La plu belle des ruses du diable est de vous persuader qu'il n'existe pas.

Charles Baudelaire
Le Spleen de Paris, 1869

O sztuce programowania

Termin „sztuka programowania" jest notorycznie nadużywany w znaczeniu komercyjnym. Autorzy i wydawnictwa nadają książkom tytuły „Sztuka programowania w języku XYZ", aby zwiększyć ich atrakcyjność i sprzedaż. Nie mają one wiele wspólnego z rzeczywistą sztuką programowania. Owa „sztuka programowania" opiewana w tych publikacjach to po prostu umiejętność kodowania. I tyle.

Janusz Zalewski
ukończył studia na Wydziale Elektroniki, doktorat obronił na Wydziale Elektrycznym Politechniki Warszawskiej. Po studiach pracował w Instytutach Badań Jądrowych w Warszawie i Świerku, komputeryzując eksperymenty z dziedziny fizyki i chemii jądrowej, a w 1989 r. wyjechał do USA, gdzie pracował w laboratoriach jądrowych oraz uczył informatyki na uczelniach w Teksasie i na Florydzie.
Jest emerytowanym profesorem Florida Gulf Coast University i profesorem informatyki na Państwowej Uczelni Zawodowej im. Ignacego Mościckiego w Ciechanowie.
Prywatnie zajmuje się tłumaczeniem na język polski literatury polskich Amerykanów oraz analizą twórczości literackiej amerykańskich bitników.

Jest to bardzo praktyczna wiedza programistyczna, ale oznaczająca niewiele więcej niż sprawne posługiwanie się konstrukcjami danego języka dla osiągnięcia założonych efektów do realizacji przez komputer. To jest wiedza ważna i niezbędna, ale prawie wyłącznie rzemieślnicza. W programowaniu musi być jakościowo o wiele więcej, by można je nazwać sztuką. Ekstremalny punkt widzenia zakłada, że do programowania pojmowanego głębiej, jako sztuka właśnie, nie jest specjalnie potrzebny ani język programowania, ani komputer.

Proweniencja terminu

Oryginalne dywagacje dotyczące sztuki programowania datują się na koniec lat 60. i początek lat 70. XX w. Trudno

ustalić, kto użył tego określenia po raz pierwszy, ale za protagonistę sztuki programowania powszechnie uchodzi Donald Knuth, autor kilkutomowego dzieła pod takim właśnie tytułem[1]. W wykładzie wygłoszonym z okazji przyznania mu Nagrody Turinga Knuth stwierdza[2]: *Gdy mówię o programowaniu komputerów jak o sztuce, myślę o nim przede wszystkim jak o formie sztuki w sensie estetycznym. Głównym celem mojej działalności edukacyjnej i autorskiej jest pomagać ludziom nauczyć się, jak pisać piękne programy.* I kończy ten wykład stwierdzeniem: *Zobaczyliśmy, że programowanie komputerów jest sztuką, ponieważ stosuje zgromadzoną wiedzę do świata, ponieważ wymaga umiejętności i inwencji, a szczególnie ponieważ wytwarza przedmioty piękna.*

Inni autorzy, współcześni Knuthowi, też mieli wiele do powiedzenia na temat sztuki programowania. Jednym z nich był holenderski pionier informatyki Edsger Dijkstra, który twierdził: *sztuka programowania jest sztuką organizowania złożoności, sztuką panowania nad wielością i unikania jej totalnego chaosu tak skutecznie, jak tylko możliwe*[3]. W słynnym wykładzie na seminarium w Newcastle (1970 r.) tak ocenia sztukę programowania: *analiza zadania programistycznego prowadzi do wniosku, że programowanie jest gigantycznym wyzwaniem intelektualnym, nie mającym precedensu w historii ludzkości*[4]. W podsumowującym te rozważania raporcie[5] Dijkstra wypowiada kluczowe stwierdzenie o roli programisty: *nie jest niczym niezwykłym – aczkolwiek błędnym – stwierdzenie, że zadaniem programisty jest wytworzenie programu. Bardziej owocne wydaje się opisanie działalności programisty jako projektowanie klasy obliczeń raczej niż tworzenie programu.*

> *Podobne opinie wypowiadał prof. Władysław M. Turski. Również w Newcastle, aczkolwiek kilka lat później, stwierdził*[6]*: przede wszystkim musimy uczyć myślenia problemowego. Musimy nalegać, aby nasi programiści myśleli w kategoriach problemów, które rozwiązują, a nie w kategoriach technik programowania, które stosują.*

Pisząc o sztuce programowania, nie sposób pominąć rozważań o pięknie programu. Interesująco na ten temat wypowiadają się autorzy pracy zbiorowej *Beautiful Code*, wydanej również po polsku[7]. Autor rozdziału „Najpiękniejszy kod, którego nigdy nie napisałem", Jon Bentley, znany ze słynnej książki *Perełki oprogramowania* i z autorstwa algorytmu sortowania Quicksort, przedstawia swoją maksymę piękności kodu, cytując Antoine'a Saint-Exupéry'ego: *projektant może uznać, że osiągnął perfekcję nie wtedy, kiedy nie pozostało już nic do dodania, ale wtedy, gdy nie można już nic odjąć.*

W rozdziale zatytułowanym „Długoterminowe korzyści pięknego projektowania", absolwent AGH, UJ i Caltechu, założyciel firmy Parasoft w Kalifornii, nieżyjący już Adam Kolawa, tak przedstawia swoje pojmowanie piękna kodu: *moja idea pięknego kodu bierze się z rozumienia, że ostatecznym celem kodu jest [poprawne] działanie. Inaczej mówiąc, kod powinien dokładnie i efektywnie wykonywać zadanie, do którego został stworzony, w taki sposób, że nie ma żadnych dwuznaczności co do tego, jak się zachowuje. […] W sumie myślę, że piękny kod musi być krótki, jasny, oszczędny i napisany z uwzględnieniem realiów. Jednak prawdziwym testem piękności – dla kodu jak i dla sztuki – będzie fakt, czy wytrzyma próbę czasu.*

W tej samej publikacji podobnego co Kolawa zdania jest guru języka C, Brian Kernighan, aczkolwiek w odniesieniu do jednego tylko aspektu programowania: *[…] rekurencja to strzał w dziesiątkę. Ta fundamentalna technika programistyczna prawie zawsze prowadzi do powstania mniejszego, przejrzystego i bardziej eleganckiego kodu niż jego odpowiednik napisany przy zastosowaniu pętli.*

Klawiatura jak pędzel

Uciekając się do analogii z malarstwem, można powiedzieć, że klawiatura i język programowania są dla programisty mniej więcej tym samym co pędzel i farba dla malarza. Artyście potrzebne jest tworzywo, a więc język programowania lub farba, i narzędzie, a więc klawiatura lub pędzel. Malarz potrzebuje płótna albo papieru lub jakiegoś innego nośnika,

1 D.E. Knuth, *The Art of Computer Programming*. Vol. 1. Addison-Wesley, 1968 (polskie wydanie: *Sztuka programowania*, WNT, Warszawa 2002).

2 D.E. Knuth, Computer Programming as an Art. *Communications of the ACM*, vol. 17, n. 12, pp. 667-673, 1974. URL: https://dl.acm.org/doi/pdf/10.1145/361604.361612

3 E.W. Dijkstra, Notes on Structured Programming, Raport EWD249, 1970. URL: https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

4 E.W. Dijkstra, The Art of Programming. Newcastle International Seminars on the Teaching of Computing Science, 1970. URL: http://homepages.cs.ncl.ac.uk/brian.randell/Seminars/136.pdf

5 E.W. Dijkstra, A Short Introduction to the Art of Programming. Raport EWD316, 1971. URL: https://lass.cs.umass.edu/~shenoy/courses/summer04/readings/Dijkstra_program_guide.pdf

6 W.M. Turski, A View of Software Problems. Newcastle International Seminars on the Teaching of Computing Science, 1976. URL: http://homepages.cs.ncl.ac.uk/brian.randell/Seminars/362.pdf

7 A. Oram, G. Wilson (red.), *Piękny kod. Tajemnice mistrzów programowania*, Helion, Warszawa 2017.

stanowiącego swego rodzaju platformę dla twórcy. Programista ma w tym celu komputer. Wartość dzieła malarza zależy od biegłości posługiwania się pędzlem i farbą. Podobnie wartość tego, co zostanie stworzone przez programistę zależy od umiejętności posługiwania się klawiaturą i językiem. Artysta malarz powinien mieć umiejętność „zakodowania" w swoim dziele przesłania do odbiorcy, programista zaś jako artysta musi mieć umiejętność „odmalowania" w swoim programie cech nadających mu pożądaną interpretację.

W obu przypadkach w grę wchodzi wytworzenie produktu nacechowanego przesłaniem adresowanym do odbiorcy. Potrzebne są do tego – przede wszystkim – pewne umiejętności konstrukcyjne. U malarza rzemiosło artystyczne pozwala mu na wykreowanie obrazu, a u programisty jest to umiejętność kodowania, dzięki czemu może on stworzyć program (tak jak artysta – wykonać dzieło). Jednakże, ażeby ten produkt malarza lub programisty zawierał cechy piękna, potrzebne są nie tylko umiejętności rzemieślnicze, lecz też umiejętności twórcze. Nie nam rozstrzygać, co wchodzi w skład umiejętności twórczych malarza. Możemy natomiast rozważyć, jakie cechy można uznać za składniki niezbędne do określenia umiejętności twórczych programisty.

Wielu z nas zaczęłoby od tego, jakie cechy powinien mieć produkt programisty, a więc program, aby można go było porównać do dzieła sztuki. Idąc śladami cytowanych autorów, można powiedzieć, że do tych cech należą m.in.: zwięzłość (Bentley), przejrzystość i efektywność (Kolawa), a można je osiągnąć przez ujarzmienie złożoności i wielości (Dijkstra), co gwarantuje elegancką całość (Kernighan). Takie rozumowanie ma sens, bo prowadzi do stworzenia pięknego kodu, jednak skupiamy się wyłącznie na produkcie, a więc na wytworze artysty. Trzeba jednak wziąć pod uwagę, że w sztuce, nie pomijając sztuki programowania, ważne są także czynności prowadzące do wytworzenia dzieła, a więc powinno się je również uwzględnić w opisie, kładąc nacisk na sam proces tworzenia.

To jest istotny element, bo artysta tworząc dzieło sztuki, ma wizję finalnego produktu i temu podporządkowuje proces twórczy. Podobnie postępuje twórca programu komputerowego, ale jego wizja ma charakter bardziej utylitarny, a nawet – można by rzec – techniczny, tym niemniej ciągle artystyczny. Właśnie to miał na myśli cytowany Edsger Dijkstra, pisząc o zadaniu programisty jako *projektowaniu klasy obliczeń*. To miał też na myśli Władysław Turski, naciskając, aby uczyć programistów myślenia *w kategoriach problemów, które rozwiązują, a nie technik programowania, których używają.*

Algorytm – ucieleśnienie wizji programisty

Te wszystkie rozważania prowadzą nas do pojęcia *algorytm*. Programista, myśląc nad problemem zadanym mu do rozwiązania, tworzy jego algorytm, a więc recepturę na utworzenie rozwiązania, co w przyszłości przyjmie postać programu komputerowego. Algorytmem malarza jest jego wizja, która go prowadzi przez proces tworzenia dzieła.

> *W procesie twórczym jest pełna analogia między wizją a algorytmem – wartość artystyczna dzieła zawiera się w sferze wizji autora.*

Tak więc sztukę programowania należy rozumieć jako sztukę rozwiązywania problemów, czyli tworzenia algorytmów. Ważnym elementem tego procesu jest optymalizacja, a więc znajdowanie algorytmów najlepszych względem pewnego, dobrze określonego kryterium jakości. Na ogół tym kryterium jest skrócenie czasu obliczeń (mierzone na przykład liczbą kroków prowadzących do uzyskania rozwiązania) lub minimalizacja zajętości pamięci (co sprowadza się do wielkości obszaru zarezerwowanego na dane, a więc liczby zmiennych i długości słowa). Często chodzi o dokładność obliczeń, co wymaga dużej precyzji w reprezentacji danych. Bardziej współczesnym kryterium optymalizacji może być też zmniejszenie przewidywanego zużycia energii. Jon Bentley fascynująco relacjonuje w cytowanej pracy swoje perypetie z udoskonalaniem algorytmu Quicksort. Sprawy szlifowania algorytmów świetnie opisuje David Harel w „Algorytmice"[8].

Z obserwacji i z praktyki wiem, że 90% programistów ma dobre pojęcie o kodowaniu, ale znikome lub żadne o projektowaniu rozwiązania, a więc o tworzeniu algorytmu, nie mówiąc o wiedzy z dziedziny zastosowań, której nabycie wymaga ścisłego współdziałania ze zleceniodawcą, nie tylko z działem marketingu własnej firmy. I tu dopiero jawi się mądrość motta z Baudelaire'a, że *najlepszą sztuczką diabła jest przekonać nas, iż nie istnieje*. Wiedzieć, że czegoś nie wiemy – tego właśnie brakuje programistom, aby programowanie stało się dla nich sztuką.

8 D. Harel, *Rzecz o istocie informatyki. Algorytmika*. WNT, Warszawa 2001.