helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

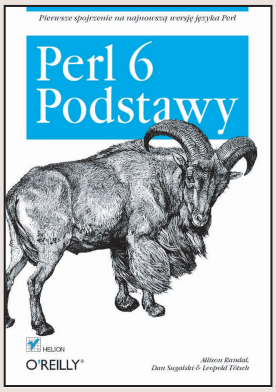
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Perl 6. Podstawy



Autorzy: Allison Randal,
Dan Sugalski, Leopold Totsch
Tłumaczenie: Rafał Szpoton
ISBN: 83-7361-299-8
Tytuł oryginału: Perl 6 Essentials
Format: B5, stron: 216

Książka „Perl 6. Podstawy” jest krótkim przeglądem projektu Perl 6, będącego powszechnie oczekiwaną, zupełnie nową wersją języka programowania Perl. Projekt ten znajduje się wciąż w fazie rozwoju i jest efektem wysiłku całej społeczności Perla, mającym na celu utrzymanie go pośród języków programowania XXI wieku.

Wiele osób wciąż zaangażowanych jest w rozwój Perla 5; w tym samym czasie główna grupa programistów Perla rozpoczęła pracę nad językiem Perl 6, nową, napisaną zupełnie od podstaw wersją języka. Chociaż w wersji tej wciąż utrzymana zostanie filozofia Perla oraz jego powszechnie znana składnia, to jednak wszystkie inne elementy języka są ponownie analizowane i tworzone od podstaw.

Książka „Perl 6. Podstawy” to przegląd bieżącego stanu rozwoju języka Perl dla wszystkich dotychczasowych jego użytkowników, jak również dla początkujących programistów, którzy swoje pierwsze programistyczne doświadczenia wiążą z Perlem. Napisana przez członków głównego zespołu programistów języka książka prezentuje wyjaśnienie różnych etapów projektu, stanowiąc jednocześnie materiał referencyjny dla programistów, którzy są zainteresowani planowanymi zmianami, jak również dla tych, którzy chcieliby przyłączyć się do projektu. Książka ta z pewnością zaspokoi ich ciekawość i ukaże, w jaki sposób zmiany wprowadzone do języka uczynią z Perla jeszcze potężniejsze i łatwiejsze do stosowania narzędzie. „Perl 6. Podstawy” stanowi pierwszą książkę oferującą możliwość wejrzenia w kolejną główną wersję języka Perl. Stanowi ona niezbędną lekturę dla wszystkich osób zainteresowanych przyszłością tego doskonałego narzędzia.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Przedmowa *5*

Rozdział 1. Przegląd projektu *9*

Narodziny języka Perl 6 9

Początek… 10

Trwająca misja 12

Rozdział 2. Rozwój projektu *17*

Rozwój języka 17

Rozwój maszyny Parrot 19

Rozdział 3. Filozofia projektu *25*

Czynniki lingwistyczne oraz poznawcze 25

Problemy architektury 32

Rozdział 4. Składnia *35*

Zmienne 36

Operatory 43

Struktury określające przepływ kontroli 59

Procedury 66

Klasy oraz obiekty 71

Gramatyki oraz reguły 74

Rozdział 5. Struktury wewnętrzne 83
Reguły projektu struktur wewnętrznych 83
Architektura maszyny wirtualnej Parrot 84
Interpreter 89
Operacje wejścia-wyjścia, zdarzenia, sygnały oraz wątki 95
Obiekty 101
Funkcje zaawansowane 104
Konkluzja 108

Rozdział 6. Język symboliczny maszyny Parrot 109
Przygotowanie do pracy 109
Podstawy 111
Praca z obiektami PMC 127
Przepływ sterowania 131
Stosy oraz ramki rejestrów 134
Zmienne leksykalne oraz globalne 137
Procedury 141
Tworzenie testów 149
Krótki podręcznik PASM 151

Rozdział 7. Pośredni kompilator kodu 175
Przygotowanie do pracy 175
Przygotowanie do pracy 176
Przepływ sterowania 186
Procedury 189
Parametry wiersza poleceń kompilatora IMCC 194
Krótki podręcznik IMCC 197

Skorowidz 205

3

Filozofia projektu

Dzisiejsza praktyka jest często niczym więcej
niż tylko zaakceptowaną postacią wczorajszej teorii

— Kenneth Pike
„Wprowadzenie do Tagemiki"

U podstaw każdego języka leży zestaw podstawowych idei nadających mu kierunek rozwoju oraz określających jego przeznaczenie. Osoby pragnące zrozumieć wybory dokonywane przez projektantów języka (dlaczego wybierają tę właściwość, a nie inną lub określony sposób jej wyrażenia), powinny najlepiej rozpocząć od prześledzenia rozumowania leżącego u podstaw tych wyborów.

Na język Perl 6 oddziałuje unikatowy zestaw czynników. Sam język jest głęboko zakorzeniony w systemie Unix, oraz jego systemach pochodnych, co wywiera szczególny wpływ na jego przydatność oraz praktyczne zastosowania. Stanowi on efekt akademickiej rywalizacji na polu nauk informatycznych oraz inżynierii programowania, co budzi chęć rozwiązywania problemów we właściwy sposób, a nie za pomocą środków doraźnych. Sam język jest głęboko przesiąknięty tradycjami lingwistyki oraz antropologii, w związku z tym jako cel stawia się wygodną adaptację do postaci używanej przez człowieka. Właśnie te czynniki oraz jeszcze wiele innych definiują kształt języka Perl oraz kierunek jego rozwoju.

Czynniki lingwistyczne oraz poznawcze

Perl jest językiem ludzkim. Istnieje jednak wiele znaczących różnic pomiędzy Perlem a językami naturalnymi w rodzaju języka angielskiego, francuskiego, niemieckiego itd. Po pierwsze, został on skonstruowany w sposób sztuczny, a nie naturalny. Jego podstawowe zastosowanie, czyli określenie zestawu instrukcji, którymi kierować się będzie maszyna, odzwierciedla jedynie ograniczony zakres czynności występujących w życiu ludzkim. Niemniej jednak Perl jest językiem, którego ludzie używają do porozumiewania się.

Wiele tych samych procesów umysłowych, które prowadzą do formułowania myśli w słowie oraz piśmie jest powielanych w postaci prowadzącej do utworzenia kodu. Proces nauki używania Perla jest bardzo podobny do nauki używania drugiego języka. Istotne są również procesy umysłowe związane z czytaniem. Chociaż podstawowym adresatem kodu napisanego w języku Perl jest maszyna, bardzo często to ludzie muszą odczytać kod podczas jego tworzenia, przeglądania oraz utrzymywania.

Na wiele decyzji podejmowanych podczas projektowania Perla duży wpływ miały założenia języków naturalnych. Poniżej zostanie przedstawionych kilka najważniejszych, to znaczy tych, które powracają wciąż podczas projektowania oraz tych, które wywarły największy wpływ podczas całego procesu.

Teoria złożoności (teoria łóżka wodnego)

Naturalną tendencją występującą w językach ludzkich jest utrzymywanie stałej ogólnej złożoności, gdy weźmiemy pod uwagę zarówno dwa różne języki, jak i zmiany danego języka wraz z upływem czasu. Podobnie jak w przypadku łóżka wodnego, jeżeli zmniejszona zostanie złożoność w jednej części języka, automatycznie zwiększy się ona w innym miejscu. Język z obszernym systemem dźwięków (fonologią) mógłby mieć prostszą składnię. Język mający ograniczony zestaw dźwięków mógłby mieć bardziej złożony sposób tworzenia słów z mniejszych elementów (morfologię). Żaden język nie jest skomplikowany w równym stopniu pod każdym z wymienionych względów, ponieważ byłby zupełnie nieprzydatny. Podobnie — żaden język nie jest całkowicie prosty, ponieważ zbyt mała liczba rozróżnień i możliwych do utworzenia kombinacji uczyniłaby go nieprzydatnym w praktyce.

Ta sama zasada funkcjonuje w przypadku języków komputerowych. Wymagają one zachowania stałej równowagi pomiędzy złożonością a prostotą. Ograniczenie możliwych operatorów do małego ich zestawu prowadzi do zwiększenia liczby metod oraz podprocedur definiowanych przez użytkownika. Nie jest to aż tak złe, lecz zachęca do tworzenia kodu, który będzie rozwlekły oraz trudny do odczytania. Z drugiej strony, język, który definiuje zbyt wiele operatorów, zachęca do tworzenia kodu zawierającego skomplikowane wiersze i jest tym samym równie trudny do odczytania. Doskonała równowaga leży gdzieś pośrodku.

Reguła prostoty

Ogólnie mówiąc, preferowane są proste rozwiązania. Prosta składnia jest łatwiejsza do nauki, zapamiętania, wykorzystania w praktyce oraz odczytania. Niemniej jednak reguła ta znajduje się w ścisłym związku z teorią łóżka wodnego. Jednym z niebezpieczeństw, któremu należy zapobiec jest uproszczenie w złym obszarze. Kolejnym jest błędne uproszczenie oraz nadmierne uproszczenie. Niektóre problemy są złożone i wymagają złożonego rozwiązania. Gramatyki języka Perl6 nie są proste. Są one złożone na poziomie języka, w sposób pozwalający na stosowanie prostszych rozwiązań na poziomie użytkownika.

Reguła możliwości przystosowania

Wraz z upływem czasu języki naturalne rozrastają się i zmieniają. Odpowiadają w ten sposób na zmiany występujące w środowisku oraz na nacisk wewnętrzny ze strony posługujących się nimi ludzi. Pojawia się nowe słownictwo, używane do zaspokojenia nowych potrzeb komunikacyjnych. Stare określenia nie są używane, ludzie powoli je zapominają, zaś w ich miejsce pojawiają się nowe, bardziej odpowiednie. Wraz z upływem czasu złożone części systemu mają tendencję do upraszczania się oraz rozbijania na mniejsze. To właśnie proces zmian utrzymuje język w postaci aktywnej oraz odpowiedniej dla ludzi go wykorzystujących. Nie zmieniają się jedynie martwe języki.

Plan rozwoju języka Perl 6 jawnie określa plany przyszłych zmian języka. Nikt nie wierzy w to, że Perl 6.0.0 będzie językiem doskonałym. Jednocześnie jednak nikt nie chce kolejnego procesu zmian tak dramatycznego, jak podczas przejścia do wersji 6 języka Perl. Dlatego też Perl 6 będzie językiem elastycznym i na tyle zdolnym do adaptacji, aby wraz z upływem czasu umożliwiał stopniowe zmiany. Postanowienie to miało wpływ na wiele decyzji podejmowanych podczas projektowania, włącznie z uproszczeniem modyfikacji sposobu parsowania (analizy składniowej) języka, ograniczeniem rozróżniania pomiędzy operacjami podstawowymi języka a operacjami zdefiniowanymi przez użytkownika, jak również ułatwieniem definiowania nowych operatorów.

Reguła ważności

W językach naturalnych określone struktury oraz konstrukcje stylistyczne służą do zwrócenia uwagi na ważne elementy. Może być to akcent, np. w zdaniu „Pies ukradł mój portfel" (pies, a nie człowiek) lub użycie dodatkowego czasownika, np. w zdaniu „To był pies, który ukradł mój portfel", a nawet zmiana kolejności wyrazów „Mój portfel został skradziony przez psa" (mój portfel, a nie but itp.) lub dowolna liczba innych zabiegów.

Perl został zaprojektowany ze swoim własnym zestawem konstrukcji służących do zaznaczenia ważności, z których niektóre zawarte są w samym języku, a niektóre umożliwiają użytkownikom zaakcentowanie ważności w tworzonym przez nich kodzie. Bloki nazwane (`NAMED`) wykorzystują duże litery w celu zwrócenia uwagi, że znajdują się na zewnątrz normalnego przepływu sterowania w programie. Perl 5 posiada dodatkową składnię dla instrukcji kontrolujących wykonywanie programu, w rodzaju `if` oraz `for`, powodującą przesunięcie ich na koniec, co daje funkcjonowanie w charakterze modyfikatorów instrukcji (ponieważ analiza wiersza w kodzie w języku Perl odbywa się od strony lewej do prawej, to właśnie lewa strona jest zawsze ważniejsza). Taka elastyczność została utrzymana w języku Perl 6, a do listy zostały dodane nowe struktury.

Zachowanie równowagi podczas tworzenia projektu polega na podjęciu decyzji, które właściwości zasługują na zaznaczenie jako ważne, a w którym miejscu składnia może być mało uniwersalna, tak aby język był bardziej wyrazisty.

Reguła końcowej wagi

Języki naturalne umieszczają duże elementy złożone na końcu zdań. I chociaż konstrukcje postaci „Dałem Marii książkę" oraz „Dałem książkę Marii" są tak samo wygodne w użyciu, to jednak zdanie „Dałem książę o historii rozwoju w Indonezji produktów opartych na orzeszkach ziemnych Marii" jest znacznie mniej wygodne niż jego odpowiednik. Dotyczy to głównie problemu z analizą takiego zdania przez mózg człowieka. Znacznie prostsze jest jednorazowe zinterpretowanie głównych bloków zdania niż rozpoczęcie od kilku pierwszych, przeanalizowanie długiego łańcucha mniej ważnych informacji, by w końcu powrócić do struktury zdania nadrzędnego. Pamięć ludzka jest ograniczona.

Reguła ta jest jednym z powodów, dla których modyfikatory wyrażeń regularnych zostały w Perlu 6 przesunięte na początek. Znacznie prostsze jest odczytanie reguły gramatycznej, kiedy od samego początku wiadomo, na przykład, że „na tę regułę nie ma wpływu wielkość liter". (Ułatwia to również znacznie rozkład składniowy dokonywany przez komputer, co jest prawie równie istotne).

Reguła ta jest również przyczyną, dla której pojawił się pomysł zmiany kolejności instrukcji `grep` na:

```
grep @array { potencjalnie długi i skomplikowany blok };
```

Jednakże tego rodzaju zmiana wywołuje takie wątpliwości w społeczności, że może nigdy nie zostać wprowadzona.

Reguła kontekstu

Podczas interpretacji znaczenia języki naturalne wykorzystują kontekst. Znaczenia określeń „gorący" oraz „gorący dzień", „gorący utwór", „gorący pomysł" oraz „gorąca dyskusja" są całkiem różne. Niejawne znaczenie określenia „jest mokro" zmienia się w zależności od tego, czy stanowi ono odpowiedź na pytanie „Czy powinienem zabrać płaszcz?" czy też „Dlaczego pies biega po kuchni?". To właśnie otaczający kontekst pozwala na rozróżnienie tych znaczeń. Kontekst pojawia się również w innych dziedzinach. Obraz przedstawiający abstrakcyjną pomarańczową sferę zostanie zinterpretowany odmiennie w zależności od tego, czy inne elementy na obrazie przedstawiają banany, klownów czy piłkarzy. Umysł ludzki stara się zrozumieć otaczający świat i używa w tym celu wszystkich dostępnych podpowiedzi.

Perl od zawsze był językiem kontekstowym. Używa on kontekstu na szereg różnych sposobów. Najbardziej oczywistym jest wykorzystanie kontekstowe skalarów oraz list — w tym przypadku zmienna może zwrócić różne wartości w zależności od tego, w którym miejscu została użyta oraz w jaki sposób. W Perlu 6 zostało to jeszcze rozszerzone o kontekstowe użycie łańcuchów znakowych, wartości logicznych, numerycznych oraz innych. Innym przypadkiem wykorzystania kontekstu jest użycie wartości domyślnych `$_` w instrukcji `print`, `chomp`, `when` lub dopasowaniach.

Właściwości zależne od kontekstu utrudniają utworzenie interpretera dokonującego ich analizy, lecz są prostsze dla ludzi, którzy używają ich codziennie. Pasują one bowiem do sposobu, w jaki ludzie myślą naturalnie, a jest to jednym z założeń języka Perl.

Reguła ZCMM

W językach naturalnych występuje pojęcie „rodowitej intuicji rozmówcy". Ktoś, kto biegle posługuje się językiem, będzie w stanie stwierdzić czy zdanie jest poprawne, nawet jeśli nie potrafi on racjonalnie wyjaśnić odpowiednich reguł. (Ma to niewielki wpływ na trudność, na jaką natrafiają nauczyciele języka angielskiego, próbując wymusić stosowanie przez studentów „poprawnej" gramatyki. Zasady formalnego języka pisanego różnią się znacznie od tych dla języka mówionego).

Dane właściwości języka powinny dawać w sposób oczywisty efekty oczekiwane przez użytkownika. Właśnie ta idea „ZCMM", to znaczy „Zrób Co Mam na Myśli" stanowi głównie kwestię intuicji. Na oczekiwania użytkownika mają wpływ jego doświadczenia, znajomość języka oraz podłoże kulturowe. Oznacza to, że intuicja jest specyficzna dla danego człowieka. Osoba posługująca się językiem angielskim nie będzie oczekiwać tych samych rezultatów, co osoba posługująca się językiem holenderskim, zaś programista języka Ada nie będzie oczekiwać tych samych efektów, co programista języka Cobol.

Sztuczka zastosowana w projekcie polega na wykorzystaniu intuicji programisty, a nie walce z nią. Jasno zdefiniowany zestaw reguł nigdy nie dorówna potędze właściwości, która „po prostu wydaje się być poprawna".

Perl 6 przeznaczony jest dla programistów języka Perl. To, co może wydawać się poprawne dla jednego programisty Perla może nie wydawać się poprawne dla innego, dlatego nie istnieje właściwość, która mogłaby zadowolić wszystkich. Jednakże w większości przypadków możliwe jest dogodzenie większości osób.

Ogólnie mówią, Perl adresowany jest do osób używających języka angielskiego. Wykorzystuje on na przykład słowa w rodzaju „given", co osobom znającym ten język ułatwia zrozumienie działania kodu. Oczywiście nie wszyscy programiści używający tego języka programowania znają język angielski. W niektórych przypadkach użyty język jest uproszczony w celu zadowolenia szerszej grupy adresatów. W przypadku reguł gramatycznych oryginalne modyfikatory mają postać *1st*, *2nd*, *3rd*, *4th* itp. ponieważ są one najbardziej naturalne dla osób posługujących się językiem angielskim od urodzenia. Mają one jednak również postać alternatywną *1th*, *2th*, *Nth* itp., ponieważ zakończenia używane w języku angielskim w przypadku liczb są chaotyczne oraz niezbyt przyjazne dla osób dopiero uczących się tego języka

Reguła ponownego wykorzystania

Języki ludzkie mają tendencję do zawierania ograniczonego zestawu struktur oraz powtórnego ich wykorzystywania w różnych kontekstach. Języki programowania również stosują zestaw zwykłych konwencji składniowych. Język, który w celu ograniczenia pętli

wykorzystywałby nawias `{...}`, lecz do ograniczenia instrukcji `if` używałby pary słów kluczowych (na przykład `if ... then ... end if`), byłby strasznie denerwujący. Istnienie zbyt wielu reguł utrudniałoby odnalezienie wzorca.

Jeżeli w czasie tworzenia projektu istnieje określona składnia służąca do wyrażenia jednej właściwości, to w celu wyrażenia związanej z nią innej właściwości często znacznie lepszym wyjściem jest użycie ponownie tej samej składni od wynajdowania czegoś zupełnie nowego. Nadaje to językowi ogólną spójność i ułatwia zapamiętanie nowych właściwości. Właśnie z tego powodu gramatyki mają postać klas[1]. Mogłyby one używać dowolnej składni, lecz klasy wyrażają wiele właściwości wymaganych przez gramatyki, np. dziedziczenie oraz ideę tworzenia egzemplarza.

Reguła rozróżnienia

Mózg człowieka znacznie łatwiej dostrzega duże różnice niż małe. Na przykład znacznie prostsze do rozróżnienia są słowa „kot" oraz „pies" niż słowa „bieg" i „brzeg". Zazwyczaj odpowiednich wskazówek dostarcza kontekst, jeżeli jednak „kot" określany byłby jako „kies", musielibyśmy bez końca poprawiać osoby, które źle nas usłyszały („Nie, powiedziałem, że Kowalscy mają nowego psa, a nie ksa".).

Głównym problemem podczas projektowania języka jest utworzenie widocznych wskazówek, dających subtelne kontrasty. Język powinien unikać tworzenia zbyt wielu różnych rzeczy podobnych do siebie. Zbytnie przeciążenie zmniejsza przejrzystość i zwiększa prawdopodobieństwo pomyłki. Stanowi to część uzasadnienia powodu rozbicia dwóch znaczeń konstrukcji `eval` na `try` oraz `eval`, dwóch znaczeń konstrukcji `for` na `for` oraz `loop`, jak również dwóch znaczeń `sub` na `sub` oraz `method`.

Reguły rozróżnienia oraz ponownego wykorzystania konstrukcji są w stałym związku. Jeżeli zbyt wiele właściwości będzie ponownie wykorzystywanych oraz przeciążonych, język stanie się niejasny. Zbyt wiele czasu będzie trzeba poświęcić na domyślanie się dokładnego znaczenia. Jeżeli jednak zbyt wiele właściwości będzie całkowicie różnych od siebie, język straci swoją spójność. Ponownie jest to problem wypracowania konsensusu.

Język nie może być oddzielony od kultury

Język naturalny bez społeczności osób posługujących się nim jest językiem martwym. Może być przedmiotem studiów z pobudek czysto akademickich, lecz jeżeli nikt nie podejmie wysiłku jego zachowania, prędzej czy później zaniknie zupełnie. Język nadaje społeczności poczucie tożsamości, gdy społeczność uznaje go za odpowiedni i przekazuje kolejnym pokoleniom. Kultura społeczności nadaje kształt językowi i wyznacza mu cel istnienia.

---

[1] Więcej informacji szczegółowych dotyczących gramatyk znajduje się w podrozdziale rozdziału 4., zatytułowanym „Gramatyki oraz reguły".

Również języki komputerowe są zależne od społeczności ich używających. Możesz mierzyć jej wielkość liczbą firm oferujących wsparcie dla danego języka, wierszami kodu znajdującymi się w użyciu, czy też zainteresowaniem użytkowników, jednak wszystko sprowadza się do jednej kwestii: nieużywany język programowania staje się martwy. Ostatnim znakiem śmierci języka jest sytuacja, gdy nie istnieje żaden jego kompilator lub interpreter, który mógłby działać na istniejącym sprzęcie oraz w istniejących systemach operacyjnych.

Na etapie projektowania oznacza to, że samo dopasowanie właściwości języka do siebie nie jest wystarczające. Ważna jest również tradycja społeczności oraz jej oczekiwania, zaś niektóre zmiany mają swoją cenę kulturową.

Reguła wolności

W językach naturalnych zawsze istnieje więcej niż jeden sposób wyrażenia idei. Autor lub rozmówca ma wolność wyboru najlepszego określenia i ponosi za nie odpowiedzialność. Powinien wybrać taki sposób wyrażenia idei, aby był on logiczny dla adresatów.

Perl zawsze oparty był na założeniu, że programiści powinni mieć wolność wyboru sposobu, w jaki utworzą swój kod. Udostępnia on łatwy dostęp do potężnych właściwości i pozostawia użytkownikom możliwość ich mądrego wykorzystania. Określa raczej zwyczaje oraz konwencje, a nie prawa.

Tego rodzaju reguła ma pod kilkoma względami wpływ na projekt. Jeżeli dana właściwość jest korzystna dla całego języka, nie powinna być odrzucona jedynie dlatego, że ktoś mógłby użyć jej w głupi sposób. Z drugiej strony, nie jesteśmy za tworzeniem niektórych skomplikowanych właściwości, jeżeli miałyby być one bardzo rzadko wykorzystywane.

Innym elementem wyzwania projektowego jest utworzenie narzędzi, które będą mieć wiele zastosowań. Nikt nie chce książki kucharskiej, którą czyta się jak powieść Stephena Kinga, nikt również nie chciałby w jednym wierszu umieszczać struktury definicji klasy. Język musi być na tyle elastyczny, aby dawać wolność.

Reguła zapożyczeń

Zapożyczenia są w językach naturalnych bardzo powszechne. W momencie gdy nowa technologia (jedzenie, ubrania itp.) jest importowana z innej kultury, całkiem naturalne jest zaadoptowanie również jej oryginalnej nazwy. Zapożyczane słowa są stopniowo adaptowane do nowego języka. W języku angielskim na przykład nikt nie wymawia słów w rodzaju „tortilla", „lasagna" lub „champagne" dokładnie w ten sam sposób, co w językach, z których te słowa pochodzą. Zostały one zmienione tak, aby pasowały do wymowy języka angielskiego.

Język Perl zawiera właściwości zapożyczone i tak samo będzie w przypadku języka Perl 6. Nie ma nic wstydliwego w przyznaniu się, że inny język wykonał wspaniałe zadanie, implementując określoną właściwość. Znacznie lepsze jest otwarte zapożyczenie dobrej

właściwości niż udawanie, że jest ona oryginalna. Perl nie musi być inny tylko z tego powodu, że jest inny. Większość właściwości jednak nie zostanie zaadoptowana bez jakichkolwiek zmian. Każdy język posiada swoje własne konwencje oraz składnię, a wiele z nich nie jest ze sobą zgodnych. Dlatego też właściwości są zapożyczane w Perlu przy użyciu odpowiednich konstrukcji języka służących do ich wyrażenia.

Problemy architektury

Drugi zestaw reguł dotyczy ogólnej architektury języka Perl 6. Reguły te są powiązane z przeszłością, teraźniejszością oraz przyszłością języka i definiują podstawowe przeznaczenie Perla. Żadna z reguł nie istnieje samodzielnie. Każda z nich dotyczy pozostałych.

Perl powinien zostać Perlem

Wszyscy zgodzą się, że Perl 6 powinien zostać Perlem. Pojawia się jednak pytanie, co tak naprawdę to oznacza? Nie oznacza to, że Perl 6 będzie posiadać dokładnie tę samą składnię. Nie oznacza również, że będzie on mieć te same właściwości. Gdyby tak było, Perl 6 byłby po prostu wersją 5 tego języka. Dlatego też istota tego pytania dotyczy kwestii, które definiują język jako „Perl".

Zachowanie przeznaczenia

Perl zachowa oryginalne przeznaczenie zamierzone przez jego twórcę. Larry pragnął utworzyć język, który mógłby wykonać postawione mu zadanie, lecz nie upierał się przy swoim podejściu. Język musi być wystarczająco potężny do wykonania złożonych zadań, lecz wciąż lekki i uniwersalny. Jak stwierdził to Larry, „Perl ułatwia rzeczy proste i umożliwia wykonanie rzeczy trudnych". Podstawowa filozofia dotycząca projektu Perla pozostała niezmienna. W Perlu 6 proste rzeczy są nieco prostsze, zaś rzeczy trudne są łatwiejsze do wykonania.

Znajomość

Perl 6 będzie znajomy dla użytkowników języka Perl 5. Podstawowa składnia jest wciąż taka sama. Została tylko nieco wyczyszczona i jest bardziej spójna. Podstawowy zestaw właściwości jest wciąż ten sam. Język dodaje kilka nowych i potężnych właściwości, które najprawdopodobniej zmienią sposób programowania przy ich użyciu, lecz nie będą one wymagane.

Nauka języka Perl 6 będzie raczej przypominać poznawanie odmiany australijskiej języka angielskiego przez obywateli Stanów Zjednoczonych niż poznawanie języka japońskiego przez obywateli Wielkiej Brytanii. Oczywiście istnieją pewne zmiany w zakresie słownictwa i akcent jest nieco inny, lecz bez żadnych wątpliwości jest to wciąż język angielski.

Możliwość przetłumaczenia

Będzie istnieć możliwość automatycznego przetłumaczenia kodu z języka Perl 5 do Perl 6. W dłuższej perspektywie nie jest to aż tak istotne jak tworzenie kodu w języku Perl 6, lecz podczas fazy przejścia z jednej wersji języka do drugiej możliwość automatycznego tłumaczenia będzie ważna. Pozwoli ona programistom na rozpoczęcie pracy z językiem zanim zrozumieją każdy subtelny niuans wszystkich zmian. Perl zawsze spełniał założenie polegające na preferowaniu nauki tego, co jest potrzebne w danej chwili i uczeniu się stopniowo pozostałych rzeczy.

Ważne nowe cechy

Perl 6 będzie zawierać wiele nowych właściwości, takich jak wyjątki, delegacje zadań, podział na wiele metod, kontynuacje, współprogramy. Udowodniły one swoją przydatność w innych językach, gdyż mają olbrzymią moc w przypadku rozwiązywania określonych problemów. Zwiększają również stabilność oraz uniwersalność języka.

Wiele z nich jest tradycyjnie trudnych do zrozumienia. Perl stosuje to samo podejście co zawsze: udostępnia potężne narzędzia, ułatwia ich stosowanie oraz pozostawia użytkownikowi decyzję czy i kiedy je wykorzystać. Większość użytkowników w chwili wykorzystywania metody `assuming` prawdopodobnie nie będzie nawet wiedzieć, że używa aliasów parametryzowanych.

Tego rodzaju właściwości stanowią ważny element przygotowywania Perla na przyszłość. Kto wie, jakie paradygmaty mogą powstać w przyszłości, przy wykorzystaniu połączenia tych zaawansowanych właściwości, udostępnionych w postaci przyjaznej dla przeciętnego programisty. Nie musi to być rewolucja, lecz z pewnością będzie to ewolucja.

Przydatność długoterminowa

Perl 6 nie jest wersją języka, która ma przetrwać kilka lat, a następnie zostać odrzucona. Ma on w założeniu przetrwać 20 lat i dłużej. Właśnie ta długoterminowa wizja języka wpływa na jego kształt i proces jego tworzenia. Nie jesteśmy zainteresowani implementowaniem kilku ekscytujących sztuczek. Pragniemy silnych zależnych narzędzi, dysponujących dużą przestrzenią do rozwoju. I nie jesteśmy przerażeni perspektywą poświęcenia nieco większej ilości dodatkowego czasu teraz, aby utworzyć je poprawnie. Nie oznacza to, że Perl 6.0 będzie idealny w porównaniu z innymi jego wersjami. Stanowi to jedynie kolejny krok w jego rozwoju.