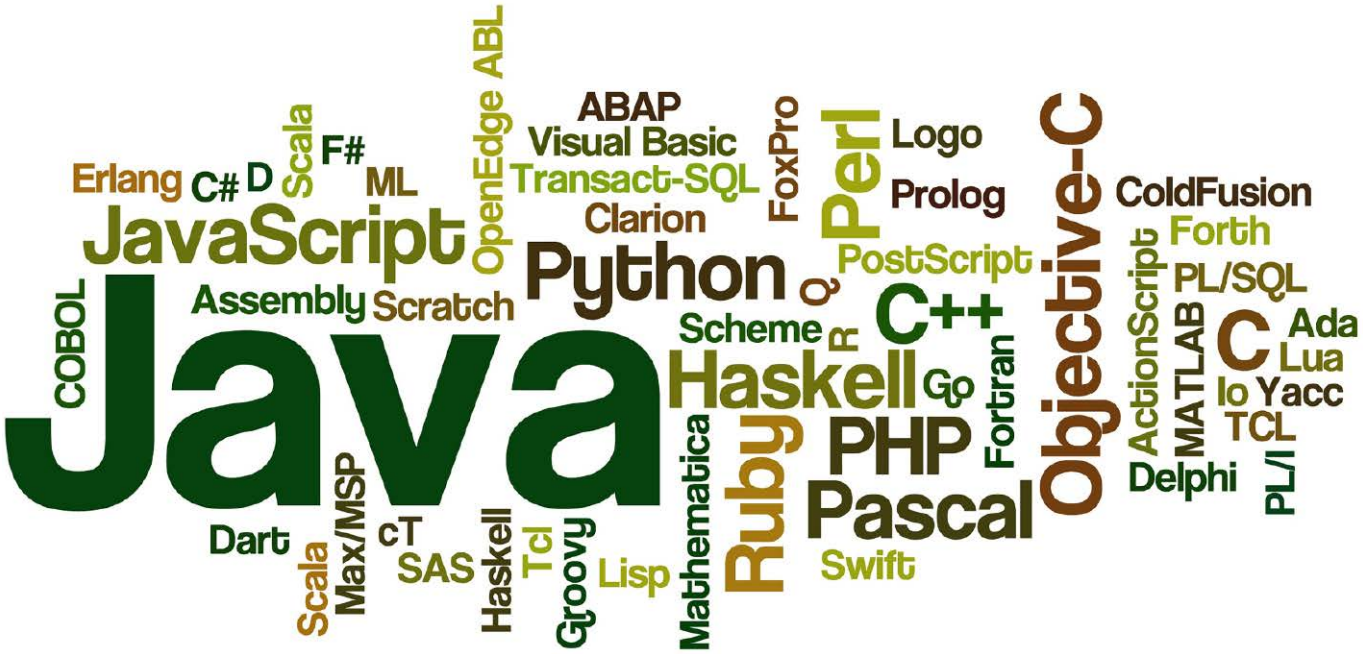


Języki programowania dla technologii sieciowych

Zawód elektronika konstruktora niewątpliwie jest bezpośrednio związany z nowoczesnymi technologiami i tak, jak każdy zawód tego typu wymaga ciągłego uczenia się oraz dostosowywania do trendów rynkowych. Tempo, w jakim zachodzą zmiany na rynku elektroniki, jest niesamowite, a wielkość i znaczenie tych zmian sprawiają, że praca elektronika podlega totalnym przekształceniom. Aktualnie wygląda na to, że większość elektroników musi znać się na programowaniu, a trendy rynkowe pozwalają przewidywać, że z czasem znaczenie programowania tym bardziej wzrośnie. Jakich więc języków programowania powinien nauczyć się inżynier elektronik?

Droga do zawodu elektronika biegnie zazwyczaj na dwa różne sposoby. Młody człowiek albo samodzielnie wykazuje zainteresowanie elektroniką i próbuje, czasem pod okiem starszych, zbudować swoje własne układy lub naprawić wybrane urządzenia, albo idzie do technikum i na studia, przechodząc przez wiele obowiązkowych zajęć i uzyskując na koniec dyplom ukończenia uczelni. W praktyce drogi te są często ze sobą powiązane i o ile na początkowym etapie da się uniknąć programowania, w trakcie studiów jest ono nauczane obowiązkowo, choć nie zawsze skutecznie. Problematyczny jest też wybór języków nauczanych na studiach, który koncentruje się raczej na podstawach programowania niż na specyfice aktualnie dostępnych i stosowanych wersji języków i platform programistycznych. A szkoda, bo praktyka pokazuje, że bardzo duża część absolwentów elektroniki zostaje koniec końców programistami, z racji szerokiej dostępności tego zawodu, dobrego wynagrodzenia i pokrewieństwa tematycznego z wiedzą z elektroniki.

Obecnie programowanie stało się też jednym z podstawowych czynności wykonywanych przez elektroników pracujących w swoim zawodzie. Wynika to z oczekiwań rynku. Projektowane urządzenia są na tyle skomplikowane, a jednocześnie multimedialne i w końcu ostatnio – łączą się z Internetem, że realizacja wszystkich koniecznych funkcji czysto za pomocą komponentów elektronicznych, bez programowania, nie byłaby możliwa do wykonania. Co więcej, oferowanie uniwersalnych, programowalnych podzespołów pozwala producentom układów scalonych wytwarzać te same komponenty masowo, bez potrzeby ich specjalizowania na etapie produkcji. Inżynier elektronik sam sobie je oprogramuje, w zależności od potrzeb. I może to zrobić na wiele sposobów.

Wzrost poziomu abstrakcji

Dawniej, gdy układy cyfrowe zaczęły pojawiać się w elektronice masowo, w postaci prostych 8-bitowych mikrokontrolerów, miały na tyle ograniczone parametry, że do ich programowania wykorzystywano asemblery. Te pozwalały stworzyć oszczędny kod, w pełni wykorzystując potencjał podzespołów. Co więcej, reprogramowanie też nie było wygodne, wiec aktualizacja raz wgranego w scalak oprogramowania raczej nie była praktykowana. Z czasem popularność zyskał znacznie upraszczający programowanie Bascom oraz język C, który przenosił profesjonalne tworzenie kodu na wyższy poziom abstrakcji. Wykorzystanie C++ wprowadziło programowanie obiektowe do świata układów scalonych oraz ułatwiło dostawcom wszelkiego rodzaju bloków IP, takich jak np. biblioteki i moduły programowe, udostępniać łatwe w integracji, zamknięte fragmenty kodu. W ten sposób producenci mikrokontrolerów zaczęli oferować – coraz częściej bezpłatnie

– np. stosy komunikacyjne, które za pomocą kilku linijek kodu można zaimportować do własnych projektów i używać w wygodny sposób.

W końcu, w nowoczesnych, zintegrowanych środowiskach deweloperskich dostępnych jest wiele kreatorów, które umożliwiają szybkie wygenerowanie fragmentów kodu w oparciu na uniwersalnych szablonach. Bywa też, że zamiast kreatora dostępny jest meta język, np. w postaci graficznej, który pozwala na łatwe wykorzystanie predefiniowanych bloków programowych i przypisanie wyjść i wejść mikrokontrolera do poszczególnych fragmentów kodu. Stworzony w ten sposób opis projektu służy następnie do wygenerowania wynikowego kodu w języku C, który można samodzielnie zmodyfikować lub od razu skompilować i uruchomić. To przykład programowania na bardzo wysokim poziomie abstrakcji.

Technologie sieciowe

Korzystanie z języków programowania o wysokim poziomie abstrakcji to niejedyny aspekt programowania nowoczesnych urządzeń elektronicznych, na jaki warto zwrócić uwagę. Tak jak kiedyś wszystkie urządzenia elektroniczne zaczęły być wytwarzane jako cyfrowe, a później także i multimedialne, tak teraz cała elektronika, by była nowoczesna, musi być podłączona do Internetu, czyli w specyficzny sposób sieciowa. Trend ten nazywamy powstawaniem Internetu Rzeczy (IoT – *Internet of Things*), przy czym nie mówi się po prostu o nowym sprzęcie jako o „urządzeniach sieciowych" z jednego, ważnego powodu. Istotne jest bowiem, by poszczególne urządzenia elektroniczne mogły się ze sobą komunikować za pomocą uniwersalnych protokołów. Nie oznacza to oczywiście, że cała elektronika będzie się świetnie rozumieć i ze sobą automatycznie współpracować, tak samo jak użytkownicy Internetu mówiący w różnych językach nie będą w stanie się ze sobą porozumieć, nawet pomimo wykorzystywania takich samych przeglądarek internetowych. Ważne jest jednak, by nowoczesne urządzenia elektroniczne, aby można je było zaliczyć do urządzeń IoT, korzystały z uniwersalnych protokołów i powszechnie stosowanych mechanizmów wymiany danych. Warto się ich nauczyć.

Podstawowa kwestia dotyczy protokołu IP oraz powiązanych z nimi protokołów TCP/IP i UDP. Zrozumienie ich działania jest bardzo ważne, jeśli buduje się wszelkiego rodzaju urządzenia sieciowe. Szczęśliwie obecnie bardzo łatwo jest zdobyć gotowe biblioteki do prowadzenia komunikacji za pomocą tych protokołów, ale aby ta miała sens, trzeba rozumieć istotę połączeń sieciowych oraz koncepcję połączeń klient-serwer. W oparciu na nich działają wszelkiego rodzaju serwisy internetowe.

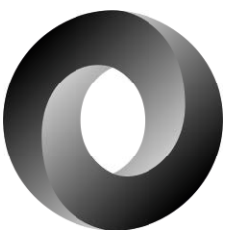
Po drugie, nowoczesne aplikacje Internetu Rzeczy opierają się przede wszystkim na mechanizmach zbierania, przesyłania i gromadzenia bardzo dużych ilości danych. Małe, „inteligentne" urządzenia są w stanie dosyć szybko wygenerować taką liczbę danych, jakiej same nie mogłyby pomieścić ani w sensowny sposób przenalizować. Dlatego muszą je przesyłać do serwerów, nierzadko umieszczonych w tzw. chmurze, czyli gdzieś w Internecie. Transmisja tych danych odbywa się typowo na dwa odmienne sposoby. Pierwszy i zarazem starszy to wykorzystanie bezpośrednich zapytań do bazy danych. W tym celu używa się języków SQL, których wersje zależą od używanego serwera bazodanowego (MySQL, PGSQL, MS SQL itd.). Poprzez odpowiednio zabezpieczone połączenie sieciowe przesyła się długie ciągi komend w postaci tekstu, po czym czeka na odpowiedź serwera z bazą danych, by poznać wynik zleconego zapytania. Wykorzystanie języka SQL pozwala dowolnie operować zasobami bazy danych (z ograniczeniem do posiadanych uprawnień), ale może być skomplikowane. Ponadto udostępnienie SQL-owego interfejsu serwera użytkownikom jednej bazy rodzi problemy związane z bezpieczeństwem – trudno jest ograniczyć działania poszczególnych użytkowników w taki sposób, by nie zniszczyli wzajemnie swojej pracy, chyba że każdemu przypisze się oddzielne bazy – a to bardzo niewygodne z punktu widzenia zbiorczego analizowania zebranych informacji.

Dlatego w aplikacjach IoT bardzo często korzysta się z serwerów chmurowych, w których dostęp do danych (do zapisu) opiera się na przygotowanym interfejsie programistycznym – tzw. API. Każdy użytkownik (czyli np. każdy sensor) otrzymuje własny klucz, który umożliwia określony dostęp do bazy i który pozwala w łatwy sposób zidentyfikować dane umieszczane przez ten właśnie węzeł sieci. Polecenia API wywoływane są poprzez przesyłanie do serwerów WWW odpowiednich, krótkich zapytań, zawierających najczęściej komendę i jej parametry wywołania oraz identyfikator i hasło (lub skrót hasła) użytkownika. Korzystanie z takiego API często realizowane jest w oparciu na prostych bibliotekach, które zawierają zestawy typowych funkcji. API każdego z serwerów tego typu trzeba poznać oddzielnie, choć w praktyce są one do siebie w ogólności dosyć podobne, zwłaszcza jeśli chodzi o ideę działania.

Zupełnie inaczej wygląda natomiast kwestia wykorzystania danych zebranych w serwerach w chmurze, w aplikacjach IoT. Tu już bardzo wiele zależy od usługi, z którą zostały połączone urządzenia, ale powszechną praktyką jest, że operator serwerów przygotowuje zestaw narzędzi do przeglądania i analizy danych, a nawet do tworzenia raportów na ich podstawie. Najbardziej zaawansowane usługi pozwalają na przetwarzanie danych za pomocą sieci neuronowych i gotowych algorytmów, działających w chmurze i ułatwiających wykorzystanie zebranych informacji.

Uniwersalne formaty danych

Ze zbiorów danych można też korzystać samodzielnie, na własnym komputerze lub urządzeniu elektronicznym, po ich pobraniu w którymś ze dostępnych formatów. W praktyce do wymiany bloków danych pomiędzy maszynami wykorzystuje się kilka typowych formatów danych. Podstawowy to plik tekstowy, gdzie kolejne dane są zapisywane w kolejnych linijkach. Nie jest to jednak wygodny mechanizm, gdyż podział na rodzaje danych i nadawanie im etykiet są w tym przypadku bardzo utrudnione. Dlatego lepszym wyjściem jest choćby stosowanie formatu, w którym poszczególne rekordy są od siebie oddzielane znakami końca linii, a pola rekordów innymi separatorami – np. tabulatorami, przecinkami lub średnikami. Tego typu bloki danych łatwo jest przygotować i zaimportować do narzędzi używanych przez człowieka – takich jak np. arkusz kalkulacyjny. Jednakże w przypadku czystej komunikacji M2M zaleta ta nie ma dużego znaczenia – ważniejsza jest możliwość bezbłędnego przesłania dowolnych danych w sposób najbardziej oszczędny, a zarazem pozwalający na najlepsze opisanie informacji. Z tego względu do komunikacji M2M w Internecie często wykorzystywany jest format JSON – JavaScript Object Notation. JSON pozwala układać dane w sposób hierarchiczny oraz przypisywać im etykiety. Jest o tyle łatwy w użyciu, że w wielu językach programowania istnieją gotowe polecenia, konwertujące dowolne obiekty do formatu JSON – czyli do ciągu znaków w formacie UTF-8, w którym hierarchia uzyskiwana jest za pomocą nawiasów, etykiety podawane są przed znakami dwukropka, a tekst umieszczany jest w cudzysłowie. Oczywiście dostępne są też funkcje pozwalające na przetworzenie ciągu w formacie JSON na zmienne obiektowe. Format JSON jest powszechnie wykorzystywany w aplikacjach opartych na modelu komunikacji REST (Representational State Transfer), w których wymiana danych polega na przekazywaniu prostych komunikatów. REST świetnie nadaje się właśnie do aplikacji IoT, w których np. sensory przekazują wybrane ze zbieranych danych do serwera w chmurze.



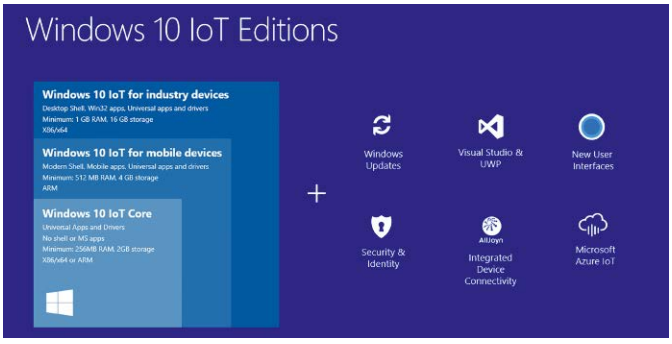
Drugim z często stosowanych formatów, którego zaletą są powszechność oraz większa czytelność niż w przypadku JSON, jest XML (Extensible Markup Language). XML jest niejako językiem, wykorzystywanym przez bardzo wiele systemów komputerowych – w tym często jako format pośredni dla metajęzyków używanych do tworzenia kodu w językach podstawowych, tak jak to zostało opisane wcześniej. XML pozwala na tworzenie czytelnej, hierarchicznej struktury danych, w której etykiety i struktura są określana za pomocą nawiasów i znaków slash (ukośników). Format XML jest często dostępny jako format eksportu danych we wszelkiego rodzaju programach komputerowych oraz, dzięki swojej powszechności i prostocie – łatwo znaleźć parsery przetwarzające go na zmienne obiektowe lub konwertujące zmienne do tekstu w formacie XML. Jest także bardzo prosty do nauki i warto go poznać, szczególnie że bywa często wykorzystywany do tworzenia plików konfiguracyjnych. Natomiast w przypadku komunikacji M2M ma tę wadę względem formatu JSON, że wprowadza większy narzut dodatkowych danych, wydłużając przesyłane bloki danych.

Programowanie w systemach operacyjnych

Fakt, że w typowych aplikacjach Internetu Rzeczy dane przesyłane są przez niewielkie, „inteligentne" sensory, pełniące funkcję klientów większych serwerów, sprawia, że zazwyczaj w komunikacji biorą udział dwa rodzaje urządzeń elektronicznych. Sensory najczęściej, choćby ze względu na oszczędność kosztów i energii elektrycznej, to urządzenia proste, oparte na mikrokontrolerach, przebywające dużą część czasu w trybie uśpienia. Natomiast serwery to prawie zawsze urządzenia z systemem operacyjnym, który zdecydowanie ułatwia gromadzenie dużych ilości danych, pochodzących od wielu węzłów sieciowych. Wielozadaniowość systemu operacyjnego jest w tym przypadku bardzo ważna, gdyż pozwala bardzo łatwo na względnie jednoczesną komunikację z wieloma urządzeniami klienckimi. Problem w tym, że programowanie urządzenia z systemem operacyjnym to zupełnie inne zadanie niż programowanie prostego mikrokontrolera.

W praktyce programowanie systemu operacyjnego jest najczęściej tak naprawdę prostsze niż tworzenie programu na mikrokontroler. Dostępność zasobów jest większa i ma się większą swobodę w zakresie wyboru języków programowania. W efekcie, jeśli koszt użytkowania gotowej aplikacji nie jest istotny, a kluczowe jest szybkie wprowadzenie produktu na rynek, może się okazać, że stworzenie zestawu urządzeń do Internetu Rzeczy, w którym końcówki sieciowe (np. sensory) będą pracować na komputerach z systemem operacyjnym. będzie miało uzasadnienie. Z myślą o takich rozwiązaniach powstało już trochę systemów operacyjnych, a wśród nich także Microsoft Windows 10 IoT.

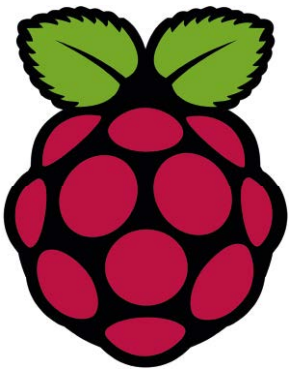
Wybór języka programowania, stosownego do danej aplikacji sieciowej, działającej na komputerze z systemem operacyjnym, będzie zależał przede wszystkim od zainstalowanego systemu oraz od interfejsu komunikacyjnego (Ethernet, Bluetooth, inny). W drugiej kolejności należy zwrócić uwagę na pożądaną wydajność, dostępność gotowych bibliotek, przydatnych w danym projekcie, a także na prostotę programowania.



W ogólności warto wyróżnić cztery rodzaje systemów operacyjnych, z jakimi można się spotkać w aplikacjach IoT: Linux, MS Windows, Android oraz Apple Mac OS X lub iOS. W przypadku wykorzystania urządzeń marki Apple, niemalże konieczne staje się pisanie kodu w języku Objective-C. Pozwoli on na optymalne wykorzystanie zasobów urządzenia i przygotowanie dowolnych programów dla systemów operacyjnych Mac OS X i iOS. Możliwe jest co prawda napisanie kodu w języku JavaScript, z użyciem takich platform jak Cordova lub node.js, ale skoro z jakiegoś powodu program ma działać właśnie na sprzęcie firmy Apple, to zalecane będzie nauczenie się Objective-C (mocno podobnego zresztą do C++).

W przypadku urządzeń mobilnych, opartych na Androidzie, korzystne będzie nauczenie się języka Java. Niestety, nie jest to zadanie proste – sprawne wykorzystanie Javy wymaga znajomości różnych bibliotek, co przychodzi dopiero z doświadczeniem.

W ostatnim czasie na rynek aplikacji IoT silnie wkracza firma Microsoft z nowym Windows 10 IoT Core. System ten został pomyślany tak, by przy minimalnym wykorzystaniu zasobów móc realizować zadania typowe dla aplikacji Internetu Rzeczy. Windows 10 IoT Core programuje się w zintegrowanym środowisku Visual Studio, tworząc uniwersalną aplikację, która będzie mogła działać także na innych wersjach Windows 10. Komputer z Windows 10 IoT Core może mieć w danej chwili uruchomioną i aktywną tylko jedną taką aplikację, co w zupełności wystarcza dla węzłów pełniących funkcję inteligentnych sensorów, a często także będzie wystarczające dla prostych, specjalizowanych serwerów sieciowych. Co ważne, Windows 10 IoT Core można uruchomić na komputerach Raspberry Pi 2 i 3 oraz np. MinnowBoard Max, a więc z rdzeniami ARM. Natomiast programy pod kątem Windows najkorzystniej jest pisać w języku C# lub np. z użyciem WinJS, czyli ograniczonego w uprawnieniach, ale prostego w użytku i rozbudowanego o dodatkowe biblioteki JavaScriptu.



W końcu dostępny jest także Linux, który daje największą swobodę wyboru języka programowania. W przypadku aplikacji wymagających dużej wydajności powszechnie wykorzystywany są języki C i C++. Jeśli natomiast ważniejsza jest łatwość wykonania projektu i dostępność uniwersalnych bibliotek, warto pomyśleć o nauce języka Python, który dominuje w przypadku aplikacji realizowanych na miniaturowych komputerach, takich jak Raspberry PI.

Serwisy WWW

To, że urządzenia IoT są – z zasady – podłączone do Internetu, jest wykorzystywane do tworzenia ich interfejsów użytkownika. Najłatwiejszym sposobem ich przygotowania jest użycie technologii stron internetowych, ładowanych przez przeglądarki internetowe. Przykładowo, interfejs konfiguracyjny dowolnego sprzętu można w łatwy sposób udostępnić przez uruchomienie prostego serwera WWW

(dostępnego w postaci gotowych bibliotek i programów) na urządzeniu oraz wskazanie mu plików źródłowych do przetwarzania i prezentacji. Aby świadomie tworzyć takie interfejsy, należy nauczyć się całkiem prostego języka HTML (HyperText Markup Language). Jeśli interfejs ma wyglądać atrakcyjnie, warto też nauczyć się sposobu tworzenia stylów w języku CSS (Cascade StyleSheets) w wersji 3 oraz poznać tajniki HTML w wersji 5. Dynamiczne reagowanie interfejsu na działania użytkownika zapewni odpowiednie wykorzystanie języka JavaScript. Język ten też może być przydatny do sprawienia, by prezentowane strony nie były statyczne, tylko by prezentować na nich wybrane i pobrane z zewnątrz dane. Alternatywnie, do pobierania danych i uzależniania ich od akcji użytkownika, można wykorzystać np. język PHP, który zazwyczaj służy do generowania wynikowego kodu HTML w serwisach internetowych. W efekcie, chcąc tworzyć prosty serwis WWW, w którym będą np. prezentowane historyczne dane z kilku czujników, warto zapoznać się z zestawem języków: HTML5+CSS3+JavaScript+PHP+SQL. Choć zbiór ten może wydawać się ogromny, już nawet podstawy każdego z nich wystarczą do zrealizowania prostego serwera z informacjami. W przypadku serwisów opartych na technologiach Microsoftu, PHP warto zastąpić językiem ASP.NET.

Warto wspomnieć, że w ostatnim czasie na popularności zyskuje język JavaScript. Jest on prosty, jeśli chodzi o składnię i kwestie związane z typami danych i zarządzaniem pamięcią. Choć pisanych w nim programów się nie kompiluje, a jedynie są one interpretowane, JavaScript zaczął być używany do realizacji wysocewydajnych, „wielowątkowych" serwerów. Wynika to z faktu, że jest z natury zdarzeniowy i pozwala reagować na przychodzące żądania, bez konieczności rezerwowania na ich realizację określonej liczby procesów czy wątków procesora. Bardzo łatwo w nim pisać kod, który będzie działał równolegle, w taki sposób, że obsługa jednego żądania nie blokuje możliwości przyjmowania i obsługiwania innych żądań.

Programowanie na zlecenie

Powyższe rozważania dotyczą sytuacji, w której inżynier elektronik chce zrealizować konkretny projekt i ma już wstępnie określone jego założenia. Pomagają dokonać wyboru optymalnych języków programowania, jakie przydatne będą do realizacji planowanego projektu. Co jednak, gdy celem jest jedynie zwiększenie swoich umiejętności, by polepszyć szanse na rynku pracy?

W takiej sytuacji warto rozważyć takie czynniki, jak zapotrzebowanie na programistów danego języka oraz przeciętne wynagrodzenie. Co ciekawe, nie ma istotnej korelacji pomiędzy tymi czynnikami. Statystyki pokazują, że od lat do najlepiej opłacanych i najbardziej pożądanych języków programowania należy Java, będąca zarazem bardzo popularnym językiem wśród programistów. Tymczasem język PHP, którego powszechność jeszcze niedawno była nawet większa, jest wyraźnie gorzej opłacany. Bardzo dobre wynagrodzenie można też otrzymać dzięki znajomości Ruby czy tak mało popularnych języków jak Clojure, Haskell, Groovy lub Scala, na które jednak zapotrzebowanie na rynku jest niewielkie. Dosyć dobrze płatny, a zarazem łatwy i względnie pożądany jest Python. Bardzo duże zapotrzebowanie jest także na znajomość języka JavaScript, choć tu wynagrodzenie wypada także poniżej średniej programistów wszystkich. W końcu niezmiernie przydatna okazuje się znajomość SQL-a, choć często nie jest on traktowany jako prawdziwy język programowania i bywa wymieniany w ogłoszeniach o pracę na stanowiska związane z finansami i analizą danych, mimo że, jak wskazaliśmy, ma też zastosowanie w aplikacjach sieciowych.

Podsumowanie

Wymienione w artykule języki programowania nie wyczerpują wszystkich technologii, użytecznych do realizacji aplikacji sieciowych. Co więcej, aktualne trendy w aplikacjach sieciowych prowadzą do wykorzystywania wielu różnych środowisk i stosowanych dla nich języków programowania lub konfiguracji w ramach pojedynczego programu. Zaawansowane systemy sieciowe są podzielone na bloki odpowiadające za model danych, sposób ich prezentacji oraz modyfikacji, a nawet na moduły realizujące przygotowanie danych, komunikację pomiędzy blokami czy obsługę wirtualnych platform w chmurze. Poszczególne z tych bloków operują innymi językami, niekiedy dosyć prostymi, których nauka sprowadza się do przeczytania kilkudziesięciostronicowej dokumentacji.

Pojawia się też coraz więcej meta języków, takich jak LESS CSS, który służy do generowania plików CSS w sposób ułatwiający ich szybkie modyfikowanie. Do tego, by wykorzystywać nowoczesne technologie, konieczne staje się podążanie za aktualnymi trendami i śledzenie popularności różnorodnych bibliotek, zdecydowanie usprawniających budowanie aplikacji. Bywa jednak, że wybrana biblioteka, nad której nauką (i użytkowaniem) spędziliśmy wiele miesięcy, przestaje być rozwijana i popada w niepamięć, w związku z czym konieczne staje się znalezienie innej, która ją zastąpi, tak by powstające aplikacje wciąż działały i wyglądały nowocześnie.

W praktyce oznacza to, że warto zapoznać się z podstawowymi językami programowania, takimi jak te nauczane na studiach. Dzięki ich dużej złożoności można zrozumieć szczegóły programowania, które później – często w uproszczeniu – pojawiają się w innych, nowszych językach. Następnie warto douczać się tych języków, które akurat potrzebne są do danego projektu i starać się je później ponownie wykorzystywać, tak by jak najwięcej zdobytej wiedzy używać ponownie.

Marcin Karbowniczek, EP