Semina □ Nr 1
Scientiarum 2002

Robert Janusz

Dlaczego informatyka poszukuje filozofii?

W tym artykule przedstawione zostaną informatyczne problemy związane ze złożonością. Informatyka, w sobie właściwy sposób, otwiera ciekawe kwestie filozoficzne. Jej rozwój domaga się też filozoficznej wizji zagadnień, które stara się ona rozwiązać. Dotyczy to szczególnie złożonych systemów informatycznych.

1. Iluzja komputerowej prostoty

Dzięki komputerom wiele zagadnień wydaje się bardzo prostych. Iluzji tej uległ Thomas J. Watson, prezes IBM, gdy powiedział w 1948 roku, że «świat potrzebuje najwyżej tuzina komputerów» do zaspokojenia swoich rachunkowych potrzeb. Na przekór temu komputerowa rewolucja dokonała się na przestrzeni kilku pokoleń w iście inflacyjnym tempie. Gdyby porównać rozwój produkcji samochodów do postępów osiąganych w dziedzinie komputerów, to w 1988 roku cena Rolls–Royce powinna spaść poniżej dolara a ok. 15g benzyny wystarczyłoby na 20 lat jazdy samochodem[1].

Zastanowimy się, skąd bierze się przeświadczenie o „komputerowej prostocie". Rozpoczniemy od refleksji nad fizyczną warstwą komputerów, a następnie zwrócimy uwagę na ich warstwę logiczną, programową.

Dzisiejsze komputery wykorzystują binarny system liczenia. Wydawać by się mogło, że nie ma nic prostszego, niż dwójkowe stany urządzeń fizycznych: płynie prąd lub nie, pewien obszar dysku jest namagnesowany lub nie, itd. Tymczasem stany te, z fizycznego punktu widzenia, wcale nie są łatwe do przetwarzania. Ostre przejście pomiędzy sygnałem „1" a „0" bardzo łatwo ulega rozmyciu na „naturalnych" impedancjach obwodów elektronicznych. Utrzymanie elektroniki w stanie binarnym wymaga bardzo zaawansowanej wiedzy teoretycznej i bardzo precyzyjnych

[1]Obliczenia pochodzą od D. O. Arnolda (zob.: Tadeusiewicz, *Wstęp do informatyki*, Kraków 1993, s. 194). Oczywiście przedstawione tutaj analogie nie biorą pod uwagę tego, że komputery, w przeciwieństwie do samochodów, rozwijają się głównie w warstwie logicznej.

technologii. Rodzi się zatem pytanie: dlaczego ludzie z takim wysiłkiem starają się opanować „ostre" sygnały binarne a nie wykorzystują w komputerach o wiele prostszych do manipulowania analogowych sygnałów fizycznych?

Odpowiedź na to pytanie nie jest trudna. Dzieje się tak dlatego, że sygnały binarne charakteryzują się wyższą hierarchicznie, nie sprzętową logiczną prostotą, emergentnie otwartą na dalsze przetwarzanie i manipulowanie, której to cechy nie posiadają fizyczne sygnały analogowe. Technologiczny wysiłek jest opłacalny, aby podporządkować się teoretycznej prostocie.

Podobne zjawisko ma miejsce także w strukturze fizycznej rozkazów jednostki centralnej komputera. Coraz doskonalsze i bardziej złożone procesory wzbogacano o nowe możliwości przez dokładanie kolejnych instrukcji. Spowodowało to, że wiele procesorów dysponuje gamą kilkuset elementarnych rozkazów maszynowych. Tymczasem w programowaniu komputerów ponad 90 procent operacji logicznych wykorzystuje jedynie ok. 10 procent dostępnych instrukcji procesora. Starano się więc maksymalnie uprościć listę rozkazów maszynowych i dzięki temu można było także zwielokrotnić szybkość jednostki centralnej[2].

Nie zawsze jednak fizyczne urządzenia wypierane są przez urządzenia lepsze. Wynaleziona w XIX w. klawiatura Christophera Scholesa rozwiązywała mechaniczny problem zacinania się czcionek przy szybkim pisaniu. Problem ten jednakże całkowicie stracił na znaczeniu w przypadku klawiatury komputera, która nie posiada mechanicznego przeniesienia na czcionki. Jednakże stare rozwiązane nadal dominuje nad lepszym logicznie układem klawiszy Augusta Dvoraka[3].

Można jednak ogólnie powiedzieć, że układy cyfrowe działające w oparciu o logikę binarną ulegały pewnym „ideowym uproszczeniom" wymuszonym przez operującą na nich warstwę logiczną. Pokonywano przy tym złożone trudności związane z utrzymaniem fizycznych stanów w binarnej formie. Przyjrzymy się teraz podobnej iluzji prostoty mającej miejsce w programowaniu.

Algebra Boole'a, która była wielkim sukcesem matematyki XIX wieku, przyczyniła się do preferowania fizycznych układów binarnych. Dzięki pomysłowi von Neumanna polegającemu na kodowaniu rozkazów w tej samej pamięci, w której przechowywano dane, możliwe stało się znaczne

[2]Komputery te określa się kryptonimem RISC = Reduced Instruction Set Computer; por.: Tadeusiewicz, op. cit., s. 34n.

[3]Zob.: Tadeusiewicz, op. cit., s. 57.

uproszczenie logiczne komputerowego sprzętu. Ludzie nauczyli się programować algorytmy nie tylko w języku maszynowym, ale pojawiły się pierwsze translatory. W programowaniu tym, podobnie jak w maszynie Turinga, operowano elementarnymi rozkazami, wśród których przeważała instrukcja skoku. Nie stanowiło to wielkiego problemu, gdyż pierwsze komputery nie wykraczały poza problemy obliczeń liczbowych, dla których matematyka, rozumiana w starym, XVIII wiecznym stylu (jako nauka o liczbie i punkcie) stanowiła naturalne odniesienie.

Dziś wiemy, że takie zadania numeryczne, związane z nauką, należą do najprostszych. To prawda, że mamy świadomość nieobliczalnych efektywnie (jak również teoretycznie) problemów, jednak prostota zadań obliczeniowych polega na tym, że mamy zwykle do czynienia — w klasie problemów naukowych — z bardzo dobrze opracowaną teorią obliczeń matematycznych lub rachunków przybliżonych. Jeżeli nawet jakiś naukowy problem jest bardzo złożony, to matematyczne twierdzenia, np. o istnieniu pewnych rozwiązań, są gwarancją sukcesu, a sztuka programowania polega na znalezieniu odpowiednio wydajnych algorytmów, czyli ogólnych metod skutecznie prowadzących do osiągnięcia zamierzonego celu. Tylko w tym sensie programowanie naukowe jest proste. Jednakże coraz częściej zaczęto używać komputerów w innych dziedzinach nauki i techniki, lecz tu pojawiły się dodatkowe kłopoty.

Prace nad symbolicznym przetwarzaniem informacji wprowadziły zupełną nowość do technik komputerowych. Zaowocowało to dość nietypowymi rezultatami. W 1968 roku Edsger W. Dijkstra, z uniwersytetu w Eindhoven, napisał ważny artykuł dotyczący programowania strukturalnego. W nowym ujęciu programowania jedna z podstawowych instrukcji sprzętowych — instrukcja skoku — została całkowicie wyeliminowana[4]. Skutkiem tego warstwa logiczna oprogramowania uległa kolejnemu krokowi ewolucji, mimo że w warstwie binarnej, po translacji programu źródłowego, skoki oczywiście nadal miały miejsce. Algorytm stał się bardzo elegancką strukturą językową. Znów pojawiła się iluzja logicznej prostoty: wystarczyło opanować kilkadziesiąt słów języka programowania, aby mieć narzędzie do tworzenia złożonych aplikacji.

[4]Zob.: Tadeusiewicz, op. cit., s. 252.

2. Złożoność w matematyce a złożoność w informatyce

Komputery powoli wchodziły w dziedziny do tej pory „odporne" na cyfrowe ujęcia. Informatyka natknęła się jednak na zupełnie inny rodzaj złożoności niż ten spotykany w „naukowym" świecie. Fizyk zwykle wierzy, że naukowe problemy mają jakiś matematyczny model, informatyk zaś jest świadom, że w wielu dziedzinach taki model jest nieznany lub zbyt skomplikowany. Co więcej, złożoność informatyczna wydaje się być czymś typowym i nie może być pominięta w takim sensie, w jakim układ planetarny może być przybliżony przez punktowe masy.

Złudzenie prostoty oprogramowania występuje w bardzo małych, amatorskich programach i systemach izolowanych. W tych sytuacjach oprogramowanie można traktować jak rzecz, którą da się wymienić na doskonalszą, bez stosowania poprawek czy ulepszeń. Nie będziemy się tutaj zajmować takimi tworami informatycznymi, a skupimy się na złożonych systemach informatycznych, po których naszym przewodnikiem będzie G. Booch[5]. Informatyczną złożoność dobrze odzwierciedlają systemy bazodanowe przechowujące miliony rekordów informacji uaktualnianej i nieustannie konsultowanej. Cechują się one długim czasem życia i wieloma zależnościami pochodzącymi od różnorodnych czynników. Taka złożoność przekracza możliwości konstrukcyjne pojedynczego człowieka, a w opanowaniu jej bynajmniej nie chodzi o to, by ona zniknęła. Strategią, która może prowadzić do sukcesu w opanowaniu złożoności w nowych dziedzinach może być podpatrywanie tego, w jaki sposób radzą sobie ze złożonością inne nauki.

Cytowany przez Boocha Brooks uważa, że złożoność oprogramowania nie jest cechą przypadkową. Wynika ona przede wszystkim ze złożoności dziedziny ujmowanego informatycznie problemu, z trudności w opanowaniu rozwoju oprogramowania i jego elastyczności oraz z samej natury systemów dyskretnych.

Złożoność dziedziny problemu wynika z miriad różnych interakcji między elementami systemu. Pojawiają się współzawodnictwa a nawet sprzeczne żądania różnych podsystemów, na które nakładają się „naturalne" wymagania, takie jak: użyteczność, koszty, zaufanie do oprogra-

[5]G. Booch, *Object–oriented analysis and design with applications*, The Benjamin/Cummings Publishing Company, Inc., second edition, 1994. W dalszej części dotyczącej informatyki będziemy odwoływać się pracy Boocha.

mowania. Zauważmy, że te cechy, które można by nazwać Turingowskimi, choć są kluczowe, to zdają się ginąć pod ciężarem wyżej wspomnianych.

Bogactwo dziedziny jakiegoś problemu jest opisywane przez obszerne tomy naukowych prac i szkiców. Charakteryzuje się ono wielością interpretacji oraz, niestety, trudnością przekazu treści. Z informatycznego punktu widzenia może ono zawierać także żądania wzajemnie ze sobą konkurujące. Co więcej, wprowadzanie oprogramowania do jakiegoś środowiska często zmienia stawiane przed nim żądania, tak więc zakładana dziedzina problemu zmienia się: pojawiające się elementy programowe modyfikują reguły, które wydawały się rządzić danym problemem. Ponadto cechą złożonych zagadnień jest ich tendencja do ewolucji w czasie i związana z tym zmiana wymagań co do ich rozwiązań.

Elastyczność charakterystyczna dla współczesnych języków programowania pozwala operować wysokim poziomem abstrakcji. Tutaj jednakże, podobnie jak w matematyce, nie istnieją żadne standardy prowadzące do osiągnięcia oczekiwanego sukcesu. Każdy ujmowany programowo problem wymaga indywidualnego potraktowania. Z tej racji modelowanie informatyczne nowego problemu rozpoczynać należy „od zera", co jest bardzo kłopotliwe. Dobra filozoficzna perspektywa mogłaby tu być wyznacznikiem wspólnego odniesienia dla wielu konkretnych dziedzin.

W informatyce mamy do czynienia z innym ważnym problemem. Modelowanie w systemach dyskretnych ma bowiem nieprzewidywalny przebieg. Systemy dyskretne łatwo doprowadzić do zachowań chaotycznych. Chaos pojawia się nie tylko w zjawiskach, o których traktują nauki takie jak np. meteorologia, biologia czy chemia, ale ulegają mu nawet cyfrowe sieci komputerowe. Atraktory porządkujące w pewien sposób zjawiska chaotyczne nie są wielką nadzieją dla symulacji komputerowych. Elementarne doświadczenia z komputerowymi symulacjami pozwalają przekonać się, że występowanie w nich chaosu jest czymś niemalże „normalnym". Sytuację pogarsza dodatkowo fakt, że wielkie aplikacje muszą opanować tysiące zmiennych i są zwykle wielowątkowe. Grozi to kombinatoryczną eksplozją wzajemnej zależności parametrów. Poza tym systemy informatyczne mają, niejako z założenia, oddziaływać z otoczeniem, a to powoduje, że układ nie może być opisany w sposób deterministyczny. Powszechnie znanym skutkiem nieprzewidzianej interakcji z otoczeniem bywa po prostu „zawieszenie się" systemu. W systemie modelowanym funkcjami ciągłymi jest prawie niemożliwe, aby zaświecenie światła przez pasażera spowodowało awarię i upadek samolotu; w systemie dyskretnym coś podobnego może mieć miejsce.

Niestety nie dysponujemy odpowiednimi narzędziami matematycznymi, które miałyby zastosowanie do opisanej wyżej sytuacji i musimy polegać na testach i zadowolić się pewnym poziomem zaufania. Złożoność oprogramowania prowadzi często do sytuacji bez wyjścia. Pojawia się zatem pewien paradoks: coraz więcej specjalistów zajmuje się utrzymaniem systemów informatycznych, a nie ich rozwojem. Wobec złożoności informatycznej jesteśmy nadal bezsilni mimo iluzji, że sobie z nią radzimy.

3. Filozoficzne spojrzenie na złożoność świata

Od pewnego czasu podkreśla się konieczność wzajemnego „śledzenia się" dyscyplin naukowych mających do czynienia ze złożonością. Także informatyka zdaje się potrzebować pewnej filozofii, która by ją bezpiecznie wprowadziła w problemy, wobec których sama informatyka jest bezradna. Dziś w wielu dziedzinach życia z powodzeniem wykorzystuje się systemy informatyczne, chociaż nie zawsze stosowane są najnowsze osiągnięcia.

Przykładem nauki, która zajmuje się złożonością *par excellence* jest oczywiście biologia. Organizmy żywe mają jednak pewną wspólną cechę, jaką jest struktura komórkowa pojawiająca się na każdym szczeblu wyższego morfologicznie zhierarchizowania (korzeń, łodyga, liście). W analizie podobnych, złożonych układów dostrzeżono, że na każdym poziomie takiej abstrakcji wyróżnić można pewne funkcje jasno odgraniczające od siebie poziomy złożoności. Wspólne elementy abstrakcyjne nie koniecznie oznaczają się „ontologiczną" tożsamością. Chociaż w biologii komórka stanowi podstawowy element strukturalny, to oczywiście w organizmach występują różne komórki. Taka hierarchiczna budowa całości systemu pozwala wyróżnić wiele niezależnych ogniw, charakteryzujących się zachowaniami emergentnymi. W złożonych układach całość przejawia przy tym charakterystyki nieredukowalne do sumy swych części (nieliniowość). W hierarchiach tych występują także powszechne „wirtualnie" elementy, takie jak komórki, system kapilarny, itd.

Aby nie pozostać tylko na przykładzie biologii można się odwołać również do fizyki i badanej przez tę naukę struktury materii. Podstawowe oddziaływania: grawitacyjne, elektromagnetyczne, jądrowe silne i słabe odpowiadają dobrze wyróżnionej hierarchii, która wykazuje także powszechne, „wirtualnie" wspólne cechy takie, jak np. prawa zachowania energii, krętu, itd. Podobnie z punktu widzenia socjologii wyodrębnić

można grupy wykonujące działania, które nie byłyby możliwe do wykonania przez pojedynczą osobę. W ten sposób widzimy pewną hierarchię poczynając od małych grup, poprzez grupy zawodowe, regionalne, narodowe czy państwowe.

Obserwacja innych dziedzin naukowych prowadzi do wniosku, że w układach złożonych interakcje między warstwami systemu są ograniczone (np. listonosz nie „oddziałuje" bezpośrednio na szefa firmy, ale intensywnie „oddziałuje" z urzędem pocztowym i biurami). Ponadto okazuje się, że różne poziomy często jednoczy pewien wspólny „mechanizm". Pozwoliło to na wyodrębnienie następujących cech systemu złożonego:

- system taki ma hierarchiczną strukturę
- podział na komponenty jest w nim względny
- wewnętrzne powiązania wewnątrz warstw hierarchii są silniejsze niż pomiędzy nimi
- podsystemy wzajemnie na siebie oddziałują i często mamy do czynienia z istnieniem pewnego wspólnego wzorca ponownego (rekursywnego) wykorzystania komponentów
- w systemach tych może pojawić się ewolucja struktur wtedy, gdy są możliwe stabilne formy (stany) pośrednie
- system jest lepszy, gdy wyewoluował z jakiegoś stanu stabilnego; tworzony od początku system złożony prawie nigdy nie funkcjonuje poprawnie.

W złożonych systemach zachodzi pewna prawidłowość: to, co było złożone na jednym poziomie staje się prostym elementem budowy na poziomie dziedziczącym. Zwykle jednak nie można opierać się tylko na jednej hierarchii dla całości, ale trzeba wprowadzać wiele różnych rodzajów hierarchii. Zauważmy przy tym pewną różnicę: „bycie hierarchią" (klasą) jest czymś różnym od „bycia częścią hierarchii" (obiektem), czyli oba te pojęcia są ortogonalne.

Badania różnych dziedzin wywołały w informatyce palącą potrzebę wypracowania nowego paradygmatu, zwanego obiektowym. Opiera się on na swoistej filozofii zasadniczo różnej od tej, która powszechnie kojarzy się z komputerami. W tym paradygmacie bowiem koncepcja algorytmu wcale nie pełni podstawowej roli. W klasycznym paradygmacie

programowania strukturalnego wszystko było algorytmem, opierało się na krokach prowadzących do rozwiązania problemu. Algorytmy miały swoją strukturę, dzieliły się na moduły itd. Dogmat ten implikował dekompozycję algorytmiczną jakiegoś systemu. W wielkich systemach ma on jednak poważne braki. Nie jesteśmy bowiem w stanie tworzyć bezbłędnych i wystarczająco ogólnych programów. Co więcej, prace Turinga (problem stopu), są jak połyskujący miecz przeciwko takim próbom. Właśnie dlatego w ostatnim czasie zaczął się rozwijać bardzo dynamicznie inny sposób dekompozycji systemów złożonych, jakim jest podejście obiektowe.

Paradygmat obiektowy od razu stawia wobec nas wymaganie „usadowienia się" w dziedzinie problemu i dokonywania w niej kluczowych abstrakcji. Skutkiem tego jest wykreowanie autonomicznych agentów zdolnych do interakcji, prowadzących do wytworzenia zachowań wyższego rzędu. Pojawiające się metody algorytmiczne nie mają już całościowych aspiracji, gdyż rolą algorytmu jest modelowanie zachowań rzeczywistego przedmiotu. Tak więc algorytm zależy *explicite* od struktury obiektu (podobna zależność była znana już w programowaniu strukturalnym). Obiekt (np. jakiś „agent") wykonuje przypisane mu operacje na wyraźne żądanie, imitując jakiś przedmiot z dziedziny. Ponieważ dekompozycja ma miejsce w dziedzinie problemu, zrozumienie oprogramowania przez innych (np. użytkowników!) jest prawie naturalne. Naturalne jest też zachowanie się oddziałujących na siebie obiektów (agentów, serwerów i klientów).

Niestety, podejście algorytmiczne stoi w relacji ortogonalnej do podejścia obiektowego. W związku z tym, nie można jednocześnie opowiedzieć się za algorytmem i za obiektem. Oba podejścia mają oczywiście swoje zalety i wady. Filozofia, która płynie z obserwacji świata wydaje się preferować podejście obiektowe. Jak sądzę, dalsze postępy w filozofii mogą wzmocnić ten cieszący się wielką popularnością paradygmat informatyczny.

* * *

W przestawionym referacie poruszyliśmy zagadnienie złożoności w wielkich systemach informatycznych. Wychodząc od złożonych układów fizycznych widzieliśmy, jak tworzą one *logicznie* prostszą strukturę wyższego poziomu. Podobne zjawisko dostrzegliśmy w warstwie programowych operacji prowadzących do programowania strukturalnego. Okazuje się jednak, że algorytmiczne podejście do modelowania świa-

ta napotyka na wewnętrzne bariery, czego skutkiem było pojawienie się podejścia obiektowego do złożonych zagadnień. Obiektowy paradygmat sam z siebie dostarcza ciekawej wizji świata, ale potrzebuje solidnej filozofii do tego, by mógł być bezpiecznie aplikowany do rzeczywistości. Wymagania te mają nie tylko ontologiczno–epistemologiczne konsekwencje. Coraz powszechniejsza wydaje się także potrzeba etycznej refleksji wokół informatyki.