Technologia i rozwiązania

JavaScript i wzorce projektowe

Programowanie dla zaawansowanych

Wydanie II

Simon Timms

Helion

[PACKT] open source* PUBLISHING community experience distilled

Tytuł oryginału: Mastering JavaScript Design Patterns, Second Edition

Tłumaczenie: Piotr Pilch

ISBN: 978-83-283-3194-5

First published in the English language under the title 'Mastering JavaScript Design Patterns – Second Edition (9781785882166)'

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/jswpz2.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/jswpz2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

1

ROZDZIAŁ

Projektowanie dla przyjemności i zysku

JavaScript to stale rozwijający się język, który od początku swojego istnienia pokonał długą drogę. Być może w większym stopniu niż jakikolwiek inny język programowania rozrastał się i zmieniał wraz z rozwojem technologii WWW (ang. *World Wide Web*). Celem książki jest przedstawienie, jak kod JavaScript może być pisany z wykorzystaniem dobrych praktyk projektowych. W przedmowie książki zamieszczono szczegółowe objaśnienie jej poszczególnych części.

W pierwszej połowie rozdziału przybliżymy historię języka JavaScript, a także to, jak stał się ważnym językiem, jakim jest obecnie. Wraz ze wzrostem znaczenia i rozwojem tego języka zwiększała się również potrzeba stosowania rygorystycznych metod jego rozbudowy. Wzorce projektowe mogą być bardzo przydatnym narzędziem, które pomaga w tworzeniu łatwego w utrzymaniu kodu. Drugą połowę rozdziału poświęcimy teorii wzorców projektowych. Na koniec przyjrzymy się w skrócie antywzorcom.

W rozdziale omówiono następujące zagadnienia:

- Historia języka JavaScript.
- Czym jest wzorzec projektowy?
- Antywzorce.

Droga do powstania języka JavaScript

Nigdy nie dowiemy się, jakie były początki ludzkiego języka. Czy powoli ewoluował z serii chrząknięć i gardłowych dźwięków wydawanych podczas codziennych rytuałów? Być może język powstał, aby umożliwić komunikację matkom i ich potomstwu. Oba warianty to teoria,

ale niemożliwa do udowodnienia. Nikt nie miał okazji obserwować naszych przodków w tym ważnym okresie. Okazuje się, że ogólny brak dowodu empirycznego spowodował, iż towarzystwo językoznawcze z Paryża odrzuciło dalsze związane z tym dyskusje, postrzegając je jako nieodpowiedni temat poważnych badań.

Początki

Na szczęście języki programowania rozwinęły się w nie tak odległych czasach, dlatego mamy możliwość obserwowania ich ewolucji i zmian. JavaScript ma jedną z bardziej interesujących historii wśród współczesnych języków programowania. W ciągu dziesięciu majowych dni 1995 roku, które musiały być absolutnie szalone, programista z firmy Netscape stworzył podstawy tego, co rozwinęło się do postaci obecnego języka JavaScript.

W tamtym czasie firma Netscape prowadziła z Microsoftem pierwszą z wojen związanych z przeglądarkami. Wizja firmy Netscape była o wiele donioślejsza niż samo rozwijanie przeglądarki. Firmie zależało na stworzeniu całego rozproszonego systemu operacyjnego, który korzystałby z będącego wtedy nowością języka programowania Java firmy Sun Microsystems. Java stanowiła znacznie nowocześniejszą alternatywę wobec języka C++, który był propagowany przez Microsoft. Firma Netscape nie miała jednak odpowiedzi na język Visual Basic. Był on językiem programowania łatwiejszym w użyciu, który kierowano do projektantów z mniejszym doświadczeniem. Język Visual Basic eliminował pewne problemy związane z zarządzaniem pamięcią, które sprawiają, że programowanie z wykorzystaniem języków C i C++ jest znacznie utrudnione. Visual Basic zapewniał również typowanie silne i generalnie oferował większą swobodę. Na następnej stronie przedstawiono ilustrację z osią czasu dla języka JavaScript .

Brendan Eich otrzymał zadanie polegające na zaprojektowaniu języka, który byłby odpowiedzią firmy Netscape na Visual Basic. Początkowo projekt miał nazwę kodową Mocha, ale przed pojawieniem się przeglądarki Netscape 2.0 w wersji beta jego nazwa została zmieniona na LiveScript. W momencie gdy udostępniono pełną wersję języka Mocha/LiveScript, nadano mu nazwę JavaScript, aby powiązać go z integracją apletów Java. Były to niewielkie aplikacje działające w obrębie przeglądarki. Aplety te miały model zabezpieczeń inny niż sama przeglądarka, dlatego nie były ograniczone w tym, jak mogły prowadzić interakcję zarówno z przeglądarką, jak i systemem lokalnym. Obecnie dość rzadko można spotkać się z apletami, gdyż spora część ich funkcji stała się częścią przeglądarki. Język Java był w tamtym czasie na fali, dlatego wszelkie związki z nim były szczególnie widoczne.

Przez lata nazwa JavaScript powodowała wiele niejasności. JavaScript to język bardzo różniący się od języka Java. JavaScript to język interpretowany z typowaniem słabym, którego kod jest uruchamiany głównie w przeglądarce. Java to język, którego kod kompilowany jest do postaci kodu bajtowego wykonywanego następnie przez wirtualną maszynę Java. Język Java może zostać zastosowany w wielu sytuacjach, od przeglądarek (za pośrednictwem apletów Java) po serwery (Tomcat, JBoss itp.) i w pełni funkcjonalne aplikacje (Eclipse, OpenOffice itp.). W umysłach większości laików pozostaje mętlik.

Język JavaScript okazał się naprawdę przydatny w przypadku interakcji z przeglądarką internetową. Nie minęło wiele czasu, a firma Microsoft również zaadaptowała ten język w swojej przeglądarce Internet Explorer, aby uzupełniał język VBScript. Implementacja Microsoftu była znana pod nazwą JScript.

Pod koniec 1996 r. było jasne, że język JavaScript okazał się zwycięskim językiem do tworzenia stron internetowych, który będzie dominował w najbliższej przyszłości. Aby ograniczyć liczbę różnic językowych między implementacjami, firmy Sun i Netscape rozpoczęły współpracę z organizacją **ECMA** (ang. *European Computer Manufacturers Association*) w celu opracowania standardu, z którym byłyby zgodne przyszłe wersje języka JavaScript. Standard został udostępniony bardzo szybko (zwłaszcza biorąc pod uwagę to, jak dynamicznie działają organizacje standaryzujące), czyli w lipcu 1997 r. Na wypadek gdyby nie istniała już wystarczająca liczba nazw odnoszących się do języka JavaScript, standardowej wersji została nadana nazwa **ECMAScript**, która nadal się utrzymuje w niektórych kręgach.

Niestety standard określał jedynie bardzo podstawowe części języka JavaScript. Biorąc pod uwagę nasilające się wojny związane z przeglądarkami, oczywistym stało się, że dowolny dostawca, który pozostanie tylko przy podstawowej implementacji tego języka, szybko znajdzie się w tyle za konkurencją. Jednocześnie czynione były intensywne starania w celu ustanowienia dla przeglądarek standardu modelu **DOM** (ang. *Document Object Model*). Model ten był w rzeczywistości interfejsem API strony internetowej, która mogła być modyfikowana za pomocą kodu JavaScript.

Przez wiele lat każdy skrypt JavaScript zaczynał działanie od podjęcia próby zidentyfikowania przeglądarki, w której został uruchomiony. Od tego zależny był sposób adresowania elementów w modelu DOM, co wynikało ze znacznych różnic między poszczególnymi przeglądarkami. Legendarny jest już wyjątkowo złożony kod wymagany do wykonania prostych działań. Pamiętam, jak przez rok czytałem 20-częściową serię poświęconą projektowaniu menu rozwijanego opartego na kodzie **DHTML** (ang. *Dynamic HTML*), które zadziała zarówno w przeglądarce Internet Explorer, jak i w programie Netscape Navigator. Taka sama funkcjonalność może obecnie zostać uzyskana z wykorzystaniem czystego kodu CSS, nawet bez konieczności sięgania po kod JavaScript.

> DHTML to termin, który cieszył się popularnością pod koniec lat 90. i na początku obecnego wieku. W rzeczywistości odnosił się do każdej strony internetowej, która zawierała dowolnego rodzaju treść dynamiczną z kodem wykonywanym po stronie klienta. Język DHTML wyszedł z użycia, gdy popularność języka JavaScript sprawiła, że obecnie niemal każda strona zawiera elementy dynamiczne.

Na szczęście w tle kontynuowane były starania mające na celu standaryzację języka JavaScript. Wersje 2 i 3 języka ECMAScript pojawiły się odpowiednio w 1998 r. i 1999 r. Wyglądało na to, że być może w końcu dojdzie do porozumienia między różnymi stronami zainteresowanymi językiem JavaScript. Prace nad językiem ECMAScript 4, który miał być kolejną wersją wnoszącą istotne zmiany, rozpoczęły się na początku 2000 r.

Przerwa

Wtedy nadeszła katastrofa. Różne grupy zaangażowane w rozwijanie języka ECMAScript zaczęły mieć znacząco różne zdania w kwestii kierunku, w którym miał podążać język JavaScript. Firma Microsoft sprawiała wrażenie utraty zainteresowania działaniami związanymi ze standaryzacją. W pewnym sensie mogło to być zrozumiałe, ponieważ był to mniej więcej okres, gdy firma Netscape dokonała autodestrukcji, a przeglądarka Internet Explorer stała się rzeczywistym standardem. Microsoft zaimplementował niektóre części języka ECMAScript 4, ale nie wszystkie. Inne firmy zapewniły bardziej kompleksową obsługę tego języka, ale z powodu braku udziału lidera rynku projektanci nie byli zainteresowani korzystaniem z tego.

Mijały kolejne lata bez konsensusu i nowej wersji języka ECMAScript. Jak to jednak często bywa, ewolucja internetu nie zatrzymała się mimo braku porozumienia między głównymi graczami. Biblioteki takie jak jQuery, Prototype, Dojo i Mootools wyeliminowały główne różnice obecne w przeglądarkach. Dzięki temu projektowanie dla różnych przeglądarek stało się o wiele łatwiejsze. Jednocześnie dramatycznie zwiększyła się ilość kodu JavaScript używanego w aplikacjach.

Pojawienie się aplikacji GMail

Być może punktem zwrotnym było udostępnienie w 2004 r. aplikacji GMail przez firmę Google. Choć w chwili wydania tej aplikacji technologia XMLHTTPRequest, na której z kolei bazuje technologia **AJAX** (ang. *Asynchronous JavaScript and XML*), była już dostępna od około pięciu lat, nie była odpowiednio wykorzystywana. W momencie pojawienia się aplikacji GMail byłem całkowicie zaskoczony tym, jak płynnie działa. Przyzwyczailiśmy się do programów, które unikają operacji pełnego ponownego ładowania, ale w tamtym czasie była to rewolucja. Aby zapewnić takie działanie, niezbędna była spora ilość kodu JavaScript.

> AJAX to metoda pozwalająca na uzyskiwanie przez klienta niewielkich porcji danych z serwera zamiast odświeżania całej strony. Technologia ta umożliwia uzyskanie bardziej interaktywnych stron, które unikają znacznego obciążenia wynikającego z ponownego ładowania całej strony.

Popularność aplikacji GMail wywołała zmianę, na którą zanosiło się od jakiegoś czasu. Zwiększający się poziom akceptacji i standaryzacji języka JavaScript sprawił, że wreszcie zaaprobowano ten język jako właściwy. Do tego momentu język JavaScript był używany głównie do wprowadzania niewielkich zmian w obrębie strony oraz sprawdzania poprawności danych wprowadzonych w formularzu. Żartuję sobie z innymi osobami, że na początku istnienia języka JavaScript `Validate()` była jedyną używaną nazwą funkcji.

Aplikacje takie jak GMail, które w dużym stopniu bazują na technologii AJAX i unikają ponownego pełnego ładowania stron, są nazywane aplikacjami **SPA** (ang. *Single Page Application*). Dzięki minimalizowaniu zmian zawartości stron użytkownikom zapewniana jest większa płynność obsługi. Przesyłanie wyłącznie ładunku danych **JSON** (ang. *JavaScript Object Notation*), a nie kodu

HTML, powoduje również zminimalizowanie wymagań dotyczących przepustowości. Dzięki temu aplikacje sprawiają wrażenie działających efektywniej. W ostatnich latach dokonano znaczących postępów w dziedzinie środowisk ułatwiających tworzenie aplikacji SPA. AngularJS, backbone.js i ember to środowiska oparte na wzorcu MVC (ang. *Model View Controller*). W okresie ostatnich dwóch lub trzech lat zyskały one znacznie na popularności, a ponadto oferują kilka interesujących zastosowań wzorców. Środowiska te są wynikiem wieloletnich doświadczeń prowadzonych przez bardzo mądre osoby stosujące najlepsze procedury związane z językiem JavaScript.

> JSON to zrozumiały dla człowieka format serializacji przeznaczony dla języka JavaScript. W ostatnich latach stał się bardzo popularny, ponieważ jest prostszy i mniej kłopotliwy niż wcześniejsze często używane formaty, takie jak XML. Format JSON pozbawiony jest wielu technologii towarzyszących i ścisłych reguł gramatycznych języka XML, ale wynagradza to swoją prostotą.

Obecnie równolegle z rozwijaniem środowisk korzystających z języka JavaScript trwają prace nad nim samym. W 2015 r. udostępniono osławioną nową wersję języka JavaScript, która była przygotowywana przez kilka lat. Wersja ta nosiła początkowo nazwę ECMAScript 6, a ostatecznie uzyskała nazwę ECMAScript-2015. Pojawiło się w niej kilka znakomitych ulepszeń ekosystemu. Dostawcy przeglądarek spieszą się z zaadaptowaniem standardu. Z powodu złożoności procesu dodawania nowych elementów języka do bazy kodu, a także tego, że nie każdy jest na bieżąco z nowatorskimi technologiami przeglądarek, na popularności zyskuje kilka innych języków dokonujących transkompilacji do kodu JavaScript. CoffeeScript to język przypominający język Python, który ma na celu zwiększenie czytelności i zwięzłości kodu JavaScript. Język Dart, który został zaprojektowany przez firmę Google, jest przez nią promowany jako ostateczny następca języka JavaScript. Budowa języka Dart zapewnia niektóre optymalizacje, które są niemożliwe w przypadku tradycyjnego kodu JavaScript. Do momentu, w którym środowisko uruchomieniowe języka Dart osiągnie wystarczającą popularność, firma Google zapewnia transkompilator przeprowadzający kompilację kodu Dart do postaci kodu JavaScript. TypeScript to projekt firmy Microsoft, w którym do języka JavaScript dodano wybrane elementy języka ECMAScript-2015, a nawet część składni standardu ECMAScript-201X, jak również interesujący system obsługi typów. Celem projektu jest rozwiązanie części problemów obecnych w dużych projektach opartych na kodzie JavaScript.

Można wskazać dwa powody omówienia historii języka JavaScript. Przede wszystkim trzeba zapamiętać, że języki nie są projektowane w próżni. Zarówno języki, jakimi posługują się ludzie, jak i komputerowe języki programowania mutują się w zależności od środowisk, w których są stosowane. Utrzymuje się popularne przekonanie, że Innuici używają sporej liczby słów odnoszących się do śniegu, ponieważ jest on tak wszechobecny w miejscu ich zamieszkiwania. Niekoniecznie musi to być prawdą, zależnie od definicji słowa i tego, kto dokładnie tworzy społeczność Innuitów. Istnieje jednak spora liczba przykładów specyficznych leksykonów, które ewoluują do tego, by spełniać wymagania dotyczące precyzyjnych definicji w konkretnych, wąskich dziedzinach. Nie trzeba daleko szukać. Wystarczy wybrać się do sklepu ze specjalistycznym sprzętem do gotowania, aby znaleźć w nim dużą liczbę wariantów produktów, które laik taki jak ja nazwałby garnkiem.

Hipoteza Sapira-Whorfa funkcjonująca w językoznawstwie wskazuje, że nie tylko środowisko wpływa na język, w którym jest on używany, ale też język ma wpływ na środowisko. Teoria, która nazywana jest również relatywizmem językowym, głosi, że procesy poznawcze różnią się w zależności od tego, jaką konstrukcję ma język. Keith Chen, który zajmuje się psychologią poznawczą, przedstawił fascynujący przykład z tym związany. W cieszącej się bardzo dużą popularnością prezentacji udostępnionej przez serwis TED dr Chen zasugerował, że występuje silna pozytywna korelacja między używaniem języka pozbawionego czasu przyszłego a umiejętnością oszczędzania pieniędzy (*https://www.ted.com/talks/keith_chen_could_your_language_affect_your_ability_to_save_money/transcript*). Zgodnie z hipotezą opracowaną przez dr. Chena, w sytuacji, gdy w używanym języku nie występuje silne poczucie związku teraźniejszości z przyszłością, prowadzi to do bardziej lekkomyślnego zachowania w czasie teraźniejszym.

Oznacza to, że poznanie historii języka JavaScript umożliwia łatwiejsze zrozumienie tego, gdzie i jak go wykorzystać.

Drugi powód, dla którego przybliżyłem historię języka JavaScript, jest taki, że absolutnie fascynujące jest obserwowanie, jak szybko rozwijało się to popularne narzędzie. W momencie pisania tej książki minęło niemal 20 lat od czasu utworzenia pierwszej wersji języka JavaScript, a jego popularność od tej pory zaczęła się gwałtownie zwiększać. Co może być bardziej ekscytującego niż praca z ciągle rozwijającym się językiem?

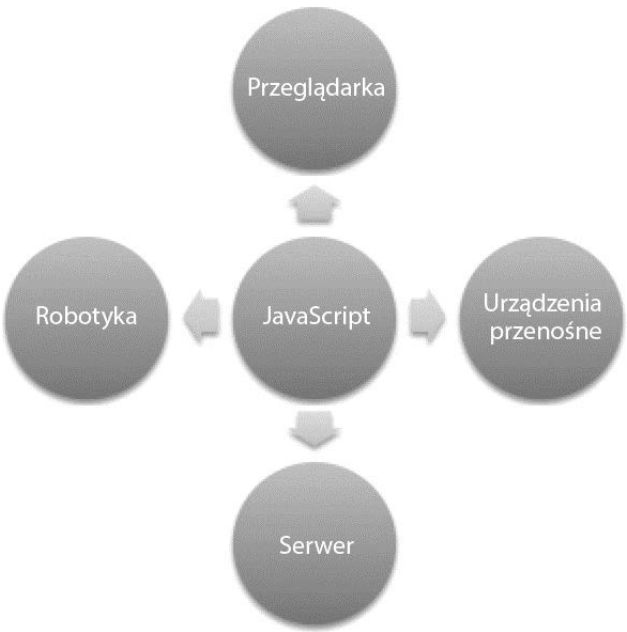
Wszechobecność języka JavaScript

Od czasu rewolucji wywołanej przez aplikację GMail język JavaScript rozwinął się w ogromnym stopniu. Nowe wojny związane z przeglądarkami, w których do konfrontacji stają aplikacje Internet Explorer i Edge oraz Chrome i Firefox, doprowadziły do powstania wielu bardzo szybkich interpreterów języka JavaScript. Opracowane zostały zupełnie nowe techniki optymalizacji. Całkiem często można się spotkać z kompilowaniem kodu JavaScript do postaci kodu maszynowego w celu zapewnienia mu większej wydajności. Ponieważ jednak wzrosła szybkość kodu JavaScript, zwiększyła się też złożoność aplikacji tworzonych z jego wykorzystaniem.

JavaScript nie jest już również wyłącznie językiem służącym do wprowadzania zmian w przeglądarce. Wyodrębniono silnik tego języka, na którym bazuje popularna przeglądarka Chrome, i obecnie jest on „sercem" kilku interesujących projektów, takich jak środowisko Node.js. Na początku zapewniało ono cechujący się wysokim stopniem asynchroniczności sposób tworzenia aplikacji serwerowych. Z czasem znacznie się rozwinęło i obecnie wspierane jest przez bardzo aktywną społeczność. Z wykorzystaniem środowiska Node.js zaprojektowano bardzo różnorodne aplikacje, od narzędzi do budowania po edytory. W ostatnim czasie udostępniono publicznie również kod źródłowy silnika języka JavaScript dla przeglądarki Microsoft Edge o nazwie ChakraCore, który może zostać osadzony w środowisku Node.js jako alternatywa wobec silnika V8 firmy Google. SpiderMonkey, czyli odpowiednik silnika z przeglądarki Firefox, także oferuje publicznie kod źródłowy, dzięki czemu zyskuje na popularności w przypadku większej liczby narzędzi.

Język JavaScript może nawet zostać zastosowany do sterowania mikrokontrolerami. Johnny-Five to środowisko programistyczne przeznaczone do bardzo popularnej platformy Arduino. Środowisko to zapewnia o wiele prostszy sposób programowania urządzeń niż tradycyjne języki niskopoziomowe używane do pisania oprogramowania dla tych urządzeń. Wykorzystanie języka JavaScript i platformy Arduino oferuje dostęp do wielu możliwości, od tworzenia robotów po czujniki stosowane na co dzień.

Wszystkie podstawowe platformy smartfonów (iOS, Android i Windows Phone) oferują opcję budowania aplikacji za pomocą języka JavaScript. W przypadku tabletów wygląda to bardzo podobnie. Obsługują one programowanie z wykorzystaniem języka JavaScript. Nawet najnowsza wersja systemu Windows zapewnia mechanizm do tworzenia aplikacji przy użyciu tego języka. Poniższa ilustracja prezentuje niektóre zastosowania możliwe w przypadku języka JavaScript.



JavaScript staje się jednym z najważniejszych języków świata. Choć wiadomo, że bardzo trudne jest uzyskanie statystyk wykorzystania języków, w każdym niezależnym źródle, w którym podjęto próbę opracowania rankingu, JavaScript plasuje się w pierwszej dziesiątce.

Zestawienie językowe	Pozycja języka JavaScript
Langpop.com	4
Statisticbrain.com	4
Codeval.com	6
TIOBE	8

Jeszcze bardziej interesujące jest to, że w większości tych rankingów zasugerowano, iż zwiększa się poziom wykorzystania języka JavaScript.

Krótko mówiąc, wynika z tego, że w ciągu kilku następnych lat język JavaScript będzie należeć do ścisłej czołówki. Coraz więcej aplikacji pisanych jest w tym języku, a ponadto pełni on funkcję języka roboczego w przypadku dowolnego rodzaju procesu projektowania aplikacji internetowych. Jeff Atwood, twórca popularnej witryny internetowej Overflow, sformułował tak zwane prawo Atwooda dotyczące szerokiej skali adaptacji języka JavaScript:

> *„Dowolna aplikacja, która może zostać napisana w języku JavaScript, ostatecznie zostanie za jego pomocą utworzona" — prawo Atwooda, Jeff Atwood*

Powyższa obserwacja została wielokrotnie potwierdzona jako prawdziwa. Obecnie wiele kompilatorów, arkuszy kalkulacyjnych i edytorów tekstu, które znasz, zostało napisanych z wykorzystaniem języka JavaScript.

Wraz ze zwiększaniem się złożoności aplikacji opartych na kodzie JavaScript programista może napotkać wiele tych samych problemów, które pojawiały się w tradycyjnych językach programowania. Jak można utworzyć taką aplikację, która będzie przystosowana do zmian?

W ten oto sposób pojawia się potrzeba właściwego projektowania aplikacji. Nie można już po prostu umieścić porcji kodu JavaScript w pliku i mieć nadziei, że zadziała poprawnie. Nie można też szukać ratunku w bibliotekach, takich jak jQuery. Biblioteki mogą jedynie zapewnić dodatkową funkcjonalność, a ponadto nie wpłyną na strukturę aplikacji. Przynajmniej trochę uwagi trzeba obecnie poświęcić sposobowi budowania aplikacji, tak aby mogły być rozszerzane i adaptowane. Otaczający nas świat nieustannie się zmienia. Każda aplikacja, która nie ma możliwości dostosowania do zmieniającego się świata, prawdopodobnie zostanie zapomniana. Wzorce projektowe zapewniają pewne wytyczne podczas tworzenia elastycznych aplikacji, które mogą się zmieniać wraz ze zmieniającymi się wymaganiami biznesowymi.

Czym jest wzorzec projektowy?

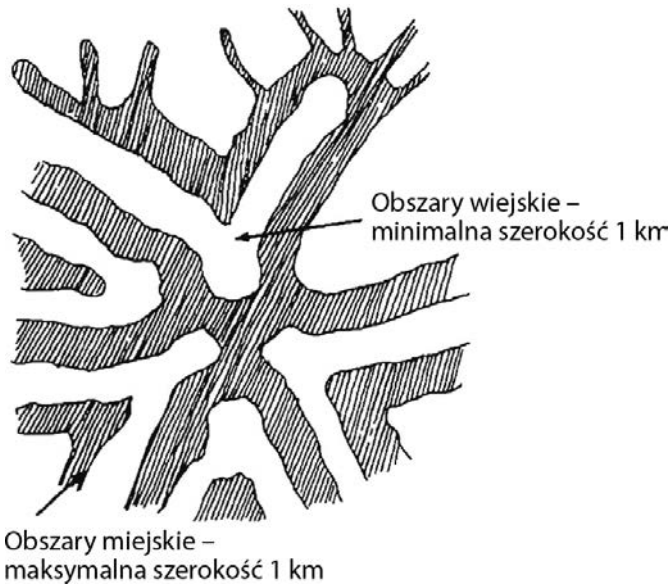
Pomysły mogą przeważnie zostać wykorzystane tylko w jednym miejscu. Użycie masła orzechowego jest znakomitym pomysłem podczas gotowania, lecz nie w przypadku szycia. Sporadycznie możliwe jest jednak określenie możliwości zastosowania wspaniałego pomysłu w miejscu innym niż pierwotnie ustalone. Właśnie z tym związane są wzorce projektowe.

Christopher Alexander, Sara Ishikawa i Murray Silverstein napisali w 1977 r. nowatorską książkę *A Pattern Language: Towns, Buildings, Construction* poświęconą temu, co zostało przez nich nazwane wzorcami projektowymi w planowaniu urbanistycznym.

W książce opisano język służący do omawiania podobieństw projektu. Wzorzec został w niej przedstawiony w następujący sposób:

> *„Elementy tego języka są jednostkami nazywanymi wzorcami. Każdy wzorzec opisuje problem występujący nieustannie w naszym środowisku, a następnie charakteryzuje rdzeń rozwiązania tego problemu w taki sposób, że może ono zostać wykorzystane ponad milion razy bez konieczności realizowania tego dwa razy w ten sam sposób" — Christopher Alexander*

Te wzorce projektowe miały na przykład za zadanie określenie sposobu planowania miast w celu zapewnienia połączenia miejskich i wiejskich elementów życia lub sposobu budowania dróg w pętlach pozwalających na uspokojenie ruchu w dzielnicach mieszkalnych. Zostało to zaprezentowane na poniższym rysunku pochodzącym z książki.



Nawet osoby niezainteresowane zbytnio planowaniem urbanistycznym znajdą w tej książce kilka fascynujących pomysłów dotyczących tego, jak zorganizować otaczający nas świat w celu promowania idei zdrowego społeczeństwa.

Wykorzystując pracę Christophera Alexandera oraz innych autorów jako źródło inspiracji, Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides napisali książkę zatytułowaną *Design Patterns: Elements of Reusable Object-Oriented Software*. Gdy książka wywiera bardzo duży wpływ na program nauczania informatyki, często nadawana jest jej pieszczotliwa nazwa. Na przykład większość absolwentów informatyki w krajach anglojęzycznych będzie wiedzieć, o jakiej książce mowa, gdy poda się nazwę *The Dragon Book* (*Principles of Compiler Design*, 1986 r.). W branży oprogramowania dla przedsiębiorstw nazwa *The Blue Book* jest powszechnie kojarzona z książką Erica Evana poświęconą projektowaniu DDD (*Domain-Driven Design*). Książka o wzorcach projektowych okazała się tak ważna, że powszechnie określa się ją mianem książki GoF lub *Gang of Four*, ponieważ miała czterech autorów.

W książce tej opisano 23 wzorce stosowane w projektowaniu obiektowym. Wzorce zostały podzielone na następujące trzy podstawowe grupy:

- **Wzorce kreacyjne**. Wzorce te opisują kilka sposobów tworzenia obiektów i zarządzania ich cyklami życia.
- **Wzorce operacyjne**. Wzorce te opisują sposób wzajemnej interakcji obiektów.
- **Wzorce strukturalne**. Wzorce te opisują gamę różnych sposobów dodawania funkcjonalności do istniejących obiektów.

Celem wzorców projektowych nie jest instruowanie co do sposobu tworzenia oprogramowania, lecz raczej zapewnianie wytycznych dotyczących możliwych metod rozwiązywania typowych problemów. Na przykład wiele aplikacji musi oferować pewnego rodzaju funkcję cofania zmian. Problem ten jest typowy dla edytorów tekstu, programów do rysowania, a nawet klientów poczty elektronicznej. Był rozwiązywany wielokrotnie, zanim uznano, że wspaniałym pomysłem byłoby zapewnienie wspólnego rozwiązania. Wzorzec polecenia oferuje właśnie takie rozwiązanie. Sugeruje on śledzenie wszystkich działań wykonywanych w aplikacji jako instancji polecenia. Polecenie to będzie dysponować działaniem przejścia do przodu i do tyłu. Każdorazowo przetwarzane polecenie umieszczane jest w kolejce. Gdy pojawi się potrzeba cofnięcia polecenia, sprowadza się to do usunięcia z kolejki poleceń najwyżej znajdującej się pozycji i wykonania dla niej operacji cofania.

Wzorce projektowe zapewniają pewne wskazówki dotyczące sposobu rozwiązania typowych problemów, takich jak problem cofania zmian. Wzorce zostały uzyskane po wykonaniu setek iteracji rozwiązujących ten sam problem. Wzorzec może nie być ściśle poprawnym rozwiązaniem danego problemu, ale powinien przynajmniej zaoferować pewne wytyczne pozwalające na łatwiejszą implementację rozwiązania.

> Znajomy konsultant opowiedział mi kiedyś historię o początku realizacji zlecenia w nowej firmie. Kierownik powiedział jemu i innym osobom, że nie sądził, iż będzie wymagane tak wiele pracy z zespołem, ponieważ na samym początku zaopatrzył projektantów w książkę GoF o wzorcach projektowych, a oni zaimplementowali każdy wzorzec bez wyjątku. Dowiedziawszy się o tym, mój znajomy był szczęśliwy, ponieważ był opłacany za liczbę przepracowanych godzin. Niewłaściwe zastosowanie wzorców projektowych przez klienta pozwoliło mu opłacić znaczną część studiów swojego dziecka.

Od czasów książki GoF pojawiało się coraz więcej literatury wyszczególniającej i opisującej wzorce projektowe. Dostępne są książki poświęcone wzorcom projektowym, powiązanym z konkretnymi dziedzinami, a także książki traktujące o wzorcach przeznaczonych dla dużych systemów klasy korporacyjnej. Kategoria serwisu Wikipedia stworzona dla wzorców projektowych oprogramowania zawiera 130 wpisów dla różnych wzorców projektowych. Przychylałbym się jednak do tego, że wiele z tych wpisów nie dotyczy prawdziwych wzorców projektowych, lecz raczej paradygmatów programowania.

Wzorce projektowe to przeważnie proste konstrukcje, które nie wymagają złożonej obsługi bibliotek. Choć w przypadku większości języków istnieją biblioteki wzorców, nie ma potrzeby wydawania mnóstwa pieniędzy na ich zakup. Implementuj wzorce, gdy uznasz to za konieczne.

Dysponowanie kosztowną biblioteką, która nadwerężyła budżet, może zachęcać do stosowania wzorców na ślepo tylko po to, aby uzasadnić wydanie pieniędzy. Jeśli nawet rozważasz zakup biblioteki, nie jest mi znana żadna biblioteka dla języka JavaScript, której jedynym celem jest zapewnianie obsługi wzorców. Oczywiście serwis GitHub jest pełen interesujących projektów powiązanych z językiem JavaScript, dlatego równie dobrze może istnieć tam biblioteka, o której nie wiem.

Są osoby, które sugerują, że wzorce projektowe powinny być wynikiem działań. Oznacza to, że już samo utworzenie oprogramowania w inteligentny sposób umożliwia dostrzeżenie wzorców wynikających z implementacji. Uważam, że może to być trafne stwierdzenie, w którym zignorowano jednak rzeczywisty koszt uzyskania tych implementacji metodą prób i błędów. Osoby znające wzorce projektowe ze znacznie większym prawdopodobieństwem na samym początku dostrzegą wzorzec wynikający z implementacji. Uczenie początkujących programistów o wzorcach to bardzo przydatne ćwiczenie. Posiadanie od początku informacji o tym, jaki wzorzec lub wzorce mogą zostać zastosowane, umożliwia pójście na skróty. Pełne rozwiązanie może zostać uzyskane wcześniej i po wykonaniu mniejszej liczby niewłaściwych czynności.

Antywzorce

Jeśli w dobrym projekcie oprogramowania można znaleźć typowe wzorce, to czy istnieją też wzorce, które mogą zostać znalezione w złym projekcie oprogramowania? Oczywiście! Można wymienić dowolną liczbę sposobów niewłaściwego realizowania działań, ale większość z nich została już wcześniej wykonana. Potrzeba prawdziwej kreatywności, żeby coś popsuć w nieznany dotąd sposób.

Mankamentem jest tutaj to, że bardzo trudno zapamiętać wszystkie sposoby niewłaściwego realizowania działań, które miały miejsce w ciągu wielu lat. Na zakończenie wielu dużych projektów członkowie zespołu zbierają się i przygotowują dokument z *wyciągniętymi wnioskami*. Dokument zawiera listę rzeczy, które mogły zostać lepiej zrealizowane w projekcie. W dokumencie mogą być nawet przedstawione pewne sugestie dotyczące tego, jak w przyszłości można uniknąć napotkanych problemów. Niefortunne jest to, że taki dokument jest sporządzany tylko na końcu projektu. Od tego momentu wielu kluczowych uczestników projektu będzie już niedostępnych, a pozostałe osoby muszą starać się zapamiętać wnioski wyciągnięte z wcześniejszych etapów projektu, które mogły być realizowane wiele lat temu. Znacznie lepszym rozwiązaniem jest tworzenie dokumentu w trakcie realizacji projektu.

Po ukończeniu dokument jest przechowywany, aby był gotowy do wykorzystania przy następnym projekcie. Taka jest przynajmniej teoria. W większości sytuacji dokument jest przechowywany i nigdy nie zostaje ponownie użyty. Trudno sformułować wnioski, które mają uniwersalny charakter. Wyciągnięte wnioski są zwykle przydatne tylko w przypadku bieżącego projektu lub dokładnie takiego samego projektu, co prawie nigdy nie ma miejsca.

Po przejrzeniu kilku takich dokumentów z różnych projektów wzorce zaczynają być jednak widoczne. Przyjmowano przy tym takie podejście, że w efekcie William Brown, Raphael Malveau, Skip McCormick i Tom Mowbray, których razem określono mianem Upstart Gang of Four w nawiązaniu do wcześniej powstałej nazwy Gang of Four, napisali pierwszą książkę poświęconą antywzorcom. W książce zatytułowanej *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* opisano antywzorce nie tylko w odniesieniu do problemów obecnych w kodzie, ale też powiązanego z nim procesu zarządzania.

Zaprezentowane wzorce to między innymi humorystycznie nazwane wzorce **The Blob** (klucha) i **Lava Flow** (potok lawy). The Blob, znany również jako „obiekt Boga", to wzorzec, w przypadku którego jeden obiekt rozrasta się w celu przejęcia odpowiedzialności za ogromne ilości elementów logicznych aplikacji. Lava Flow to wzorzec pojawiający się wraz z dojrzewaniem projektu, gdy nikt nie wie, czy dany kod jest nadal używany. Projektanci z obawą podchodzą do usuwania kodu, ponieważ może być gdzieś wykorzystywany lub może ponownie okazać się przydatny. W książce opisano wiele innych wzorców, które są warte poznania. Tak jak w przypadku wzorców, antywzorce są wynikiem pisania kodu, lecz w tym przypadku jest to kod, nad którym utracono kontrolę.

W książce nie będą prezentowane antywzorce języka JavaScript, ale warto pamiętać, że jednym z antywzorców jest nadmierne stosowanie wzorców projektowych.

Podsumowanie

Wzorce projektowe mają bogatą i interesującą historię. Na początku pełniły one funkcję narzędzi ułatwiających opisywanie sposobu budowania struktur, które umożliwiają ludziom współpracę. Wzorce rozwinęły się do poziomu pozwalającego na zastosowanie ich w wielu dziedzinach.

Minęła już dekada, odkąd pojawiła się nowatorska praca poświęcona zastosowaniu wzorców projektowych podczas programowania. Od tego czasu opracowano ogromną liczbę nowych wzorców. Niektóre z nich to wzorce ogólnego przeznaczenia, takie jak te opisane w książce GoF, jednak wiele z nich stanowią bardzo specyficzne wzorce, które zaprojektowano pod kątem użycia w wąskiej dziedzinie.

Język JavaScript również ma interesującą historię i naprawdę dojrzewa. Jeśli weźmiemy pod uwagę rozwój oprogramowania serwerowego opartego na kodzie JavaScript i rozpowszechnienie dużych aplikacji JavaScript, pojawia się potrzeba większej staranności przy tworzeniu takich aplikacji. Do rzadkości należy właściwe wykorzystanie wzorców w większości nowoczesnych kodów JavaScript.

Bazowanie na wiedzy zapewnianej przez wzorce projektowe przy tworzeniu nowoczesnych wzorców języka JavaScript pozwala uzyskać to, co najlepsze w obu tych „światach". Isaac Newton jest autorem następującego słynnego stwierdzenia:

> *„Jeśli widzę dalej, to tylko dlatego, że stoję na ramionach olbrzymów".*

Wzorce oferują nam łatwo dostępne „ramiona", na których możemy stanąć.

W następnym rozdziale przyjrzymy się kilku technikom budowania struktury w kodzie JavaScript. System dziedziczenia w języku JavaScript różni się od obecnego w większości innych języków obiektowych, co stwarza nam zarówno możliwości, jak i ograniczenia. Dowiemy się, jak tworzyć klasy i moduły w świecie języka JavaScript.

Skorowidz

G

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ż

JavaScript i wzorce projektowe

Programowanie dla zaawansowanych

Wydanie II

JavaScript przez lata był wykorzystywany głównie jako technika uzupełniania podstawowej funkcjonalności stron internetowych. Dziś to się zmienia: JavaScript należy do najpopularniejszych języków i jest wykorzystywany na wiele sposobów. Wzorce projektowe to jedna z ciekawszych możliwości: dzięki nim programista bierze pod uwagę sprawdzone rozwiązania.

Niniejsza książka jest przeznaczona dla osób używających JavaScriptu, które chcą nauczyć się programowania obiektowego w tym języku, a także dobrze poznać standard ECMAScript 2015. Przedstawiono tu wzorce kreacyjne, strukturalne i operacyjne oraz metody ich stosowania. Przeanalizowano wzorce widoku modelu i wzorce do budowy aplikacji internetowych. Obszernie omówiono mikrousługi, a także wzorce do testowania kodu za pomocą atrap obiektów i środowisk atrap obiektów oraz techniki monkey patching. Opisano też kilka wzorców zaawansowanych, w tym wzorzec wprowadzania zależności i przetwarzania końcowego w czasie rzeczywistym.

Wzorce projektowe
— niezbędne w przyborniku programisty!

W książce znajdziesz:

- czym są wzorce projektowe i jak należy organizować kod
- poszczególne grupy wzorców projektowych i ich zastosowanie
- programowanie funkcyjne i reaktywne
- programowanie aspektowe
- inne narzędzia zgodne ze standardem ECMAScript 2015/2016

Simon Timms jest głównym projektantem oprogramowania w firmie Clear-Measure w Austin w Teksasie. Jest ekspertem w dziedzinie technologii serwerowych .NET, interesuje się wizualizacjami i przetwarzaniem w chmurze. Angażuje się w rozwijanie metodyki DevOps. Często wypowiada się na jej temat w ramach grupy Calgary.Net, której jest przewodniczącym.

[PACKT] open source PUBLISHING community experience distilled

Helion

księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-3194-5
9 788328 331945
cena: 54,90 zł