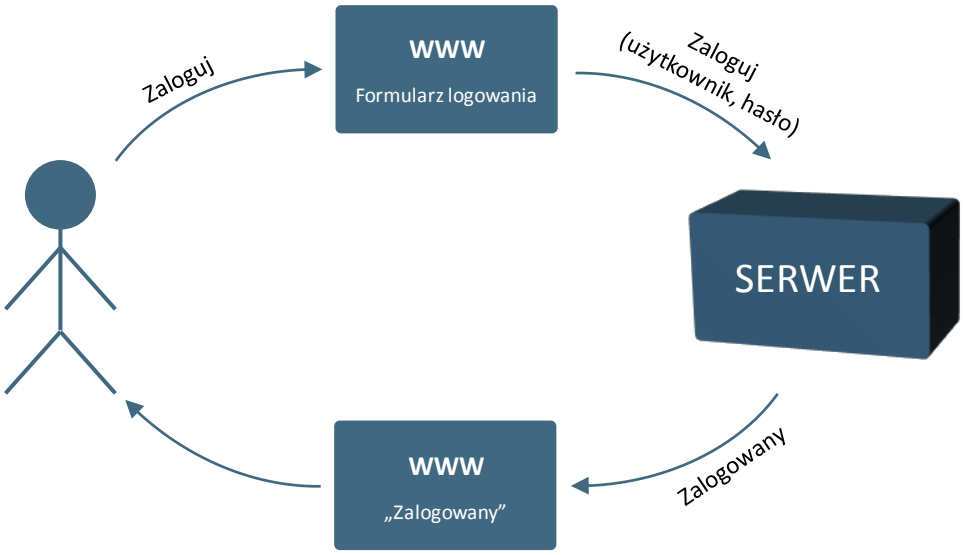
Rys. 10 Fragment kodu źródłowego w języku HTML

Skąd się bierze zachowanie stron HTML?

Tak jak wspomnieliśmy – strona HTML opisuje tylko wygląd interfejsu użytkownika. W momencie, gdy użytkownik kliknie na stronie jakiś odsyłacz albo przycisk (np.: *Zaloguj*), za każdym razem przeglądarka wysyła informację o tym do serwera. Serwer interpretuje tę informację, generuje nową stronę HTML (za pomocą specjalnego generatora) i odsyła ją powrotem do przeglądarki użytkownika.

Ostatecznie cała ta komunikacja przypomina pokaz slajdów: serwer pokazuje slajd (stronę), użytkownik klika, serwer generuje nowy slajd (nową stronę). Można powiedzieć, że strona HTML tylko „udaje" zachowanie. Użytkownik ma wrażenie, że coś się dzieje, gdy tym czasem po każdym kliknięciu otrzymuje zupełnie nową stronę wygenerowaną przez serwer, nawet jeżeli zmieniła się tam tylko jedna litera[4].

[4] Obecnie technologie pozwalają na tzw. *partial page rendering* czyli generowanie przez serwer tylko fragmentu strony HTML zamiast całej.



Rys. 11 „Udawanie” zachowania przez aplikację webową

Zatem warstwa interfejsu użytkownika została podzielona na dwie części: stronę HTML – umieszczoną w przeglądarce na komputerze użytkownika (a więc po stronie klienta) oraz generator stron HTML po stronie serwera. Warstwa logiki znajduje się w całości na serwerze. Warstwa danych nie uległa zmianie, wciąż są to dokumenty udostępniane użytkownikom.

Warstwa interfejsu użytkownika	Strona HTML, generator stron HTML na serwerze
Warstwa logiki	Na serwerze
Warstwa danych	Dokumenty

Rys. 12 Aplikacja webowa w podziale na warstwy

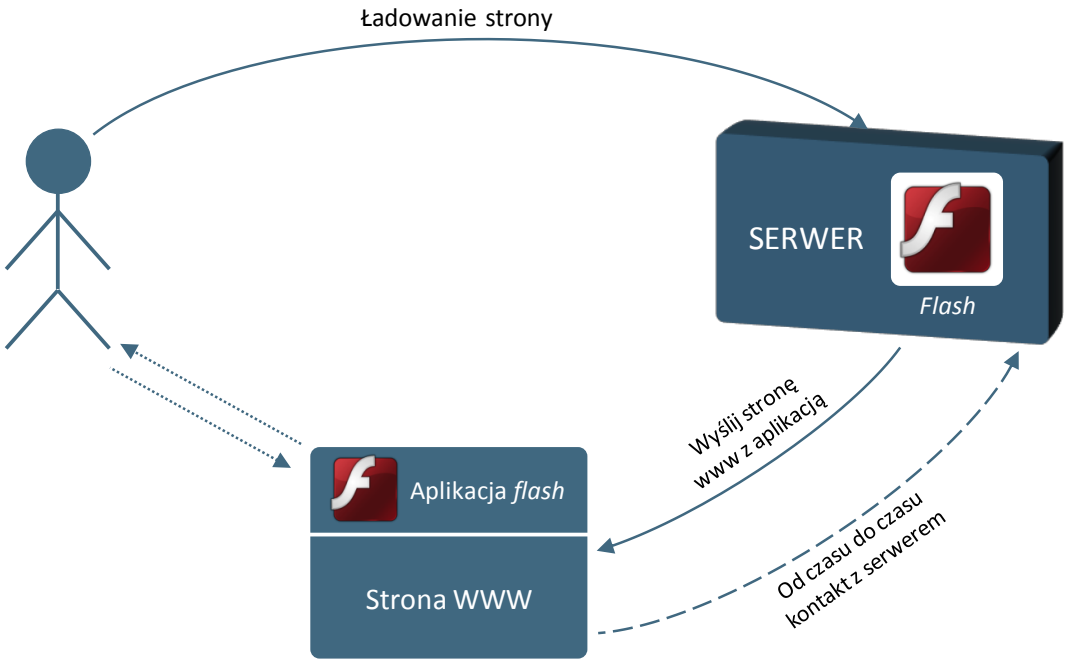
Ponieważ strona internetowa w przeglądarce użytkownika pełni rolę klienta i jednocześnie nie ma zachowania, programiści nazywają ją *cienkim klientem* (*thin/lean/slim client*).

Czy istnieje gruby klient?

Cienki klient ma jedno poważne ograniczenie: z każdą aktywnością użytkownika wiąże się wymiana danych pomiędzy przeglądarką a serwerem, a to może być bardzo czasochłonne. Twórcy systemów informatycznych wpadli na pomysł przeniesienia części logiki do cienkiego klienta (na stronę HTML).

Aby zrealizować ten pomysł potrzeba było bardziej wyrafinowanego rozwiązania niż prosta strona internetowa. Jednym z pomysłów było wykorzystanie technologii *Adobe Flash*. Technologia ta pozwala stworzyć małe objętościowe aplikacje, które pozwalają wchodzić w interakcję z użytkownikiem. Ze względu na mały rozmiar można je bez problemu użyć na stronie internetowej.

Idea tego rozwiązania jest następująca: klient po raz pierwszy wpisuje adres strony internetowej, serwer generuje dla niego stronę oraz jednocześnie wstawia tam gotową aplikacje *flash*, którą odsyła do przeglądarki razem ze stroną. Programiści mówią, że aplikacja (albo komponent) *flash* jest *osadzona* na stronie HTML. Od tego momentu aplikacja *flash* znajduje się już na komputerze użytkownika (po stronie klienta), potrafi wykonać dla niego niektóre rzeczy (logikę), a w przypadku bardziej złożonej logiki kontaktuje się z serwerem.



Rys. 13 Aplikacja *flash* osadzona na stornie internetowej

Tego rodzaju aplikację *flash* programiści nazywają grubym klientem (*fat client, rich client*), ponieważ pozwala zrealizować dużo więcej niż zwyczajna strona HTML.

Warstwa interfejsu użytkownika	Wygląd – aplikacja *flash* osadzona na stronie internetowej
Warstwa logiki	Logika klienta, logika serwera
Warstwa danych	Dokumenty

Rys. 14 Aplikacja webowa z grubym klientem w podziale na warstwy

Rysunek 14 pokazuje jak zmienił się rozmieszczenie elementów w poszczególnych warstwach systemu.

Przedstawiony pomysł doprowadził do rozwinięcia się nurtu technologicznego o nazwie *RIA* (*Rich Internet Application*), który zajmuje się dostarczaniem rozwiązań do tworzenia aplikacji webowych z grubym klientem. W tym nurcie powstają platformy do tworzenia tego typu aplikacji. Jedną z darmowych jest *OpenLaszlo*[5], a jej komercyjnym odpowiednikiem jest *Adobe Flex*.

Technologia *flash*, czy platforma *Adobe Flex* nie są jedynymi możliwościami jeśli chodzi o tworzenie aplikacji z grubym klientem. Inne możliwości to: *Microsoft Silverlight, JavaFX*.

Historia kołem się toczy

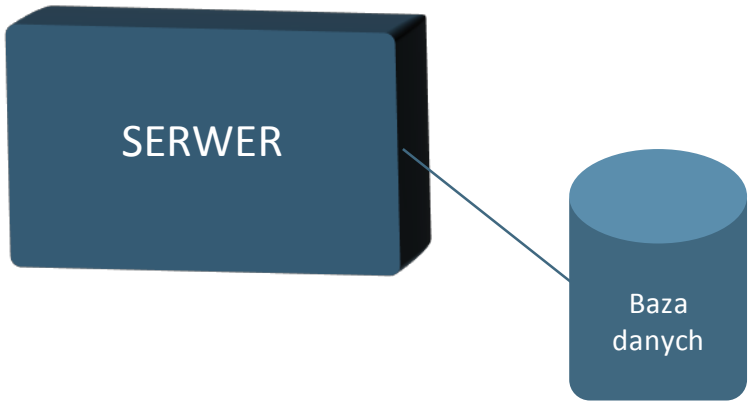
Koncepcja RIA wciąż się rozwija, powstają kolejne platformy dla programistów wspomagające programistów w tworzeniu aplikacji webowych z grubym klientem. Jedną z nich jest *Adobe Air*, która umożliwia tworzenie aplikacji z grubym klientem zarówno w technologii *flash* jak i HTML/AJAX.

Ciekawe jest, że aplikacje *Adobe Air* mogą być uruchamiane bez osadzania na stronie internetowej – jako zwyczajne aplikacje desktopowe[6]. W ten sposób historia zatoczyła koło – wyszliśmy od aplikacji desktopowej, ulepszaliśmy ją, przesuwaliśmy funkcjonalności pomiędzy różnymi warstwami, poznaliśmy aplikację webową oraz jej ograniczenia, a na koniec po raz kolejny odkryliśmy aplikację desktopową. Podobnych kół w historii inżynierii oprogramowania jest bez liku, ale to już zupełnie inna historia…

Inne pytania, na które warto odpowiedzieć

Co się może dziać z warstwą danych?

W naszym opowiadaniu warstwa danych była raczej niezmienna ograniczała się do dokumentu MS Word umieszczonego na komputerze użytkownika albo na serwerze. Ten sposób przechowywania danych jest dobry, ale ma jeden mankament – wszystkie operacje na danych trzeba samemu stworzyć (np. zapisywanie pliku albo jego wyszukiwanie). Kolejnym mankamentem jest wydajność: przy milionie plików do przeszukania programista musiałby wymyślać skomplikowane algorytmy. Programiści zazwyczaj wolą korzystać z gotowców, które już mają wbudowane tego typu funkcjonalności. Nazywają je bazami danych.



Rys. 15 Idea bazy danych

[5] http://www.openlaszlo.org.

[6] Popatrz na aplikację *Balsamiq Mockups* *http://www.balsamiq.com/products/mockups*. Pozwala ona na szybkie szkicowanie ekranów użytkownika. Aplikację tę napisano w *Adobe Air*. Można uruchomić zarówno jako aplikację webową z grubym klientem (link *Try it now*) jak i aplikację desktopową (link *Desktop App).*

Na rynku istnieje wiele różnego rodzaju baz danych. Do najpopularniejszych należą darmowe: *MySQL, PostgreSQL* oraz komercyjne: *Oracle Database, Microsoft SQL Server*.

Co to jest technologia?

Każdy system informatyczny oprócz specyficznych funkcjonalności oczekiwanych przez klienta, zawiera również szereg powtarzalnych elementów takich jak:

- przechowywanie danych,
- bezpieczeństwo,
- dbanie o to by dane były nieuszkodzone,
- weryfikacja danych wprowadzanych przez użytkownika.

Plus kilka bardziej szczegółowych. Technologia jest czymś, co w koncepcyjnie spójny i jednolity sposób dostarcza gotowe klocki (komponenty), które dbają o poprawne działanie systemu w wymienionych obszarach. W przeciwnym wypadku programista musiałby tworzyć wszystko od zera i z pewnością nie zakończyłby tworzenia systemu ani w zakładanym czasie, ani w budżecie.

Klejem, który umożliwia łączenie i wspomnianych kloców w większe całości jest język programowania. Zazwyczaj technologia jest „firmowana" przez jeden konkretny język programowania.

Technologie o najszerszym zastosowaniu, to:

- *Java SE/EE/ME* z językiem *Java*,
- *.NET* z językiem *C#*

Technologie o specjalizowanych zastosowaniach:

- *PHP* językiem *PHP5*
- *Adobe AIR* z językiem *AS3*

Technologie mają swoich zwolenników i przeciwników,….

Ile jest języków programowania?

Po pobieżnym przejrzeniu Internetu naliczyliśmy 582 różne języki programowania, lecz można założyć, że jest ich więcej. Języki programowania powstawały wraz z postępem inżynierii oprogramowania. Bardzo wiele z nich ma ścisłą specjalizację, w której sprawdzają się najlepiej np.: obliczenia matematyczne, przetwarzanie tekstów, języki dedykowane określonym urządzeniom (telefony, sprzęt AGD).

Według rankingu na portalu http://tiobe.com[7] w roku 2009, najbardziej popularnymi językami programowania były :

1. *Java*

[7] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

2. *C*
3. *C++*
4. *PHP*
5. *Visual Basic*
6. *C#*
7. *Python*
8. *Perl*
9. *Objective-C*

Najczęstsze nieporozumienia

Programista czy informatyk?

Zdecydowanie programista! Programowanie jest oczywiście częścią szerokiej dziedziny nauki, jaką jest informatyka. Jednakże potocznie przyjęło się nazywać informatykiem, każdą osobę która wie o komputerach cokolwiek więcej niż przeciętny użytkownik albo ma wykształcenie w jakiś sposób związane z informatyką. W szczególności informatykiem nazywa się osoby w firmach, które rozwiązują różnego rodzaju problemy ze sprzętem komputerowym od zainstalowania oprogramowania do usunięcia zaciętego papieru z drukarki włącznie (zatem bliżej takiej osobie do funkcji *helpdesk*).

Programiści nie lubią, gdy nazywa się ich informatykami, gdyż nie oznacza to nic konkretnego. Natomiast programowanie wymaga ściślej określonej wiedzy i umiejętności, a biegłość w programowaniu jest dla wielu programistów powodem do dumy.

Programiści mówią dziwnym językiem

Jest w tym stwierdzeniu sporo prawdy. Większość terminów programistycznych wywodzi się języka angielskiego. Narzędzia dla programistów również mają menu w języku angielskim. Dynamiczny rozwój tej dziedziny wiedzy sprawia, że publikacje i instrukcje trzeba czytać po angielsku, gdyż po przetłumaczeniu na język polski mogą stać się już nieaktualne. Sytuacja ta prowadzi do powstawania charakterystycznej gwary środowiskowej, na którą składają się spolszczone terminy anglojęzyczne. Za przykład może posłużyć zdanie: *Skiluj tomkata, bo war się nie chce zdeplojować…*

Na końcu niniejszego opracowania znajdziesz krótki słownik nowomowy programistycznej. Z pewnością pomoże Ci on lepiej zrozumieć, o czym mówi programista.

Programiści są mało komunikatywni

W tym stereotypie jest ziarno prawdy, lecz zdecydowanie mniejsze niż uważa większość nieprogramistów. Programiści są przede wszystkim oszczędni w słowach. Po pierwsze wynika to, ze specyfiki ich pracy, gdzie przez większość czasu są skupieni na rozwiązywaniu złożonych problemów. W tym przypadku częste odpowiadanie na nagle pojawiające się pytania, skutecznie rozprasza programistów. Po drugie programiści bardzo intensywnie komunikują się za pośrednictwem Internetu: wysyłają maile, udzielają się na forach, itp. Szukanie przydatnych informacji na forum

internetowym, na którym użytkownicy piszą bardzo długie wiadomości albo zamiast rozmawiać o danym problemie, ubliżają sobie nawzajem, jest dla programistów stratą czasu. Dlatego na wielu forach branżowych przyjęto określone etykiety prowadzenia rozmów, np.:

- Jeśli się z czymś nie zgadzasz, to zaproponuj alternatywę albo milcz,
- Jeśli się z czymś zgadzasz, to napisz „+1" (czyli coś w stylu „jestem za"), zamiast wyrażać aprobatę w pięciu zdaniach,
- Przestrzegaj tematu rozmowy, a jeśli chcesz napisać „o niczym", zrób to na forum ogólnym.

Programiści komunikują się więc bardzo skutecznie tyle, że w charakterystyczny sposób.

Obecnie bardzo rozwija się nurt pracy nad projektami programistycznymi nazywany *Agile*. Jednym z jego postulatów jest **ciągła interakcja z klientem**. W firmach wdrażających metodyki prowadzenia projektów w nurcie *Agile* widoczny jest postępujący wzrost umiejętności interpersonalny wśród programistów. Można przypuszczać, że tendencja ta będzie się utrzymywać.

Programiści nie dbają o wygląd fizyczny

Jest to zdecydowanie nieprawda. To przekonanie prawdopodobnie ma swoje źródło w latach osiemdziesiątych. W tamtych czasach wiele firm zatrudniało administratorów systemów operacyjnych UNIX, którzy o komputerach wiedzieli wszystko, a zwykli użytkownicy nie wiedzieli prawie nic. Z jakiś powodów, być może dla podkreślenia swojej odrębności, wielu z nich nosiło długie włosy i brody oraz bardzo nieformalne stroje. Ludzie Ci zdecydowanie wyróżniali się swoim wyglądem zwłaszcza w dużych firmach (bo przede wszystkim takie stać było na wdrożenie drogich systemów i zatrudnienie administratorów) od ubranych w garnitury i elegancko przystrzyżonych menadżerów.

W rzeczywistości odsetek programistów niedbających o wygląd, nie jest wcale większy niż w całej populacji. Jedynie duże zagęszczenie programistów na metr kwadratowy może sprawiać, że bardziej zwraca się na nich uwagę i wyciąga ogólne wnioski.

Mężczyźni są lepszymi programistami niż kobiety

Nie jest to prawdą. Odsetek kobiet studiujący na kierunkach technicznych wynosi od kilku do kilkunastu procent[8]. Zatem jeśli ktoś raz w życiu ma okazję pracować z kobietą-programistką i akurat zdarzy się, że nie będzie ona kompetentna, to łatwo wyciągnąć ogólny wniosek na temat wszystkich kobiet. W rzeczywistości stosunek ilość dobrych programistek wśród płci pięknej jest podobna do ilości dobrych programistów wśród mężczyzn (z zachowaniem wspomnianej na początku proporcji).

Programiści lubią pomagać, ale…

Cytujemy fragment rozmowy pomiędzy Specjalistą ds. Sprzedaży a Programistą:

- [Specjalista ds. Sprzedaży] *Cześć, jak wstawić w kolorową tabelkę w łordzie?*
- [Programista] *Nie wiem…*
- [Specjalista ds. Sprzedaży] Nie *wiesz? Co z Ciebie za informatyk?*

[8] http://www.studia.org/wydarzenia/372-ile-jest-kobiet-na-politechnikach

Tego typu rozmowy zdarzają się dość często. Większość osób, dla których oprogramowanie komputerowe jest narzędziem pracy, uważa, że programista zna się na wszystkim, co tylko jest związane z IT. Od pisania programowania począwszy, poprzez konfigurację sieci, sprzętu (a w tym również znajomość cen i najnowszych trendów), a skończywszy na wszelkim możliwym oprogramowaniu biurowym, pocztowym, do odtwarzania muzyki i filmów. Ten dowód bezgranicznego zaufania jest bardzo miły, a jednocześnie dużo przesadzony i czasem stresujący dla programistów. Owszem, programiści czasem interesują się którymś z wymienionych obszarów, ale nie wiedzą wszystkiego. Mają bardzo przydatną umiejętność szybkiego wyszukiwania informacji i błyskawicznie orientują się w nowym oprogramowaniu, ale zazwyczaj nie mają tej wiedzy pod ręką.

Programiści przede wszystkim znają się na programowaniu i na narzędziach programistycznych. Rzadko korzystają z oprogramowania biurowego, a jeśli już to raczej z podstawowych funkcji. Jeśli programista nie wie jak wstawić kolorową tabelkę w „łordzie", to szuka w Internecie, dowiaduje się i wstawia. Jeśli poprosisz programistę o pomoc, to z pewnością Ci pomoże (i to dość szybko) albo wskaże kogoś, kto mógłby pomóc. Nie oczekuj jednak, że będzie wiedział wszystko od razu.

Jawa to nie tylko motocykl

Każdy chciałby, aby doceniać jego pracę. Programiści również. Aby zrozumieć radość programisty, wynikającą z odkrycia lub zaprogramowania czegoś nowego, trzeba choć minimalnie wiedzieć na czym polega jego praca i choć trochę rozumieć problemy, z którymi styka. Jeśli *jawa*, oznacza dla Ciebie wyłącznie markę motocykla lub kawę, to osobista relacja nawiązana programistą będzie bardzo uboga.

To co widać, to czubek góry lodowej

Użytkownicy oprogramowania zazwyczaj są świadomi tylko wyglądu jego zewnętrznego oraz funkcjonalności. Dość często nie zdają sobie sprawy, jakie konsekwencje mają ich wymagania. Poniżej zamieściliśmy cytat z autentycznej rozmowy z klientem.

*Nowy program do budżetowania ma działać tak jak nasze arkusze Excel **z tym wyjątkiem**, że przy kwocie znajdzie plusik, a po jego wciśnięciu będzie można obejrzeć części składowe kwoty aż do pojedynczej pozycji na fakturze. Czyli wiadomo jak działa Excel, więc to powinno zająć chwilę, prawda?*

Wymaganie jest dość zrozumiałe. Kłopot pojawia się w momencie, gdy klient zaczyna oczekiwać, że „to zajmie chwilę", bo przecież „jest takie podobne do Excela". Nie przypadkowo wyróżniliśmy fragment „z tym wyjątkiem". To właśnie ten wyjątek sprawia, że nowa funkcjonalność, w dziewięćdziesięciu procentach nie będzie podobna do Excela. Jest tak dla tego, że to co widzi użytkownik to tylko czubek góry lodowej. Ogrom pracy nad oprogramowaniem wykonuje się „pod spodem" (w warstwach logiki i danych), a interfejs użytkownika (choć bardzo ważny), stanowi jedynie część prac.

Jeśli zatem programista mówi, że widzi problem w stworzeniu lub zmianie danej funkcjonalności, to ten problem z pewnością i istnieje. Może nie jest tak poważny, jak się to na początku wydaje, ale jest rzeczywisty.

Program programowi nie równy

Przytaczamy autentyczne sytuacje z tworzenia oprogramowania na zamówienie pewnej firmy.

1. [Sprzedawca, pierwsze spotkanie z klientem]: *(…) zatem każdy z pracowników będzie miał dostęp do systemu. Po zalogowaniu się będzie wykonywał zadania z poziomu panelu.*

2. Po przeanalizowaniu wymagań, zespół zaproponował architekturę aplikacji webowej, a po otrzymaniu akceptacji rozpoczął pracę.

3. [Klient, spotkanie na dwa tygodnie przed wdrożeniem systemu]: *A jak to oprogramowanie będzie konkretnie instalowane, kto się tym zajmie?*

4. [Sprzedawca]: *Dokładnie tak samo jak każdy program, który Pan instalował. Włoży Pan płytkę do napędu i zainstaluje.*

W wyniku nie zrozumienia podstawowej charakterystyki tworzonego systemu, Sprzedawca zbudował u klienta oczekiwanie, że otrzyma aplikację desktopową. Tym czasem zespół stworzył aplikację webową. W ciągu dwóch tygodni należało znaleźć rozwiązanie problemu tak, aby firma nie straciła twarzy przed klientem.

Aplikacje działające na laptopie, w Internecie (w architekturze webowej), w telefonie, w systemie operacyjnym Windows, Macintosh albo Linux, nawet jeśli wyglądają dokładnie tak samo i wykonują dokładnie to samo, **są zupełnie różnymi programami**. Czubek góry lodowej jest ten sam, ale podstawa inna. W większości wypadków programy te są nieprzenośne pomiędzy wymienionymi środowiskami i dla każdego środowiska trzeba je pisać na nowo. Czasami bywają przenośne po pewnych kosmetycznych poprawkach, które jednak wymagają czasu i pieniędzy

Słownik nowomowy programistycznej

- **patern** - od *pattern,* pełna nazwa: *desing pattern;* zazwyczaj oznacza konstrukcję w języku programowania rozwiązującą określony problem; *paterny* stanowią dziedzictwo społeczności programistycznej, są wielokrotnie używane przez pokolenia programistów

- **debagować** – od *debug*; poszukiwanie błędów w programie za pomocą specjalistycznego narzędzia (*debugera*)

- **logować** – w kontekście programowania oznacza to zapisywanie informacji diagnostycznych przez program do specjalnego pliku (*loga*); dopiero w drugiej kolejności oznacza to podanie nazwy użytkownika, hasła i wciśnięcie przycisku *zaloguj*

- **źródło** – pełna nazwa: *kod źródłowy;* opis w język programowania tego, co ma robić program

- **repozytorium** – lub krócej: *repo*; centralne miejsce, w którym trzymany jest kod źródłowy; pozwala ono na jednoczesną współpracę wielu programistów nad tym samym kodem

- **komitować** – od commit; przesłanie pliku z kodem źródłowym do repozytorium

- **krud** – od *Create Retrieve Update Delete*; zestaw elementarnych operacji wykonywanych w każdnym systemie informatycznych

- **deplojować** – od *deploy*; zazwyczaj odnosi się do aplikacji webowych i oznacza skopiowanie pliku z aplikacją na serwer i uruchomienie jej

- **kilować** – od *kill*; siłowa zakańczanie działania programu, który z jakiś powodów przestał działać prawidłowo i nie chce się zamknąć z własnej woli

- **egzek** – od *executable*; plik z programem, który można uruchomić poprzez podwójne kliknęcie; zazwyczaj dotyczy programów dla systemu operacyjnego Windows

- **binarka** – od binary; jakikolwiek plik, który nie jest plikiem czystym tekstowym *txt*; w kontekście systemów operacyjnych innych niż *Windows* (*Macintosh*, *Linux*) oznacza również plik z programem, który można uruchomić

- **grinbar** – od *green bar;* informacja, że test framgentu programu zakończł się pomyślnie; słowo nawiązuje do paska postępu, który wskazuje ilość uruchomionych testów; pasek zielony oznacza, że wszystkie testy zakończyły się poprawnie; pasek czerwony oznacza, że przynajmniej jeden test zakończył się błędem

- ***dżar*** – plik uruchomieniowy programu (najczęściej aplikacji desktopowej okienkowej lub działającej w trybie konsolowym) napisanego w języku Java

- ***war*** – plik uruchomieniowy aplikacji webowej napisanej w języku Java

- ***ear*** – plik uruchomieniowy złożonego systemu informatycznego działającego jako aplikacja webowa i napisanego w języku Java

- ***damp*** – od *dump*; „zrzut" do pliku; najczęściej używany w dwóch znaczeniach: (1) *damp pamięci* – dane znajdujące się w pamięci operacyjnej komputera zapisane w pliku; służy do poszukiwanie przyczyn błędów w programie; (2) *damp bazy danych* – dane zawarte w bazie danych „zrzucone" do jednego pliku; używany przez programistów do analizy zawartości bazy danych jeśli nie mogą otrzymać do niej stałego dostępu (w postaci nazwy użytkownika i hasła)

- ***napisać fora, ifa, kejsa, foricza, łajla, dułajla*** – od *for, if, case, foreach, while, do while;* napisać pojedynczą instrukcję w języku programowania (każda z nich ma swoje specyficzne przeznaczenie)

- **program** – końcowy efekt programowania; w zależności od skali, kosztów i fantazji bywa również nazywany *systemem* albo *aplikacją*

- **gik** – od *geek*; wśród programistów słowo to oznacza absolutnego niekwestionowanego eksperta w danym temacie

- **cul** – od *cool; świetnie! super!*

- **rulez** – wyraz uznania np.: *PHP rulez*! – *PHP rządzi!*

- **lol** – od *laughing on the flor;* używane w mailach i komunikatorach

- **rotfl** – od *rolling on the floor laughing;* używane w mailach i komunikatorach

- **imho** – od *In my humble opinion;* używane w mailach i komunikatorach

- **btw** – od *by the way;* używane w mailach i komunikatorach
- **thx** – od *thanks, thank you;* używane w mailach i komunikatorach
- **+1** – *zgadzam się;* używane w mailach w odpowiedzi na prośbę o ustosunkowanie się do konkretnej kwestii

Słownik technologii programistycznych

- **Java** – język programowania opracowany przez firmę Sun Microsystems, który jest podstawą technologii umożliwiającej tworzenie złożonych systemów informatycznych; główne składowe tej technologii to *Java SE* – do tworzenia aplikacji desktopowych, *Java EE* – do tworzenia złożonych systemów korporacyjnych w tym aplikacji webowych; *Java ME* – do tworzenia oprogramowania na telefony komórkowe
- **C#** - język programowania opracowany przez firmę Microsoft, który jest podstawą technologii umożliwiającej tworzenie złożonych systemów informatycznych; jest podstawą technologii *.NET* uważanej za rozwiązanie konkurencyjne do technologii *Java*
- **PHP** – język programowania wykorzystywany przede wszystkim do tworzenia aplikacji webowych
- **Flash** – technologia pozwalająca na tworzenie niewielkich (rozmiarowo) programów; programy *flash* ze względu na ich rozmiar i duże możliwości są chętnie osadzane na stronach intrenetowych
- **Flex** – technologia umożliwiająca tworzenia aplikacji webowych typu *RIA*(patrz rozdział: *Architektura systemów informatycznych*), w których interfejs użytkownika skonstruowany jest we *flash*
- **AIR** – technologia umożliwiająca tworzenia webowych aplikacji typu *RIA* oraz aplikacji desktopowych opartych o *flash*
- **HTML** – język do opisu wyglądu stron internetowych
- **XML** – bardzo ogólny język do opisu różnego rodzaju danych; coraz więcej programów tworzy dokumenty opisane za pomocą *XML*
- **JavaScript** – język programowania umożliwiający dodawanie zachowania do statycznych stron HTML ; nie ma nic wspólnego z językiem Java
- **AJAX** – idea umożliwiająca budowanie złożonych interfejsów użytkownika opartych o strony *HTML* (nawet typu *RIA*) w aplikacji webowej;