

Kultura *software'u*. Wpływ oprogramowania na humanistykę

ANNA PAPRZYCKA

Uniwersytet im. Adama Mickiewicza w Poznaniu
ap70298@amu.edu.pl

Wprowadzenie. Kod jako niewidoczne spoiwo

Trudno nie zgodzić się z tezą, że oprogramowanie posiada obecnie przeważający wpływ na życie społeczne, infrastrukturę, technologię czy komunikację. Pomimo tak znaczącej siły oddziaływania, często bywa ono wypierane z powszechnej świadomości zbiorowej, dyskursu kultury masowej, a nawet z badań naukowych. Właśnie z tego powodu znawca mediów, Lev Manovich, zwracał uwagę na symultaniczną transparentność oprogramowania oraz jego fundamentalną pozycję w kulturze: „Szkoły, szpitale, bazy wojskowe, laboratoria naukowe, lotniska, miasta – wszelkie społeczne, ekonomiczne i kulturowe systemy współczesnego społeczeństwa – są napędzane przez oprogramowanie. Kod jest niewidzialnym klejem, który wiąże wszystko ze sobą [*The school and the hospital, the military base and the scientific laboratory, the airport and the city – all social, economic, and cultural systems of modern society – run on software. Software is the invisible glue that ties it all together*]” (Manovich 2013: 8). Sformułowana przez Manovicha diagnoza kondycji współczesnego społeczeństwa staje się jeszcze bardziej przekonująca, gdy weźmie się pod uwagę fakt, że przypatrując się otaczającej rzeczywistości niełatwo wskazać jakikolwiek wytwór działalności człowieka, który nie miałby w sobie oprogramowania lub w samym procesie produkcji po-

zbawiony byłby styczności z komputerem. Komputeryzacja jest nieuniknionym elementem rozwoju technologicznego i wraz z upływem czasu zaczyna obejmować kolejne aspekty życia człowieka, przy czym ekspansja ta odbywa się w sposób bardzo dyskretny, niemal nieodczuwalny. Właśnie z tego powodu Manovich nazwał oprogramowanie niewidzialnym klejem. Spaja ono bowiem większość elementów funkcjonujących w systemie stabilizującym epokę cyfrową, jednak jego reprezentacja wizualna w naturalnym procesie ewolucji kulturowej zastała wchłonięta w obręb zbioru reprezentacji. Innymi słowy, wszystko, za czym ukrywa się kod – interfejsy ekranów komputerów, telefonów, nawet sposób prezentowania informacji na plakatach, wszelkie cyfrowe urządzenia technologiczne, systemy zarządzające, jak na przykład sygnalizacja świetlna i wiele innych elementów rzeczywistości cyfrowej kultury – zostały społecznie oswojone i zaakceptowane jako regularny element otaczającego świata.

Skoro oprogramowanie ma tak duży wpływ na rzeczywistość – bo nie ma dziedziny życia, której by w jakiś sposób nie organizowało – niezbędne jest wypracowanie teoretycznych narzędzi krytycznych, które pozwolą na ujawnienie mechanizmów rządzących współczesną kulturą. Jednym z propagatorów tego sposobu myślenia był wspomniany już Lev Manovich. W dobie konsumpcjonizmu, w której szczególnie trudno wykroczyć poza bierne przetwarzanie informacji produkowanych przez kulturę masową, wśród priorytetów nauk humanistycznych, a przede wszystkim medioznawstwa, znalazło się rozwinięcie badań nad kulturową rolą oprogramowania. Dzięki temu powstaje aparat krytyczny, za pomocą którego oprogramowanie pozwala postrzegać się jako polityczne, kulturotwórcze, uwikłane w systemy ekonomiczne i społeczne, ale przede wszystkim przestaje już być materią niewidzialną.

Software studies

Manovich zwracał szczególną uwagę na znaczenie oprogramowania we współczesnej kulturze i podkreślał potrzebę poprowadzenia dyskursu naukowego poddającego tę kwestię analizie:

Choć możemy uważnie przestudiować „kulturę oprogramowania” – praktyki programowania, wartości i ideologie programistów i firm dystrybuujących oprogramowanie, kulturę Doliny Krzemowej i Bangaluru itd. – to jeśli będziemy ograni-

czać się tylko do tego, rozminiemy się z prawdziwym znaczeniem oprogramowania (Manovich 2013: 32)¹.

Badacz mediów za pomocą tych słów nawoływał do rozpoczęcia poszukiwań prawdziwego znaczenia kodu. Zgodnie z teorią Manovicha, *software* jest czymś więcej niż elementem komputera umożliwiającym jego działanie – kod jest w istocie czynnikiem zdolnym kształtować rzeczywistość, podobnie jak alfabet, matematyka, silnik spalinowy, prasa drukarska lub elektryczność (Manovich 2013: 32).

Określenie *software studies* odnosi się do nowego kierunku w obszarze humanistyki, skupiającego się na badaniu kulturowego znaczenia oprogramowania. Zostało po raz pierwszy sformułowane w książce *Język nowych mediów* przez Lva Manovicha, który pisał:

Żeby zrozumieć nowe media, musimy odwołać się do informatyki. To właśnie tam znajdziemy nowe terminy, kategorie i funkcje charakteryzujące media, które stały się programowalne. Od medioznawstwa zmierzamy w stronę czegoś, co można by nazwać programoznawstwem, czyli od teorii mediów do teorii oprogramowania (Manovich 2006:117)².

W 2007 roku Manovich wspólnie z Matthew Fullerem podjęli się założenia laboratorium Software Studies Initiative w Nowym Jorku. Obecnie centrum posiada drugą siedzibę w La Jolla w Kalifornii. Celem tego przedsięwzięcia było zorganizowanie interdyscyplinarnej grupy badawczej, która wypracuje i przedstawi metodologię opisywania zjawisk związanych z oprogramowaniem po to, aby spopularyzować ten dyskurs w badaniach dotyczących kultury.

Konstytutywna dla Software Studies Initiative była seria publikacji pod hasłem Software Studies wydana przez The MIT Press. W jej skład weszło pięć pozycji: *Software Studies: A Lexicon* pod redakcją Matthew Fullera (2008)³,

¹ Bengaluru to metropolia, nazywana indyjską Doliną Krzemową przez wzgląd na jej wpływ na rozwój technologiczny na Wschodzie. Przekład własny za: „And while we can certainly study »the culture of software« – programming practices, values and ideologies of programmers and software companies, the cultures of Silicon Valley and Bangalore, etc. – if we only do this, we will miss the real importance of software”.

² Piotr Cypriański przetłumaczył tu „software studies” jako „teorię oprogramowania”.

³ Jeszcze przed wydaniem serii Software Studies przez MIT Press, Matthew Fuller napisał w 2003 książkę *Behind the Blip. Essays on the Culture of Software* (Fuller 2003), w której zawarł propozycję stworzenia nurtu krytycznego w odniesieniu do oprogramowania

Expressive Processing: Digital Fictions, Computer Games, and Software Studies (Wardrip-Fruin 2009), *Programmed Visions: Software and Memory* (Hui Kyong Chun 2011), *Code/Space: Software and Everyday Life* (Kitchin & Dodge 2011), *Speaking Code: Coding as Aesthetic and Political Expression* (Cox & McLean 2012). Rok później została wydana podsumowująca publikacja Manovicha pod tytułem *Software Takes Command* (Manovich 2013).⁴

Wydana jako pierwsza pozycja z serii, *Software Studies: A Lexicon* była antologią krótkich tekstów napisanych przez artystów, designerów, programistów, badaczy kultury i innych specjalistów z różnych dziedzin. Książka ta miała prezentować wstępne założenia grupy Software Studies Initiative, wśród których można wymienić chęć stworzenia pola komunikacji pomiędzy badaniami na temat mediów, analizami kulturowymi i naukami ścisłymi. Autorzy powzięli za cel wypracowanie interdyscyplinarnego języka, z którego będą mogli korzystać przedstawiciele wszystkich dziedzin, w szczególności humanistycznych (Fuller 2008). Publikacja ta wywołała poruszenie także na gruncie polskim, ponieważ w tym samym roku przygotowany został egzemplarz czasopisma „Kultura Popularna”, który poświęcony został tematyce *software studies*. Zawarto w nim teksty znamienitych polskich artystów sztuki medialnej – Pawła Janickiego oraz Michała Szoty – a także wywiad z Aleksandrem Galloway’em. Główną część numeru przeznaczono na publikację przetłumaczonych na język polski wybranych rozdziałów *Software Studies: A Lexicon*. Polski medioznawca, Mirosław Filiciak, w artykule pochodzącym z omawianego numeru, zadał następujące pytanie:

Czy ta – trzeba to podkreślić – radykalna zmiana paradygmatu w studiach nad kulturą pozwoli nam ją lepiej zrozumieć? Wydaje się, że w chwili obecnej przedstawiciele studiów nad oprogramowaniem przede wszystkim wskazują na zaniedbywany obszar badań kulturoznawczych i dyskutują, jakich narzędzi użyć, by sporządzić jego mapę. Droga do celu – zrozumienia tego, co znajduje się po drugiej stronie lustra ekranu – jest jednak wciąż długa (Filiciak 2008: 18).

(*software criticism*). W tej publikacji Fuller zamieścił wnikliwą analizę działania popularnego oprogramowania, takiego jak wyszukiwarki internetowe albo edytory tekstów. Nie opierała się ona na opisie zewnętrznego działania tych programów, ale za jej pomocą autor starał się określić sposób, w jaki w ich obrębie funkcjonowało oprogramowanie i jak warunkowało ich działanie oraz zachowanie użytkownika.

⁴ Pierwsze wersje tej publikacji zostały udostępnione w sieci w 2008 roku, jednak kompletne, papierowe wydanie ukazało się w roku 2013.

Badacz trafnie podsumował charakter pierwszej publikacji z serii *Software Studies*, podkreślając, że miała ona charakter wstępnych rozpoznai w nowo powstającej dziedzinie. Jednak w ciągu następnych lat zaczęły się pojawiać kolejne pozycje z owego cyklu, zawierające konkretne propozycje metodologiczne.

Druga książka z serii *Software Studies, Expressive Processing: Digital Fictions, Computer Games, and Software Studies*, kontynuowała ideę tworzenia narzędzi językowych służących do mówienia o oprogramowaniu poza specjalistycznym dyskursem informatycznym. Noah Wardrip-Fruin prezentował metody mówienia o działaniu algorytmów i ich konsekwencjach estetycznych bez wchodzenia na poziom składni języków programowania. Autor zwracał uwagę na to, że pierwsze przykłady analiz kulturowych oprogramowania pochodzą z badań nad grami komputerowymi (odwoływał się do przykładu hipertekstowej struktury programu ELIZA z 1966 oraz operatorów logicznych rządzących grą *The SimCity*). Rozważania dotyczące logiki oprogramowania gier stały się dla Wardrip-Fruina podstawą dla wprowadzenia do analizy systemów sztucznej inteligencji (Wardrip-Fruin 2009). Z kolei Wendy Hui Kyong Chun przedstawiła społeczną perspektywę myślenia o oprogramowaniu. Autorka *Programmed Visions: Software and Memory* zaznaczała, że kod, wraz ze specyficznymi dla siebie właściwościami, wpłynął na kształt systemów władzy oraz ekonomii, a także metafor, za pomocą których człowiek określa otaczający go świat. Sieciowa natura oprogramowania przygotowała grunt dla rozwoju neoliberalnych technologii rządowych. Hui Kyong Chun poświęciła dużą część tej publikacji objaśnianiu różnic pomiędzy pojęciami takimi jak na przykład *hardware*, *firmware* i *wetware* albo między kodem źródłowym, skompilowanym kodem a spisаныmi instrukcjami. Badaczka podkreślała również wartość pisanie o tak zwanym *vaporware*, czyli o oprogramowaniu, które nigdy nie weszło do użytku publicznego, pomimo że praca nad nim została rozpoczęta. Jest to istotne z tego względu, że wielu badaczy, wśród których wymienić można Alexandra Gallowaya (2004: 17), opowiadało się za odrzuceniem rozważań na temat *vaporware*, zarzucając tego typu badaniom hipotetyczny charakter. Hui Kyong Chun zwracała uwagę na to, że projekty programów związane z *vaporware* nie były przypadkowymi zdarzeniami w historii rozwoju oprogramowania, ale mogą one stanowić podstawę do jej rozumienia. Specyficzną cechą analiz przeprowadzonych przez tę badaczkę jest to, że uwzględnia ona zjawiska, które nie wpisują się w główny nurt historii oprogramowania, takie jak marzenia z nim związane, oczekiwania czy pomyłki (Hui Kyong Chun 2011).

Kolejną, bardzo znaczącą pozycją na liście publikacji serii *Software Studies*, jest *Code/Space. Software and Everyday Life*, napisana przez Roba Kitchina i Martina Dodge'a. Książka ta wyznaczyła w obrębie badań nad oprogramowaniem innowacyjny kierunek, zorientowany na pojęcie nowej geografii. Autorzy tej publikacji wprowadzili nową kategorię geograficzną – kodoprzestrzeń. Termin ten odnosił się do specyficznego dla współczesnej kultury cyfrowej statusu przestrzeni, w którym wyraża się fakt, że nie może ona być już dłużej postrzegana wyłącznie z perspektywy wymiaru materialnego, fizycznego. W kodoprzestrzeni jest ona kształtowana przez oprogramowanie, które nie tylko wpływa na jej formę, ale również jest jej nieodłączną częścią, koordynuje systemy nią zarządzające. Kod jest tu rozumiany jako kategoria przestrzenna. Współcześnie stworzenie funkcjonalnych środowisk, takich jak na przykład supermarket albo szpital, nie jest możliwe bez udziału oprogramowania, które podłącza je do sieci wymiany informacji i pozwala funkcjonować w systemie miejskim. Rozważania Kitchina i Dodge'a wpisują się w nurt analiz oprogramowania przeprowadzanych z perspektywy społecznej. Według badaczy oprogramowanie ma tak duży wpływ na miejską infrastrukturę, że może być brane pod uwagę jako osobny, pozaludzki aktant w analizie kulturowej (Kitchin & Dodge 2011: 23). Zgodnie z ich rozpoznaniem, kod nie jest już blokiem tekstu, zestawem instrukcji, nośnikiem danych, ale przestrzenią. Dlatego też może przenikać się z wszelką materią fizyczną. Samą przestrzeń należy tu postrzegać jako czynność wywołaną i określoną przez język programowania. Autorzy *Code/Space. Software and Everyday Life* uznawali podwójną naturę oprogramowania. Według nich było ono jednocześnie produktem i procesem (*Software as both – product and process*; Kitchin & Dodge 2011: 32).

Serię pięciu publikacji fundamentalnych dla Software Studies Initiative zamyka *Speaking Code. Coding as Aesthetic and Political Expression* autorstwa Geoffa Coxa, który wziął odpowiedzialność za jej treść naukową i Alexa McLeana, który wzbogacił ją przez napisane przez siebie oprogramowanie. W tej książce została zawarta propozycja analizy oprogramowania z perspektywy badań nad językiem i ekspresją językową. Jej autorzy postrzegali kod jako narzędzie biorące udział w procesie komunikacji, które transmituje zawarte w nim znaczenie zarówno ludzkim, jak i nie-ludzkim odbiorcom. Analizowali oni również sprawczość i performatywność kodu, jego wpływ na odbiorcę, a także systemy społeczne, ekonomiczne i polityczne. Tytułowe „speaking code” ma na celu podkreślenie, że kod jest środkiem wyrazu, za pomocą którego człowiek może się komunikować. Cox zaznacza, że proces ten działa w obie strony i kod może rów-

niez wyrażać się przez człowieka (*Code is speaking us*) (Cox & McLean 2013: IX). Oznacza to, że oprogramowanie kształtuje sposób myślenia i poruszania się oraz postrzegania rzeczywistości. Kod mówi przez człowieka, przy czym sam pozostaje niewidzialny, a użytkownik nie poddaje tego żadnej refleksji, nie szuka metod na przepracowanie tego procesu. Cox zwracał również uwagę na polityczny charakter mowy, szczególnie zaś komunikatów wypowiadanych publicznie, które przybierają formę aktów wykonywanych na rzecz społeczeństwa. Oprogramowanie także funkcjonuje w obiegu publicznym, a rozumiane jako akt mowy, nabiera politycznego charakteru. Cox podkreślał tę zależność, przy czym bronił on języka (również rozumianego jako oprogramowanie) przed całkowitą inwazją systemów politycznych i ekonomicznych. Programowanie powinno zachować autonomiczność środka wyrazu ludzkiej ekspresji, jakkolwiek paradoksalne może się to wydawać w erze wszechobecnej komputeryzacji (Cox & McLean 2013).

Innym interesującym zagadnieniem, które omawia autor *Speaking Code. Coding as Aesthetic and Political Expression*, jest myślenie o oprogramowaniu jako o samospełniającej się przepowiedni. Teza ta powstała w oparciu o teorię Johna Langshawa Austina, dotyczącą sprawczości aktów mowy i performatywów (Cox & McLean 2013: X). Cox podkreślał, że kod zawiera w sobie dane na temat potencjalnego statusu rzeczywistości jeszcze zanim zostanie uruchomiony. Istotne jest, że badacz przypisywał oprogramowaniu charakter indeterministyczny – a więc jeden zestaw reguł może doprowadzić do wielu różnych rozwiązań (Cox & McLean 2013: 6).

Publikacja Lva Manovicha, inicjatora Software Studies Initiative, została wydana jako ostatnia. *Software Takes Command* stanowi zarówno syntezę poszukiwań badawczych tego ugrupowania, jak i wyraz zainteresowań naukowych badacza nowych mediów. W książce tej Manovich zwracał uwagę, że tak samo, jak istnieje rzetelnie prowadzony dyskurs związany z historią sztuki i na przykład historią perspektywy malarskiej, tak nie jest w podobnej skali rozwijany nurt dotyczący historii technologii i, co za tym idzie, historii oprogramowania (Manovich 2013: 4). Jako przyczynę wytwarzania się poczucia ahistoryczności oprogramowania Manovich wskazywał sposób, w jaki komputery osobiste zostały wprowadzone do użytku publicznego i zaczęły pojawiać się w mieszkaniach. Komputery nie posiadają swojej mitologii, pojawiły się znikąd i w sposób nagły. Użytkownicy programów nie potrzebują znajomości historii, żeby z nich korzystać. Wątek historyczności technologii również nie jest szeroko podejmowany w naukowym dyskursie kulturowym. Z tego powodu badacz przeznaczył sporą część publikacji na przybliżenie historii oprogramowania.

Manovich podkreślał, że badania nad oprogramowaniem powinny uwypuklać jego różnorodność gatunkową, przedstawiać wielość rodzajów kodu, a nie tylko *visible software*, czyli ten używany powszechnie przez konsumentów, ale także *grey software*, który zarządza wszystkimi systemami współczesnego społeczeństwa (Manovich 2013: 21). Badacz przyznał, że nie posiada odpowiedniej wiedzy i kwalifikacji, żeby mówić o ukrytych systemach społecznych, skupił się więc na oprogramowaniu, którego używał w życiu codziennym i podporządkował je pod wypracowaną przez siebie kategorię *cultural software*⁵. Odnosiła się ona do wszelkiego rodzaju oprogramowania, które funkcjonuje w obiegu społecznym. Manovich podkreślał, że współcześnie przeniknęło ono całą kulturę. Nawet otworzenie pliku może być rozumiane jako *software performance*, ponieważ nie chodzi tu tylko o ekspozycję obrazu, lecz o fakt, że przed użytkownikiem otwiera się cała sieć możliwości związanych z edytowaniem, udostępnianiem i innymi funkcjami, które umożliwia *software* (Manovich 2013: 34). Dla badacza ważne było sformułowanie określenia odnoszącego się do oprogramowania, które w czytelny sposób osadzać je będzie w porządku kulturowo-społecznym. Traktował on oprogramowanie jako kolejny wynalazek rozwoju technologicznego, zupełnie rewolucjonizujący każdą dziedzinę ludzkiego życia.

Software Studies Initiative to aktywny ośrodek naukowy, który wprowadza w obszar nauk humanistycznych innowacyjne i potrzebne dziś kierunki poszukiwań. Trzeba jednak zaznaczyć, że rozważania na temat kulturowego znaczenia oprogramowania były podejmowane wcześniej, ale nigdy w tak zorganizowanej postaci. Pod koniec XX wieku niemiecki badacz mediów, Friedrich Kittler, zasłynął ze sformułowania odważnej tezy, iż każdy student, który zajmuje się badaniem kultury, powinien znać co najmniej dwa języki programowania (Kittler 1996: 741). Duży wkład w rozwój badań nad oprogramowaniem miał również wspomniany już Alexander Galloway, który zwracał uwagę na brak krytyki politycznego charakteru oprogramowania (Galloway 2006: 319), a także analizował relację pomiędzy kodem a interfejsem w książce *The Interface Effect*.

⁵ Określenia tego użył po raz pierwszy w 2001 roku w książce *Język nowych mediów*, natomiast w *Software Takes Command* korzystał z niego i szczegółowo je opisał.

Ujawnianie oprogramowania na poziomie kulturowym

Pomimo iż popularność teorii wypracowanych w ramach Software Studies Initiative stale rośnie, wydaje się, że badania humanistyczne nie są w stanie rozwijać się tak samo szybko, jak nabierający coraz większego tempa postęp technologiczny, mający bezpośredni wpływ na współczesną kulturę. Niesymetryczność tę dostrzegał również Matthew Fuller, zwracający uwagę na to, że wzrastającym teoretycznym refleksjom na temat kulturowego znaczenia *software'u* powinny towarzyszyć stworzone w bezpośrednim związku z nimi języki programowania, aplikacje i programy, udostępniane w ramach licencji Open Source (Fuller 2008: 7). To połączenie pozwala na jednoczesne rozwijanie się humanistycznych teorii naukowych oraz rozpowszechnianie ich idei za pomocą środków komunikacji masowej, co znacznie przyspiesza proces ich dystrybucji, a także zasięg ich oddziaływania. Do realizacji tak sformułowanych założeń przystąpili między innymi Ben Fry i Casey Reas, dwaj studenci MIT, którzy stworzyli język programowania Processing. Miał on służyć zbudowaniu przestrzeni komunikacji pomiędzy przedstawicielami wszelkich dziedzin zarówno ścisłych, jak i humanistycznych czy artystycznych. Oczekiwano więc, że stanie się on swego rodzaju uniwersalnym językiem programowania, który pozwoli każdemu użytkownikowi, bez względu na jego kwalifikacje, nauczyć się programować i, co za tym idzie, prowadzić samodzielny dyskurs w kulturze, która dostarcza gotowych narzędzi i interfejsów. Do fundamentalnych cech Processingu należy jego przystępność oraz darmowa licencja *open source*, która pozwala jego użytkownikom na rozwijanie tego języka. Dzięki temu w sieci zorganizowały się duże wspólnoty kreatywne dzielące się między sobą inspiracjami oraz przygotowanymi przez siebie poszerzeniami. Filozof i matematyk, Daniel Schiffman, założył The Processing Foundation, której misją jest popularyzowanie tego języka programowania poprzez organizację konferencji i warsztatów czy udostępnianie szkoleń w internecie. Wart uwagi jest fakt, że Processing przyczynił się do rozpowszechniania świadomości oprogramowania na poziomie kulturowym, a więc zrealizował cel bliski badaczom z nurtu *software studies*. Przykład ten umacnia myśl Fullera, który wskazywał potrzebę uzupełniania się teorii naukowych i alternatywnego oprogramowania.

Większość artystów, którzy generują dzieła przy pomocy algorytmów komputerowych, nie dokłada starań do tego, aby wraz z efektem końcowym przedstawić również sposób działania programu, na którym został on oparty. Najczęściej, nawet w opisie prac i wykorzystanych narzędzi, nie wymieniają

oni nazw programów, z których korzystali. Zazwyczaj przy określaniu materiałów składających się na dany projekt, autorzy uciekają się do używania ogólnej formuły *custom software*. Kwestia ta była jednak szczególnie istotna dla jednego z twórców Processingu – Bena Fry’a. W *Deprocess* wydrukował on na dużym formacie cały kod, który był podstawą jednego z obrazów z cyklu *Process* autorstwa Casey’a Reasa (Fry 2008). Prace tego ostatniego stanowiły fantazyjne wizualizacje wygenerowane przez kod przygotowany w języku Processing. Dzieła te nie przybliżały działania oprogramowania, chyba żeby uznać, że robiły to na poziomie innym niż wizualny, na przykład poprzez zaintrygowanie odbiorcy niecodziennością wygenerowanych wzorów, co mogłoby spowodować, że zainteresowałby się on zagadnieniem tego, w jaki sposób powstają podobne grafiki. Sam tytuł projektu Fry’a wskazuje na to, że celem artysty było podjęcie próby dekodowania jakiegoś procesu, rozłożenia go na części. Fry nie tylko wyeksponował ukryty za wizualizacją Reasa kod, ale również przedstawił, poprzez naniesienie na obraz niebieskich smug, sposób, w jaki komputer odczytuje kolejne linijki skryptu. Można było dostrzec, że proces ten przebiega nieliniarnie i wielokrotnie się zapętla. *Deprocess* miało w sobie coś z demistyfikacji – odsłaniało kod schowany za nieregularnymi grafikami Reasa. Praca ta miała edukacyjny charakter, ponieważ nie tylko zwracała uwagę na obecność kodu w kulturze cyfrowej, ale także przedstawiała sposób jego działania. Projekt Fry’a nie przybliżał odbiorcy metody funkcjonowania oprogramowania w sposób dokładny, precyzyjny, lecz raczej ogólny, poglądowy. Pozwalał on zauważyć elementarną różnicę pomiędzy tym, jak człowiek przeczytałby podobny blok tekstu, a w jaki sposób proces ten jest przeprowadzany przez program. Dzięki projektowi Fry’a kod okazał się czymś więcej niż monotonnym blokiem niezrozumiałego tekstu, bowiem artysta przedstawił go jako żywą, dynamiczną strukturę.

Tematyzowanie samego działania oprogramowania, a także jego estetyzacja nie są powszechną strategią w sztuce nowych mediów, a z pewnością jest to działanie, które również przyczynia się do wprowadzania kwestii software’u w obręb powszechnego obiegu treści kulturowych. Innym przykładem, realizującym tę ideę, jest instalacja *PRISM: The Beacon Frame*, której autorami są Julian Oliver i Danja Vasiliev. Artyści należą do The Critical Engineering Working Group, założonej w Berlinie w 2011 roku. Jej członkowie opowiadają się za krytycznym podejściem do oprogramowania i sformułowali jedenaście postulatów w upublicznionym w sieci manifestie. Jednym z najważniejszych założeń tego bardzo specyficznego dekalogu (zastosowana w nim numeracja,

podobnie jak w większości języków programowania, zaczyna się od cyfry zero) jest stwierdzenie, że inżynier krytyczny nie tylko wykorzystuje znaczące dla kultury języki programowania, ale również bada je po to, by ujawnić ich sposób działania i oddziaływania na rzeczywistość – w tym na użytkownika. Zadaniem krytycznego inżyniera jest też osadzanie *software'u* w kontekście polityczno-ekonomicznym (Oliver, Vasiliev & Savičić 2011). Instalacja *PRISM: The Beacon Frame* została przedstawiona podczas festiwalu Transmediale w 2014 roku. Centralnym punktem projektu była czarna walizka, posiadająca wbudowany niewielki komputer, projektor, sześcienny szklany pryzmat, antenę GSM oraz stację bazową łączności komórkowej. Tak złożony zestaw pozwalał na skanowanie znajdujących się w pobliżu wież transmisyjnych sieci komórkowych, określenie listy ich unikalnych właściwości, a następnie podszycie się pod tego typu obiekty. Doprowadziło to do tego, że telefony mobilne pozostające w zasięgu walizki podłączyły się do fałszywej wieży, traktując ją jako godną zaufania, co z kolei pozwoliło autorom instalacji na pobieranie danych o urządzeniu połączonych użytkowników i wysyłanie wiadomości do ich prywatnych telefonów. Zebrane dane były natychmiast przetwarzane oraz wyświetlane za pomocą niewielkiego rzutnika, będącego częścią zestawu. Wiązka światła była z kolei przepuszczana przez nieustannie obracający się pryzmat, dzięki czemu tworzyła na ścianach pomieszczenia służącego do ekspozycji zachwycającą, ruchomą wizualizację, która prezentowała prywatne dane znajdujących się w pobliżu zwiedzających. Zebrani wokół walizki nie musieli podjąć żadnej interakcji z instalacją, żeby ich urządzenia zostały podłączone do fałszywej sieci, co czyniło ją inwazyjną i wywołało negatywne reakcje. Celem pracy była polityczna krytyka kodu poprzez zwrócenie uwagi na korzystanie przez ośrodki władzy ze społecznej niewiedzy na temat oprogramowania. Zastosowano tu te same strategie lokalizacji i mapowania urządzeń bezprzewodowych, które są używane przez państwowe agencje nadzoru na przykład w Stanach Zjednoczonych czy Wielkiej Brytanii. Jednak większość uczestników Transmediale w 2014 nie odczytała intencji autorów instalacji, szybko wezwana została policja i praca musiała zostać usunięta z wystawy (Oliver & Vasiliev 2014). Instalacja *PRISM: The Beacon Frame* jest więc przykładem bardzo radykalnej strategii rozpowszechniania krytycznego dyskursu na temat oprogramowania. Dowodzi również, że duża część społeczeństwa nie jest gotowa na uświadomienie sobie pewnych mechanizmów rządzących polityką i ekonomią.

Podsumowanie

Istnieje społeczne zapotrzebowanie na wprowadzenie edukacyjnego dyskursu dotyczącego kwestii oprogramowania zarówno pod względem użytkowania, jak i zrozumienia jego kulturowego znaczenia. Dostrzegany współcześnie rozwój tego typu zjawisk może być również interpretowany z perspektywy prognostycznej jako znak, że w nadchodzącej przyszłości rola oprogramowania będzie sukcesywnie wzrastać, a umiejętność posługiwania się nim zyskiwać będzie coraz większą powszechność, być może nawet stanie się niezbędna do funkcjonowania w życiu społecznym i kulturowym. Prawdopodobne jest, że pojawiające się dziś w obiegu naukowym pojęcia, takie jak kodoprzestrzeń, oprogramowanie kulturowe czy kultura oprogramowania, stanowią dopiero zapowiedź nadchodzącej zmiany kulturowej. Już dziś powszechnie mówi się o przełomie cyfrowym i o tym, w jakim stopniu zredefiniował on podstawowe kategorie społeczno-kulturowe, takie jak sposób międzyludzkiej komunikacji i identyfikacji czy metodę zarządzania systemami politycznymi bądź ekonomicznymi. Od czasu wynalezienia oprogramowania rozwój technologiczny przyspieszył, a zmiany kulturowe zaczęły zachodzić szybciej, co może oznaczać, że obecnie ludzkość dopiero stoi u progu gwałtownej i całkowitej rekonfiguracji kultury. Kierunek i sposób przebiegu tych zmian jest zapisany w oprogramowaniu, które zgodnie z założeniami Geoffa Coxa, warunkuje przyszłość. Oprogramowanie w znaczący sposób kształtuje rzeczywistość, a przez to w sposób pośredni prowokuje powstawanie nowych dziedzin naukowych, zajmujących się jej analizą. Dlatego do zadań współczesnej humanistyki powinno należeć zwrócenie się ku badaniu *software'u*. Postęp technologiczny dokonuje się bardzo szybko, w znacznie większym tempie niż rozwój nauk teoretycznych, które biorą odpowiedzialność za analizę zjawisk kulturowych. Stąd też wynika potrzeba, aby szeroko rozumiana humanistyka także się zmieniała, aby móc efektywniej badać zachodzące obecnie procesy kulturowe. Działający w niej nurt *software studies* wprowadza w tym obszarze zmiany systemowe, takie jak otworzenie się na interdyscyplinarność czy propagowanie narzędzi typu *open source*.

Źródła cytowań

COX, GEOFF, ALEX McLEAN (2013), *Speaking Code. Coding as Aesthetic and Political Expression*, Cambridge, Mass.: The MIT Press.

- GALLOWAY, ALEXANDER (2004), *Protocol. How Control Exists after Decentralization*, Cambridge, Mass.: The MIT Press.
- GALLOWAY, ALEXANDER (2006), 'Protocol', *Theory, Culture, Society*: 2-3 (23), ss. 317-320.
- FILICIAK, MIROSŁAW (2008), 'Zajrzeć pod powierzchnię ekranu', *Kultura Popularna*: 4 (22), ss. 14-18.
- FULLER, MATTHEW (2003), *Behind the Blip. Essays on the Culture of Software*, Brooklyn, NY: Autonomedia.
- FULLER, MATTHEW (2008), *Software Studies. A Lexicon*, Cambridge, Mass.: The MIT Press. 2008
- FRY, BEN (2008), 'Deprocess', *Benfry.com*, online: <http://benfry.com/deprocess/> [dostęp: 14.07.2018].
- HUI KYONG CHUN, WENDY (2011), *Programmed Visions: Software and Memory*, Cambridge, Mass.: The MIT Press.
- KITCHIN, ROB, MARTIN DODGE (2011), *Code/Space. Software and Everyday Life*, Cambridge, Mass.: The MIT Press.
- KITTLER, FRIEDRICH (1996), 'Technologies of Writing/Rewriting Technology', *New Literary History*: 27 (4), ss. 731-742.
- MANOVICH, LEV (2013), *Software Takes Command*, New York: Bloomsbury.
- MANOVICH, LEV (2006), *Język nowych mediów*, przekł. Piotr Cypriański, Warszawa: Wydawnictwa Akademickie i Profesjonalne Spółka z.o.o.
- OLIVER, JULIAN, DANJA VASILIEV (2014) 'PRISM: The Beacon Frame. Speculative NSA Forensics Equipment', *Criticalengineering.org*, online: <https://criticalengineering.org/projects/prism-the-beacon-frame/> [dostęp: 14.07.2018].
- OLIVER, JULIAN, DANJA VASILIEV, GORDAN SAVIČIĆ, (2011) 'The Critical Engineering Manifesto', *Criticalengineering.org*, online: <http://criticalengineering.org> [dostęp: 14.07.2018].
- WARDRIP-FRUIIN, NOAH (2009), *Expressive Processing. Digital Fictions, Computer Games and Software Studies*, Cambridge, Mass.: The MIT Press.