Grzegorz KONAT*

Strategiczne rozwidlanie w tworzeniu Wolnego i Otwartego Oprogramowania[1]

Wstęp

Pojawienie się Wolnego i Otwartego Oprogramowania (WOO) postawiło naukowcom wiele istotnych pytań z dziedziny ekonomii, prawa oraz technologii. Z punktu widzenia ekonomisty interesujące może być na przykład, jak fenomen WOO wpływa na rynek oprogramowania o zamkniętym kodzie źródłowym. Zaskakującym, ale niemniej ciekawym, okazał się fakt poświęcania przez programistów ich czasu, wiedzy i umiejętności na udział w projektach, które nie przynoszą im żadnych wymiernych korzyści finansowych. Zwłaszcza dziś – w dobie Internetu – aplikacje komputerowe pobierane bez ograniczeń i całkowicie za darmo, czyli stanowiące „niewyłączalne dobra publiczne" (ang. *non-excludable public goods*), fascynują, ponieważ takie dobra zwykle charakteryzuje niedostateczna podaż. Szczególnie gdy ich produkcja jest – jak w tym wypadku – nieodpłatna [Ghosh, 2002]. Weber proponuje dla tych problemów i kwestii związanych z WOO, które wymagają szerokiej ekonomicznej analizy i wyjaśnienia, podział na trzy grupy: „kluczowe mikroprzyczyny" (tj. motywacje kierujące osobami uczestniczącymi w projektach WOO), „ekonomiczną logikę" produkcji *open source*, oraz „społeczne i polityczne struktury", które prowadzą do właściwego i udanego rozwoju WOO w warunkach nakreślonych przez motywacje społeczności oraz „ekonomiczną logikę tworzenia oprogramowania" [Weber, 2000]. W niniejszej pracy zostanie zaprezentowana analiza „kluczowych mikroprzyczyn", a w szczególności zjawiska, które jest kluczem do rozstrzygnięcia występujących wśród badaczy wątpliwości dotyczących reputacji jako podstawowego bodźca do tworzenia WOO. Tym zjawiskiem jest strategiczne rozwidlanie[2].

---

* Autor jest pracownikiem Instytutu Badań Rynku, Konsumpcji i Koniunktur oraz doktorantem w Kolegium Zarządzania i Finansów Szkoły Głównej Handlowej w Warszawie. Uwagi, komentarze: gk31527@sgh.waw.pl. Artykuł wpłynął do redakcji w lipcu 2008 r.

1 Artykuł powstał na podstawie pracy magisterskiej pt. „Ekonomiczna analiza Wolnego i Otwartego Oprogramowania", obronionej w czerwcu 2008 r. w Szkole Głównej Handlowej, na kierunku Ekonomia i jest wynikiem badań prowadzonych w ramach Studenckiego Koła Naukowego Mikroekonomii Stosowanej SGH. Autor chciałby podziękować prof. dr. hab. J. Prokopowi za wiele cennych uwag, które korzystnie wpłynęły na poziom niniejszej pracy.

2 Termin „strategiczne rozwidlanie (rozwidlenie)" jest autorską propozycją tłumaczenia angielskiego pojęcia „strategic forking", które dotychczas nie posiadało swojego polskiego odpowiednika.

Definicja Wolnego i Otwartego Oprogramowania

Rozważania niniejsze należy rozpocząć od przybliżenia pewnych zasad procesu produkcji oprogramowania, a konkretnie od wyjaśnienia pojęcia kodu źródłowego programu. Kod źródłowy jest to sekwencja wpisanych przez programistę, powszechnie używanych w języku (zwykle jest to język angielski) słów, które składają się w zdania stanowiące komendy dla komputera. Polecenia te decydują o logicznej strukturze tworzonego w tym procesie programu. Programista, mając do dyspozycji kod źródłowy, może – oczywiście z uwzględnieniem potrzebnego na to czasu – ustalić dokładnie jak dany program działa. Co więcej, ma on możliwość – modyfikując treść kodu – dostosować aplikację do swoich potrzeb i preferencji, poprawić błędy w programie lub też użyć części tegoż kodu do stworzenia własnej, odrębnej aplikacji komputerowej [Johnson, 2001]. Warto zaznaczyć, iż z punktu widzenia użytkownika oprogramowania, kod źródłowy ma niewielkie, jeśli w ogóle jakiekolwiek znaczenie [Kieseppa, 2002].

Kiedy kod źródłowy aplikacji komputerowej jest dostępny, mówimy, iż mamy do czynienia z „otwartym kodem źródłowym" (ang. *open source*) [Johnson, 2001]. Jednak fundamentem tego, co w języku angielskim określa się ogólnie i zwyczajowo terminem *open source*, jest „wolny" (ang. *free*) kod źródłowy. Oznacza to, iż jest on udostępniany wraz z aplikacją każdemu, kto tylko decyduje się na jej użycie. „Wolność" w tym kontekście ma swoje ściśle określone zasady. Jest to swoboda uruchamiania programu w jakimkolwiek celu, wolność studiowania jego działania, adaptowania do własnych potrzeb oraz redystrybuowania jego kopii, wreszcie możliwość dowolnego wprowadzania do programu zmian i dzielenia się nimi z tzw. społecznością *open source*, z pożytkiem dla ogółu [Free Software Foundation, 1996]. Powyższe nie oznacza bynajmniej, iż *open source* musi koniecznie być darmowe[3]. Przeciwnie, oprogramowanie tego typu może być (i często jest) przedmiotem zwyczajnej wymiany handlowej, podobnie jak każde inne dobro na rynku [Weber, 2003].

Wyjaśnienia wymaga również współistnienie dwóch terminów (*open source* i *free software*) na określenie tego samego zjawiska. Najogólniej rzecz ujmując, pojęcia te mają różne konotacje ideologiczne, wynikające z odmiennych wizji stojących za nimi i promujących je organizacji. Mimo iż cel – polegający na ochronie swobodnego dzielenia się kodem źródłowym – jest wspólny, to podejście Inicjatywy Otwartych Źródeł (*Open Source Initiative* – OSI)[4] oraz

---

[3] Nieporozumienia w tej kwestii wynikają z prostego faktu, iż angielski termin *free software* może oznaczać zarówno „wolne", jak też „darmowe" oprogramowanie. Popularną propozycją usuwającą wątpliwości jest zastąpienie angielskiego słowa *free*, francuskim *libre*, co daje termin *software libre*, który w żaden sposób nie może zostać odczytany jako *software gratis*. Notabene w języku polskim również powstają nieporozumienia związane z interpretacją pojęcia „Wolne Oprogramowanie" – termin, przez osoby nieznające w dostatecznym stopniu tematu, bywa odczytywany jako „oprogramowanie niedziałające odpowiednio szybko".

[4] *Open Source Initiative* to organizacja zajmująca się promocją oprogramowania *open source*, założona w 1998 roku przez Bruce'a Perensa i Erica Raymonda (www.opensource.org).

Fundacji Wolnego Oprogramowania (*Free Software Foundation* – FSF)[5] jest odmienne, zwłaszcza jeśli chodzi o stosunek do oprogramowania o zamkniętym kodzie źródłowym, będącego komercyjnym „przeciwieństwem" *open source*. FSF twierdzi, iż oprogramowanie zamknięte jest „(...) aspołeczne, czyli nieetyczne, a w konsekwencji zwyczajnie złe (...) [Stallman, 2001]" i mówi o korzystaniu z wolnego oprogramowania w kategoriach wyboru moralnego (wartości, tj. wolność, itp.). OSI nie podziela tego punktu widzenia utrzymując, iż *open source* raczej dowiodło swojej wyższości na polu technologii (dotyczy to zwłaszcza efektywności) i to właśnie z przyczyn związanych z efektywnością, a nie wolnością, powinno się stosować ten model produkcji i rozpowszechniania oprogramowania. Różnica ta, poza wymiarem ideologicznym, ma także *stricte* praktyczne konsekwencje. OSI „dopuszcza" bowiem – w przeciwieństwie do FSF – schematy licencjonowania bardziej otwarte na wykorzystanie *open source* przez prywatne korporacje [Pianon, 2004].

Ze względu na zarysowane powyżej rozbieżności, stworzony został termin „Wolne i Otwarte Oprogramowanie" (WOO – ang. F/OSS: *Free/Open Source Software*), który abstrahuje od tych różnic i traktuje zarówno model tworzenia i dystrybucji oprogramowania, jak również cały ruch i społeczność z nim związane jako jedną, spójną całość. Pojęcie to jest szczególnie popularne i zasadne w pracach akademickich, dlatego również w niniejszej posłuży za podstawowe określenie, zamiennie z *open source*, używanym odtąd wyłącznie w znaczeniu całości zjawiska. Należy zaznaczyć, iż pod pojęciem „społeczność WOO" rozumie się „(...) [grupę – GK] uczestników, którzy biorą udział w tworzeniu i rozwijaniu oprogramowania dobrowolnie, bez jakiejkolwiek kompensaty pieniężnej [Kieseppa, 2002]".

Na koniec rozważań definicyjnych trzeba jeszcze powiedzieć wyraźnie, czym Wolne i Otwarte Oprogramowanie nie jest. W Internecie (choć nie tylko) można znaleźć bardzo dużą ilość oprogramowania typu *freeware* lub *shareware*. Pierwsze z nich to takie, z którego można korzystać bez opłat. Przeważnie wolno też rozpowszechniać je bez ograniczeń. *Shareware* z kolei, to swego rodzaju wersja próbna oprogramowania: użytkownik może korzystać z niego tylko przez pewien czas lub niedostępne są dla niego wszystkie funkcje programu, który bez ograniczeń udostępniany jest dopiero po zakupieniu [Kieseppa, 2002]. Jeden i drugi typ aplikacji – choć dostępne za darmo, co z punktu widzenia przeciętnego użytkownika komputera jest podstawowym i często jedynym wyznacznikiem oceny i kategoryzacji oprogramowania – nie należą do rodziny WOO, ponieważ nie spełniają fundamentalnego kryterium otwartości kodu źródłowego. Są to więc jedynie szczególne przypadki oprogramowania zamkniętego, wciąż będącego podstawowym modelem biznesowym w tradycyjnej branży produkcji oprogramowania.

---

[5] *Free Software Foundation* to założona w 1985 roku przez Richarda Stallmana instytucja sponsorująca Projekt GNU, zajmująca się tworzeniem, ochroną i promocją wolności użytkowania, kopiowania, modyfikowania i rozprowadzania programów komputerowych oraz obroną praw użytkowników Wolnego Oprogramowania (www.fsf.org).

Mikroekonomiczne aspekty produkcji Wolnego i Otwartego Oprogramowania

W języku ekonomii Wolne i Otwarte Oprogramowanie jest dobrem publicznym. Z definicji wynika, że jest ono dobrem „nierywalizacyjnym" (ang. *non-rival good*), czyli jego konsumpcja przez jednego konsumenta nie zmniejsza w żaden sposób „ilości" oprogramowania, która pozostaje dla innych. Co więcej, *open source* jest dobrem „niewykluczalnym" (ang. *non-excludable good*) w tym sensie, iż jego producent nie jest w stanie w żaden sposób narzucić, kto może z tego dobra korzystać, a kto nie. Zgodnie z teorią mikroekonomii „niewykluczalne" dobra publiczne powodują „niesprawności rynku" (ang. *market failures*), polegające na skłonności konsumentów do przeznaczania mniejszych środków na finansowanie takich dóbr, niż ich korzyści czerpane z istnienia tychże. To z kolei wypływa z przekonania, że zawsze znajdzie się ktoś płacący za takie dobra, i istnieje duża szansa „jazdy na gapę" [Gravelle, Rees, 2004]. Fakt, iż „niesprawności rynku" nie występują w wypadku WOO (programiści jak najbardziej chcą produkować, w dodatku nieodpłatnie), zdaje się więc przeczyć teorii. W rezultacie ekonomiści zajmujący się WOO z perspektywy mikroekonomii, skupili się w swych dociekaniach na problemie motywacji programistów. Efekty ich ustaleń można zasadniczo podzielić na dwie grupy czynników: wypływające z użyteczności czerpanej z samego stworzonego przez samego siebie oraz te, gdzie użyteczność pochodzi z innych źródeł.

Produkcja Wolnego i Otwartego Oprogramowani a popytowe podejście do innowacji

Zjawisko WOO – choć z punktu widzenia ekonomii zaskakujące – nie jest bezprecedensowe. Znane są przypadki występowania tzw. popytowego podejścia do innowacji (ang. *User-Driven Innovation*), czyli takiej sytuacji, w której nowa technologia zostaje rozwinięta przez zaawansowanych użytkowników (konsumentów), a nie producentów [Kieseppa, 2002]. To właśnie użytkownicy rozpoznają zapotrzebowanie, rozwiązują problem poprzez wynalazek lub innowację, następnie „budują prototyp" i dowodzą jego wartości w praktyce [von Hippel, 1988]. Zdaniem von Hippela istnieją dwa powody, dla których takie zjawisko ma miejsce. Po pierwsze, producenci nie są w stanie zidentyfikować potrzeb konsumentów tak dobrze, jak oni sami (wynika to zwłaszcza z wysokich kosztów transferu informacji). Po drugie, nawet gdyby powód pierwszy nie zachodził, producentom brakowałoby odpowiednich bodźców, aby wyjść naprzeciw oczekiwaniom użytkowników w każdej kwestii, ponieważ ich produkty zwykle stanowią kompromisy pomiędzy oczekiwaniami jak największej rzeszy odbiorców [von Hippel, 1994]. Związany z tym jest także problem potrzeb i oczekiwań konsumentów (zwłaszcza użytkowników oprogramowania), zmieniających się w czasie wraz z tym, jak zmieniają się oni sami, nabierając doświadczenia w procesie korzystania z danego dobra – w tym wypadku: programu komputerowego.

Powyższe rozważania stanowią teoretyczne podwaliny dla modelu prywatnego dostarczania dóbr publicznych, gdzie motywację – jak w przypadku popytowego podejścia do innowacji – stanowi użyteczność czerpana przez produkującego z samego tylko produktu (owocu swej pracy). Na przestrzeni lat powstało wiele takich modeli, jednak żaden nie znalazł zastosowania do analizy szczególnego przypadku, jakim jest Wolne i Otwarte Oprogramowanie, aż do czasu skonstruowania przez Johnsona modelu odpowiedniego dla *open source*. W jego pracy występuje skończona liczba jednostek, które korzystają z WOO, i które muszą podjąć decyzję, czy samodzielnie produkować oprogramowanie. Aplikacja powstaje, jeżeli chociaż jedna z nich decyduje się na produkowanie [Johnson, 2001].

Model Johnsona zarysowuje wiele ciekawych problemów i sugeruje kilka interesujących propozycji ich rozwiązań. Najistotniejszym zagadnieniem jest to, związane z wpływem liczby uczestników gry na prawdopodobieństwo, że aplikacja zostanie wyprodukowana. Wzrost ich liczby wywiera bowiem dwojaki wpływ na szanse, że jednostka zdecyduje się programować. Z jednej strony prawdopodobieństwo rośnie wprost proporcjonalnie do liczby uczestników, z drugiej jednak, wzrost liczby graczy powoduje spadek motywacji do programowania, gdyż każdy „czuje", że nawet bez własnego wkładu pracy uda mu się skorzystać ze wspólnego dobra na prawach „jeźdźca na gapę" [Kieseppa, 2002]. W tym wypadku jednak model Johnsona nie udziela odpowiedzi, który z tych efektów działa w większym stopniu.

W swojej pracy autor szuka także odpowiedzi na pytanie, dlaczego niektóre proste aplikacje *open source*, jak np. edytory tekstu, nie powstają, tymczasem dużo bardziej skomplikowane przedsięwzięcia – jak np. narzędzia sieciowe – cieszą się zainteresowaniem programistów i skupiają rzesze chętnych, aby je produkować. Zdaniem Johnsona problem da się wyjaśnić poprzez określenie korelacji pomiędzy wartością, jaką projekt może przynieść potencjalnemu uczestnikowi-producentowi, a kosztem, jaki taka produkcja dla niego stanowi. Prawdopodobieństwo, że aplikacja nie powstanie jest duże, gdy korelacja jest wysoka i pozytywna – w tym wypadku użytkownicy, którzy najbardziej skorzystaliby na powstaniu takiego produktu, to jednocześnie ci, dla których proces produkcji jest zadaniem najbardziej kosztownym. Z drugiej strony, gdy korelacja przyjmuje wartości ujemne, jest prawdopodobne, że aplikacja powstanie – tutaj użytkownicy najbardziej zainteresowani powstaniem programu to ci sami, którzy mogą go stworzyć z największą łatwością. Przekładając to na przykład WOO można powiedzieć, iż edytor tekstu to aplikacja najbardziej pożądana z punktu widzenia tych członków społeczności, którzy jednocześnie posiadają najmniejsze umiejętności, aby taki program stworzyć. Tymczasem narzędzia sieciowe są najbardziej użyteczne z punktu widzenia najbardziej zaawansowanych programistów [Johnson, 2001].

Motywy kierujące programistami są także – z oczywistych przyczyn – istotne dla ewaluacji jakości produktów WOO. Kuan sugeruje, iż jeżeli aplikacja została napisana dla potrzeb osobistych, jej jakość generalnie powinna być wyższa niż porównywalnego programu zamkniętego [Kuan, 2002]. Z drugiej strony

– jak zauważył Saint-Paul – programy WOO, pisane w wyniku szczególnego zapotrzebowania tworzących je jednostek, są generalnie niedostosowane do potrzeb innych konsumentów [Saint-Paul, 2002]. Pojawiają się także głosy, iż producenci zamkniętego oprogramowania mają jasne, czytelne i stałe bodźce do testowania swoich produktów, co zwiększa bezpieczeństwo, gdy tymczasem procedury sprawdzające w społecznościach *open source* są bardziej doraźne, przypadkowe i improwizowane, a przy okazji obowiązek testowania spada na użytkowników [Mundie, 2002]. Nie da się jednak nie powiązać tego typu argumentacji raczej z obawami producentów zamkniętego oprogramowania, związanymi z pojawieniem się i gwałtownym rozwojem WOO, niż z rzeczywistością.

Inne motywy produkcji Wolnego i Otwartego Oprogramowania

Podobnie jak w wypadku każdego innego dobra, w analizie ekonomii WOO producenci są postrzegani przez pryzmat ponoszonych kosztów i uzyskiwanych korzyści, które generuje proces produkcyjny. Racjonalny gracz ma maksymalizować użyteczność przy założeniu wystąpienia danych kosztów i korzyści. W przypadku WOO, kosztem produkcji intuicyjnie zdaje się być wyłącznie koszt czasu spędzonego przez programistę nad stworzeniem aplikacji. Innymi słowy jest to ekonomiczny koszt utraconych możliwości, który – poza wieloma innymi czynnikami – zależy od popytu na programistów na rynku pracy [Kieseppa, 2002].

Lancashire dowodzi powyższej tezy pokazując, jak w przeciągu dekady lat 90. punkt ciężkości produkcji WOO przesunął się z USA do Europy. Jego zdaniem jest to bezpośrednio związane z tym, iż w owym czasie nakłady inwestycyjne na technologie informacyjne *per capita* w USA znacznie przewyższyły te w Unii Europejskiej, co spowodowało duży popyt na wysoko wykwalifikowanych programistów na rynku amerykańskim. W konsekwencji również koszt alternatywny produkcji WOO zdecydowanie wzrósł w USA, powodując, że dla tamtejszych specjalistów zajmowanie się tego typu działalnością stało się zdecydowanie mniej opłacalne [Lancashire, 2001]. Pomimo atrakcyjności zaprezentowanej przez autora argumentacji, trzeba zauważyć, iż ma ona pewne słabości. W krajach rozwijających się koszt utraconych możliwości udziału w projektach WOO jest zdecydowanie niższy niż w Europie, tymczasem tamtejsi programiści nie produkują proporcjonalnie więcej WOO od swoich europejskich kolegów. W tym wypadku należałoby się być może zastanowić nad wpływem jeszcze jednego czynnika – szeroko rozumianej kultury technologicznej.

W omówionym wcześniej modelu Johnsona jedynym, co motywowało udział w produkcji WOO, były czynniki „osobiste". Tymczasem wydaje się, iż mogą występować także inne bodźce do tego typu działalności. W swojej pracy Levy przedyskutował rozwój i naturę subkultury hakerów i – jego zdaniem – narodzinom „sposobu życia" hakerów towarzyszy rozwój specyficznej „etyki hakerskiej", której zasadnicze poglądy i imperatywy można przedstawić w kilku punktach:

- dostępność komputerów powinna być całkowita i niczym nieograniczona,
- każda informacja powinna być wolna,
- należy promować decentralizację – nie należy ufać wszelkiego rodzaju władzom,
- hakerzy powinni być oceniani przez pryzmat swoich umiejętności, a nie kryteriów, takich jak: stanowisko, wiek, rasa czy pozycja społeczna,
- na komputerze można tworzyć sztukę i piękno,
- komputery mogą zmieniać życie na lepsze [Levy, 1984].

Nie ulega wątpliwości, iż przedstawione poglądy mogą służyć za motywację do współpracy w ramach społeczności WOO. Na wyższym stopniu ogólności, możemy się w tym kontekście posłużyć argumentacją Webera, który dzieli „psychologiczno-osobowościowe" motywy uczestniczenia w projektach WOO na cztery kategorie. Pierwsza to poglądy normatywne, zgodnie z którymi oprogramowanie o zamkniętym kodzie źródłowym jest nieetyczne. Druga to motywacje „artystyczne", wyrażające przekonanie, iż hakerzy pracują jak artyści oraz że motywuje ich „radość tworzenia, wyzwania i piękno", które widzą w swojej pracy. Jak jednak słusznie zauważa Weber, nie wyjaśnia to w żadnym wypadku, dlaczego programiści – w przeciwieństwie do zdecydowanej większości artystów – udostępniają swoje prace za darmo. Jako trzecia kategoria wspomniane jest przywiązanie do bycia w opozycji wobec faktycznego monopolisty na rynku systemów operacyjnych dla komputerów osobistych – firmy Microsoft. Ostatnim wreszcie motywem jest „nadmuchiwanie ego" (ang. *ego-boosting*) [Weber, 2000]. Raymond wyjaśnia to ostatnie pojęcie jako „(...) zaspokajanie własnego ego oraz [budowanie – GK] reputacji wśród innych hakerów [Raymond, 2001]".

Idąc podobnym do „nadmuchiwania ego" tropem, Lerner i Tirole podjęli próbę analizy motywów udziału w projekcie WOO przy pomocy bardziej „konwencjonalnych" pojęć ekonomicznych. Podzielili oni zysk netto, który uczestnik takiego projektu uzyskuje w wyniku uczestniczenia w nim, na wypłatę natychmiastową oraz wypłatę opóźnioną w czasie. Wypłata natychmiastowa, to różnica pomiędzy natychmiastową korzyścią, a natychmiastowym kosztem. Analogicznie, różnica pomiędzy opóźnionymi w czasie korzyścią i kosztem stanowi wypłatę opóźnioną w czasie [Lerner, Tirole, 2002].

Autorzy wyróżniają dwa typy korzyści natychmiastowych: korzyść usprawnienia pracy, polegającą na naprawieniu jakiegoś błędu w programie lub stworzeniu poprawki/ulepszenia, co powoduje przyspieszenie i usprawnienie wykonywanej pracy, oraz korzyść relaksu, wynikającą z założenia, iż tworzenie WOO jest traktowane jako swego rodzaju hobby, a czynność ta może być dla programisty przerwą w nudnych obowiązkach, co w rezultacie poprawia efektywność ich wykonywania. Natychmiastowy koszt z kolei, to wyłącznie koszt utraconych możliwości. Z jednej strony programista zawsze w spędzonym nad WOO czasie może podjąć pracę (co oznacza dodatkowy zarobek), a z drugiej zaangażowanie w projekt powoduje zwykle zarzucenie lub przynajmniej opóźnienie innych obowiązków jednostki [Lerner, Tirole, 2002].

Wspomniane wcześniej opóźnione w czasie korzyści związane są z tzw. bodźcami sygnalizacyjnymi. Lerner i Tirole identyfikują dwa takie bodźce. Pierwszy

to bodziec akceleracji kariery, czyli uznanie w świecie twórców, a być może także odbiorców WOO. Może to – zdaniem autorów – prowadzić do intratnych ofert pracy oraz innych gratyfikacji (tj. udziały w komercyjnych spółkach zajmujących się WOO lub dostęp do rynku *venture capital*). Drugi bodziec sygnalizacyjny to „czynnik próżności", czyli możliwość zaspokojenia potrzeby bycia rozpoznawalnym w środowisku. Istnieją trzy okoliczności pozytywnie wpływające na (wzmacniające) „bodźce sygnalizacyjne", które są tym większe:

- im bardziej zauważalny dla odpowiedniej „publiczności" jest efekt pracy włożonej w projekt WOO,
- im większy jest wpływ nakładu pracy na ww. zauważalność,
- im lepiej efekt pracy informuje o talencie i/lub umiejętnościach programisty [Lerner, Tirole, 2002].

Kluczowe w tym „rachunku" jest to, iż nie zidentyfikowano opóźnionych w czasie kosztów tworzenia WOO. W związku z występowaniem silnych korzyści opóźnionych w czasie oraz względną równowagą czynników natychmiastowych, w praktyce może to być decydujące dla zaangażowania jednostki w projekt.

Zdaniem Lernera i Tirole bardzo istotne są także korzyści ujawniające się w porównaniu z projektami oprogramowania o zamkniętym kodzie źródłowym. Korzyści natychmiastowe, to „efekt absolwenta" (ponieważ oprogramowanie jest darmowe, jest używane w szkołach oraz na uczelniach, więc – jeśli jest ono znane przyszłym programistom – obniża to koszty programowania) oraz korzyści związane z możliwością dostosowania środowiska pracy do indywidualnych potrzeb, ale przede wszystkim z prędkością i skutecznością reakcji na błędy w kodzie, co stanowi korzyść nie tylko dla programisty, ale także dla ewentualnie zatrudniającej go firmy. Korzyści opóźnione w czasie w porównaniu z zamkniętym oprogramowaniem ponownie związane są z „bodźcami sygnalizacyjnymi". To, w tym wypadku:

- większa możliwość oceny efektów pracy programisty, gdyż użytkownicy, ze względu na otwarty kod źródłowy, mogą zaobserwować indywidualny wkład danego dewelopera w projekt, a także wielkość zadania (np. ilość linii kodu) czy jego trudność,
- pełna inicjatywa, czyli np. bycie „jedynym własnym przełożonym" i realizacja własnych pomysłów, w swoim tempie,
- większa wszechstronność i wiedza programistów WOO, która może być wykorzystywana w liczniejszych i bardziej różnorodnych projektach [Lerner, Tirole, 2002].

Odnosząc się do kwestii, jakie czynności przynoszą ze sobą możliwości sygnalizacyjne, Lerner i Tirole dochodzą do wniosków zbieżnych w dużej mierze z zaprezentowanymi w modelu Johnsona. Ich zdaniem czynności ważne z punktu widzenia użytkownika końcowego, jak stworzenie dokumentacji czy przyjaznego interfejsu użytkownika, nie pociągają za sobą bodźców sygnalizacyjnych, a można oczekiwać, iż zadania takie nie zostaną wypełnione. Jako empiryczny dowód swoich konkluzji autorzy wspominają fakt, iż użytkownicy generalnie czerpią korzyści z uczestniczenia w wielu projektach naraz. Dalej dowodzą, że wspominanie autora projektu jest kluczowe w projektach WOO,

a także, iż korzyści związane z reputacją, jakie przynosi uczestniczenie w przedsięwzięciach *open source*, wydaje się mieć rzeczywisty wpływ na uczestniczących [Lerner, Tirole, 2002].

Z argumentacją Lernera i Tirole, zaprezentowaną w kilku powyższych akapitach, zdaje się nie zgadzać wielu innych autorów, a pośród nich także Weber, który twierdzi, iż udział w projektach WOO może prowadzić nie tylko do „nadmuchiwania ego", ale także do „kruszenia ego" (ang. *ego-damage*), gdyż propozycje danego programisty mogą zostać przez społeczność lub liderów projektu odrzucone. Kluczową jednak krytyką ze strony Webera argumentacji opartej na reputacji jest uwaga, iż taka argumentacja nie wyjaśnia w żaden sposób efektywnej kooperacji w ramach projektów WOO. Jego zdaniem, gdyby kluczowym bodźcem do udziału w przedsięwzięciach *open source* były faktycznie kwestie związane z reputacją, projekty te doświadczałyby zdecydowanie większej niż to w rzeczywistości ma miejsce konkurencji, np. o pozycję lidera. Taka konkurencja mogłaby przyjmować postać bezpośredniego „ataku" na pozycję kierującego projektem lub strategicznego rozwidlania. Tymczasem, jak twierdzi Weber, żadne z tych zjawisk nie zachodzi [Weber, 2000].

Sprzeczność między mocno osadzoną w ortodoksji ekonomii argumentacją Lernera i Tirole, a argumentami Webera (i wielu innych) ogniskuje się w ustaleniu, czy wspominane zjawisko strategicznego rozwidlania zachodzi, co mogłoby stać się dowodem potwierdzającym lub obalającym tezę o reputacji jako kluczowym czynniku motywującym programistów do działania w ramach WOO.

Charakterystyka i analiza zjawiska strategicznego rozwidlania

Definicja strategicznego rozwidlania

Podstawową i wyczerpującą definicję strategicznego rozwidlania, która zostanie wykorzystana w dalszej analizie, przedstawia Stenborg stwierdzając, iż: „(...) rozwidlanie pojawia się, gdy jednostka 'bierze' podstawowy kod, włącza do niego swoje moduły, ustanawia pakiet jako 'nowy' projekt WOO, wreszcie zaprasza innych, aby przyłączyli się do tego projektu. Strategiczne rozwidlenie ma miejsce nie ze względów technicznych, lecz wyłącznie w celu stworzenia nowego projektu, na którego czele 'rozwidlający' mógłby stanąć [Stenborg, 2004]". Rossi – za Weberem – nazywa rozwidlanie „(...) odstępstwem od głównej ścieżki rozwoju, motywowanym ambicją przewodzenia", które „(...) powoduje wydanie różnych i konkurencyjnych wersji kodu źródłowego, mających zatem tendencję do ewoluowania w wielu, często niekompatybilnych kierunkach [Rossi, 2004]." Jak słusznie zauważają Baldwin i Clark, rozwidlanie tworzy dwa lub więcej kodów, tam gdzie jest zapotrzebowanie tylko na jeden, a przez to proces ten w pewnym sensie podcina korzyści ekonomiczne z kolektywnego działania [Baldwin, Clark, 2003]. Lanzara i Morner zwracają uwagę, iż zdarza się, że nowy projekt podejmuje bezpośrednią konkurencję z przedsięwzięciem, z którego wyewoluował [Lanzara, Morner, 2003]. McGowan dość odważnie, choć nie bez racji, próbuje wyjaśnić zjawisko strategicznego rozwidlania za

pomocą narzędzi klasycznej teorii przedsiębiorstwa, pisząc, iż „z perspektywy prawnej możemy scharakteryzować rozwidlanie jako (...) metodę wchodzenia w posiadanie – wrogie przejęcie [McGowan, 2001]".

Warunki determinujące wystąpienie zjawiska

W powszechnym mniemaniu reputacja i status mają znaczenie tylko we wspólnocie „równych sobie". Jest tak w głównej mierze dlatego, ponieważ są one (a także np. fakt przewodzenia grupie) tzw. dobrami „pozycyjnymi" (ang. *positional goods*) [Hirsch, 1976]. Powoduje to, iż zagregowana konsumpcja netto statusu jest w zasadzie równa zero, więc w konsekwencji satysfakcja jednego gracza jest zależna od tego, czy pozostali „konsumują" takie dobro w dodatnich ilościach [Stenborg, 2004]. Jest to tym samym warunek absolutnie konieczny, aby mówić o strategicznym rozwidlaniu lub jego braku.

Podstawowym uwarunkowaniem wystąpienia strategicznego rozwidlenia jest pozwalająca na to licencja. Bez ram prawnych, które dopuszczają taką możliwość, zjawisko to (wówczas z pewnością na zdecydowanie mniejszą skalę) nawet jeśli nie zostałoby wyeliminowane zupełnie, byłoby nielegalne. Zauważa to McGowan, pisząc: „(...) licencje *open source* dają możliwość wystąpienia rozwidlenia, które oczywiście [w związku z tym – GK] okazjonalnie ma miejsce [McGowan, 2001]." Rossi natomiast stwierdza wprost, iż biorąc pod uwagę warunki licencji WOO – które praktycznie pozwalają każdemu na modyfikowanie i redystrybuowanie ogólnodostępnego kodu źródłowego – zaskakującym jest fakt, iż większość projektów WOO nie uległa rozwidleniu [Rossi, 2004].

W swoim eseju Raymond podaje przykład systemów operacyjnych Linux i BSD, z których pierwszy zasadniczo nie doświadczył rozwidlania, tymczasem BSD dotknęło to co najmniej trzykrotnie. Jego zdaniem warunkiem wystąpienia (lub niewystąpienia) strategicznego rozwidlenia jest struktura organizacyjna społeczności tworzącej dany projekt WOO. Autor twierdzi, iż paradoksalnie to scentralizowana struktura projektu BSD, z jego jasno wyznaczoną hierarchią autorytetów i „zabezpieczeniami" przeciwko rozwidlaniu przyczyniły się do wystąpienia tego zjawiska, podczas gdy w zdecentralizowanej i amorficznej społeczności Linuksa nic takiego nie miało miejsca. „Wygląda na to, iż projekty najbardziej otwarte rozwojowo wykazują zasadniczo najmniejszą tendencję do rozwidlania [Raymond, 1999a]". Potwierdzają to rozważania Cowstona i Howisona, którzy doszli do wniosku, iż jeżeli projekty mają nie ulegać rozwidlaniu, muszą być, po pierwsze, otwarte na wpływ użytkowników, dalej powinny wykazywać się zdecentralizowaną strukturą, która zmniejsza możliwości kontroli, a zwiększa – indywidualnego wkładu [Crowston, Howison, 2004]. W tym samym kontekście Kogut i Metiu podają – obok Linuksa – przykład serwera Apache jako projektu, który oparł się próbom podziału właśnie ze względu na przeciwdziałające rozwidlaniu rozwiązania w strukturze organizacyjnej projektu [Kogut, Metiu, 2001]. Niejako w opozycji do poglądów ww. autorów pozostaje tymczasem stwierdzenie Kenwood, której zdaniem to brak „sformalizowanej struktury korporacyjnej" może prowadzić do fragmentacji kodu źródłowego

i/lub rozwidlania [Kenwood, 2001]. Ma ona jednak zapewne na myśli w większym stopniu warunki licencyjne powstawania projektów WOO, niż strukturę organizacyjną samych społeczności.

Ilkka Tuomi, również opierając się na przykładzie systemu Linux, zarysowuje jeszcze inny warunek wystąpienia strategicznego rozwidlania – rozmiar projektu, czyli liczbę osób w nim uczestniczących. W 1992, gdy Linux dopiero raczkował, ale świetnie się zapowiadał i był już postrzegany jako doskonała platforma dla realizacji wszelkich pomysłów, zaledwie kilka osób uczestniczyło w tym przedsięwzięciu. Zdaniem Tuomiego znamienne jest, iż mimo różnych zawirowań projekt ten w tak wczesnej fazie rozwoju nie uległ podziałowi [Tuomi, 2000]. Można zatem wysnuć wniosek, iż warunkiem wystąpienia strategicznego rozwidlania jest duża liczba uczestników projektu, a nawet zaryzykować twierdzenie, iż zagrożenie nim rośnie proporcjonalnie wraz z rozwojem społeczności. Nie tylko bowiem zwiększa to szansę na wystąpienie okoliczności lub bodźców skłaniających do takiej akcji. Mniejsza grupa pozwala – nawet wyłącznie za pośrednictwem Internetu – uniknąć anonimowości i wytworzyć swoiste więzi między uczestnikami, co, poprzez proste mechanizmy psychologiczne i społeczne, znacznie zmniejsza szanse na rozwidlenie.

Nie można w tymi miejscu zignorować argumentacji Webera, który w swych rozważaniach twierdzi coś dokładnie przeciwnego wobec postawionej powyżej tezy. Jego zdaniem, im większa społeczność tworząca projekt, a zwłaszcza im większa liczba deweloperów (programistów) pracujących w ramach takiego projektu, tym szanse na zajście strategicznego rozwidlenia mniejsze. Przyczyną takiej sytuacji miałby być fakt, iż inicjatorowi podziału byłoby trudno uzasadnić taki postępek. Weber przypomina, że wraz ze wzrostem liczebności grupy programistów zajmujących się konkretną aplikacją, coraz trudniej jest zebrać drugą (względnie homogeniczną) grupę dla kolejnego takiego projektu. W konsekwencji trudniej byłoby takiemu „rozłamowcowi" wytłumaczyć się ze swojej decyzji aspektami technicznymi [Weber, 2000]. Nie można jednak nie dostrzec w takim rozumowaniu pewnej dozy naiwności i „życzeniowego" toku myślenia autora.

W końcu na inny jeszcze warunek wystąpienia strategicznego rozwidlenia wskazuje Nuvolari. Jest nim brak legitymizacji „posiadaczy/właścicieli" projektu WOO. Jeżeli lider nie jest wiarygodny nie tylko jako organ decydujący o przyszłości i kierunkach rozwoju projektu oraz zarządzający poszczególnymi „wkładami" do realizowanego zadania, ale także nie wydaje się kompetentny jako osoba scalająca całe przedsięwzięcie i jego członków oraz potrafiąca rozwiązywać konflikty, może to być warunkiem rozwidlenia [Nuvolari, 2003]. Warto nadmienić, iż sytuacja ta nie dotyczy jedynie – jakby się mogło wydawać – projektów o scentralizowanej strukturze, gdyż nawet w tych zdecentralizowanych, gdzie brak wyraźnej dominacji liderów, muszą występować pewnego rodzaju koordynatorzy, których to rolę niejako automatycznie przejmują zwykle twórcy/inicjatorzy projektów, czyli właśnie wspomniani przez Nuvolariego „posiadacze/właściciele".

Możliwe bezpośrednie przyczyny wystąpienia strategicznego rozwidlania

Pomimo iż definicja strategicznego rozwidlania dość jasno precyzuje, że stojącą za tym zjawiskiem przyczyną jest szeroko rozumiana ambicja, to jednak – jak się wydaje – sama w sobie nie jest wystarczającym bodźcem do podjęcia tego typu działania. Potrzebny jest bezpośrednio działający czynnik czy właściwie „pretekst", który zapoczątkowałby proces decyzyjny o podjęciu się podzielenia projektu. Tym, co różni tę kategorię od wcześniej wspomnianych warunków wystąpienia rozwidlenia jest fakt, iż opisane w części B) okoliczności są, z punktu widzenia analizowanego zjawiska, egzogeniczne. Bezpośrednie przyczyny można tymczasem eliminować, zmniejszając tym samym znacznie ryzyko podziału[6].

Mogące wystąpić bezpośrednie przyczyny fragmentacji kodu źródłowego i podziału projektu Kenwood w swej pracy dzieli zasadniczo na dwa typy. Zdaniem autorki możliwe są sytuacje: gdy fragmentacja zachodzi z „dobrych" powodów (np. gdy osoba koordynująca projekt słabo wywiązuje się ze swoich zadań), oraz kiedy dochodzi do niej ze „złych" przyczyn (np. występuje konflikt osobisty/osobowościowy pomiędzy czołowymi deweloperami) [Kenwood, 2001].

Prawie wszyscy autorzy wskazują na zasygnalizowane przez Kenwood konflikty w społeczności programistów jako najczęstsze bezpośrednie przyczyny wystąpienia rozwidlenia. Baldwin i Clark twierdzą, iż szczególnie problematyczne są nierozwiązane spory [Baldwin, Clark, 2003], podczas gdy Nakakoji et al. argumentują, iż do podziału może dojść przede wszystkim w wyniku braku spójności wizji liderów projektu z potrzebami (oczekiwaniami) większości członków społeczności. Zaznaczają oni, iż kluczowym czynnikiem jest responsywność (a raczej jej brak) głównych deweloperów i ich wrażliwość na sygnały wysyłane przez grupę [Nakakoji et al., 2002].

„Dobrą" przyczyną może okazać się niewłaściwe, tj. nieudolne bądź sprzeczne z przyjętymi normami, prowadzenie projektu przez jego lidera/liderów. McGowan uważa, iż w takim wypadku zagrożenie strategicznym rozwidleniem działa jako narzędzie dyscyplinujące, a do samego podziału wcale nie musi dojść [McGowan, 2001]. Jak to ujął Raymond: „(...) uważa się za niezwykle istotne, aby 'ostatnia deska ratunku' [czyli strategiczne rozwidlenie – GK] mogła zaistnieć w wypadku niekompetencji lub opuszczenia projektu przez zarządzającego nim (...) [programistę – GK] [Raymond, 1999b]".

Do strategicznego rozwidlania mogą także prowadzić tzw. wrogie poprawki. Raymond definiuje je jako takie, których intencją jest skierowanie projektu w kierunku niezgodnym z wolą zarządzających przedsięwzięciem. Wprowadzenie wrogiej poprawki jego zdaniem oznacza *de facto* decyzję o podjęciu konkurencji z oryginalnym projektem i jest poważnym problemem, gdyż większa ilość takich zmian zwykle prowadzi w kierunku całkowitego podziału [Raymond, 1999a].

---

[6] Więcej na ten temat w części E.

Wreszcie, stając niejako w opozycji do definicji strategicznego rozwidlania, Tuomi stwierdza, iż bodźcem do takiej akcji może być konflikt na tle technicznym, narastający np. wokół implementacji i/lub funkcjonalności konkretnego modułu programu. Prowadzi to często – w jego mniemaniu – do rozwoju odrębnych wersji takiego modułu, a w konsekwencji do podziału [Tuomi, 2000].

Przykłady i konsekwencje wystąpienia

W swoim eseju Raymond – choć generalnie jego teza jest inna – przyznaje, że strategiczne rozwidlanie zdarza się wśród projektów WOO. Rossi podaje konkretne przykłady rozwidlania, których doświadczyły „bardzo popularne programy, takie jak program Berkeley Unix (chodzi o system operacyjny znany jako BSD – GK), Sendmail pod koniec lat 80. czy bardziej współcześnie – projekt GNU Emacs [Rossi, 2004]". Nakakoji et al. – uściślając, iż w wypadku projektu Emacs chodzi o wyewoluowanie programu XEmacs – dodają do tej listy projekt EGCS, będący „dzieckiem" kompilatora GCC [Nakakoji et al., 2002]. Kenwood tymczasem rozwija myśl o wspomnianym projekcie systemu operacyjnego z Berkeley. Twierdzi ona, iż najlepszymi przykładami rozwidlania są komercyjne i niekomercyjne implementacje systemu UNIX, takie jak odpowiednio SCO, Solaris, IRIX, HP-UX czy właśnie BSD, w swoich różnych wersjach (OpenBSD, NetBSD, BSDI, itp.) [Kenwood, 2001]. Varian i Shapiro dodają, iż „rozszczepianie"[7] UNIX-a zatrzymało rozwój tego systemu i uczyniło go bardziej wrażliwym na konkurencję ze strony MS Windows zarówno na rynku oprogramowania na stacje robocze, jak i na serwery. „Uważamy rozszczepianie to za jedno z kluczowych zagrożeń dla projektów *open source* na przestrzeni czasu" dodają, wprowadzając tym samym do dyskusji o konsekwencjach strategicznego rozwidlania [Varian, Shapiro, 2003]. Ich obaw zdają się nie potwierdzać jednak wyniki badań sondażowych przeprowadzonych przez Henkela i Tinsa, w których zaledwie 11,6% ankietowanych użytkowników systemu Linux uznało, jakoby rozwidlanie stanowiło zagrożenie dla projektów WOO [Henkel, Tins, 2004].

Jako główna i najpoważniejsza konsekwencja wystąpienia strategicznego rozwidlenia wskazywana jest utrata kompatybilności w wyniku rozwijania danego systemu/aplikacji w ramach więcej niż jednego projektu. Tuomi nazywa to utratą „synergii rozwojowej", gdyż w konsekwencji deweloperzy muszą wybrać jedną z wersji jako podstawę dla ich dalszej pracy [Tuomi, 2000]. Gacek, Lawrie i Arief jako pochodną wspomnianego rezultatu rozwidlania wymieniają fakt, iż taka sytuacja powstrzymuje część osób i firm potencjalnie zainteresowanych uczestnictwem w projektach WOO od przystąpienia do nich [Gacek, Lawrie, Arief, 2004]. Warto dodać, iż poza utratą kompatybilności różnych systemów występuje swego rodzaju zanik użyteczności czerpanej przez członków społeczności, ale zwłaszcza przez końcowych odbiorców. Potwierdza to Weber, kładąc szczególny nacisk na efekty sieciowe jako czynnik cementujący społeczność *open source*

[7] „Rozszczepianie" jest terminem, którego Varian i Shapiro używają w swej pracy na określenie zjawiska strategicznego rozwidlania.

[Weber, 2000]. Przykładem niech będzie sposób udzielania wsparcia nowym użytkownikom i pomocy w sytuacjach problematycznych. Ponieważ generalnie projekty WOO – w przeciwieństwie do produktów komercyjnych – nie posiadają żadnego oficjalnego wsparcia (tj. gwarancja, serwis przez telefon, itp.), za takie służą przeważnie dobrze rozwinięte fora internetowe konkretnych projektów lub programów. Gdy jednak następuje podział, z pierwotnego projektu odchodzi nie tylko grupa programujących (co, oczywiście, obniża jakość samego projektu i jest konsekwencją samą w sobie), ale także np. część doświadczonych użytkowników, którzy na szeroką skalę udzielali pomocy np. na forum programu. W konsekwencji zarówno „stare", jak i „nowe" forum są dużo mniej pomocne, zwłaszcza gdy programy przestają być kompatybilne i ich użytkownicy nie mogą w związku z tym korzystać wzajemnie z „konkurencyjnej" pomocy.

Warto w tym miejscu wspomnieć również o pojawiających się głosach, iż strategiczne rozwidlanie ma pozytywne konsekwencje. Dalle i David należą do autorów, którzy twierdzą, że dla programistów kluczowa jest reputacja. W ich opinii uczestnicy projektów WOO uzyskują więcej korzyści wypuszczając nowy kod, niż przyczyniając się do rozwoju istniejących projektów, a także pracując raczej nad programami we wczesnym niż późnym stadium rozwoju, oraz nad często wykorzystywanymi modułami (np. jądro [kernel] systemu operacyjnego) niż samymi aplikacjami[8]. Autorzy dochodzą w konkluzji do optymistycznego wniosku, iż jeżeli tylko praca nad coraz nowymi wersjami i modułami jest pożądana oraz bardziej korzystna z punktu widzenia społeczności, to należy związane ze wspomnianymi warunkami zjawiska, takie jak choćby strategiczne rozwidlanie, postrzegać jako pozytywne [Dalle, David, 2003].

Przyczyny niewystępowania strategicznego rozwidlania

Swoją szeroką argumentację w kwestii niewystępowania strategicznego rozwidlania Raymond rozpoczyna od stwierdzenia, iż – wbrew pozorom – społeczność WOO posiada cały zbiór niepisanych zasad, szczegółowo regulujących, kto może modyfikować program, jakie warunki muszą zostać spełnione, aby modyfikacja była możliwa, a zwłaszcza, kto ma prawo do redystrybuowania zmodyfikowanej wersji z powrotem w społeczności. Raymond określa te zasady mianem „tabu" i wymienia trzy zasadnicze z perspektywy niniejszej pracy:

- rozwidlania projektów nie mogą mieć miejsca, z wyjątkiem sytuacji naprawdę wyższej konieczności, czemu jednak musi towarzyszyć szerokie wyjaśnienie i uzasadnienie oraz zmiana nazwy projektu,
- wprowadzanie do projektu zmian bez współpracy z koordynatorami (poza sytuacjami wyższej konieczności) nie jest możliwe,
- Usuwanie nazwisk(a) z historii projektu, listy twórców lub współpracowników jest niedopuszczalne, chyba że zainteresowany sam wyraźnie o to poprosi [Raymond, 1999a].

---

[8] Porównaj: uprzednio przedstawiona argumentacja Johnsona.

Raymond znajduje szereg wyjaśnień, dlaczego obowiązują takie tabu, czyli pośrednio również dlaczego nie zachodzi strategiczne rozwidlanie. Pierwsze z nich ujmuje następująco: „(...) hakerzy generalnie powstrzymują się od rozwidlania (...), aby móc zakwestionować legalność takich samych działań wymierzonych w ich projekt [Raymond, 1999a]". Na obawie przed wzajemnością oparta jest także inna istotna przyczyna. Podzielenie projektu oznacza podważenie kompetencji osób ten projekt dotychczas tworzących, nawet jeżeli nigdy wcześniej żadne zarzuty wobec nich nie były podnoszone, a kwestie techniczne nie były bezpośrednim bodźcem do podziału. Ważnym czynnikiem jest także fakt, iż wystąpienie któregokolwiek z tabu szkodzi wszystkim członkom społeczności, zmniejszając w wypadku każdego z nich prawdopodobieństwo, że udział w tej „kulturze daru" zostanie wynagrodzony. Alternatywnym wyjaśnieniem przytaczanym w omawianym eseju jest to, iż uczestnicy projektów zwyczajnie (i abstrahując zupełnie od wszelkiego rodzaju formalnych (GPL), i nieformalnych (tabu) ograniczeń) uważają rozwijanie dwóch lub więcej podobnych projektów za stratę czasu oraz marnotrawstwo sił i możliwości [Raymond, 1999a].

Innym uzasadnieniem dla istnienia tabu oraz ich utrzymania się w ramach „etyki hakerskiej"[9] jest argument McGowana. Twierdzi on, że wspomniane przez Raymonda normy rozwinęły się i utrzymały, ponieważ społeczność *open source* chciała koniecznie odróżnić się od tradycyjnego, konkurencyjnego modelu produkcji oprogramowania. Dodaje, iż normy takie pozwalają skupić się twórcom WOO bardziej na programowaniu niż na indywidualnych korzyściach [McGowan, 2001]. Z jednej strony McGowan uznaje wyższość modelu tworzenia oprogramowania *open source* nad modelem zamkniętym stwierdzając, iż jest on bardziej efektywny. Sugerowałoby to jego powstanie w pewnym naturalnym procesie ewolucyjnym od modelu klasycznego do WOO. Z drugiej jednak strony przyczyny powstania, a zwłaszcza rozwoju takiego modelu produkcji oprogramowania mogły być również – jak już wspominano wcześniej – ideologiczne[10].

Wracając do przyczyn występowania tabu, Feller i Fitzgerald podkreślają ich znaczenie w kontekście wspólnej pracy nad projektem grupy ludzi, którzy przeważnie nigdy nie mają okazji spotkać się osobiście. Co jednak kluczowe, autorzy przeciwstawiają się właściwie całej dotychczasowej argumentacji dotyczącej zdefiniowanych przez Lernera i Tirole bodźców sygnalizacyjnych. Ich zdaniem strategiczne rozwidlenie nie stanowi potencjalnej szansy dla dewelopera na atrakcyjne zatrudnienie w wyniku zasygnalizowania rynkowi swojego istnienia oraz umiejętności programistycznych i zarządczych. Wprost przeciwnie, takie działanie może być źle postrzegane jako destrukcyjne i – w konsekwencji – stać się dla reputacji ryzykiem, a nie szansą [Feller, Fitzgerald, 2000]. Związany

---

[9] Termin „etyka hakerska" został szeroko wyjaśniony w przełomowej pracy autorstwa Pekki Himanena: *The Hacker Ethic and the Spirit of the Information Age*, Foreword by Linus Torvalds, afterword by Manuel Castells, Random House, 2001.

[10] Weber w swojej pracy przywołuje np. analogię pomiędzy społecznością WOO, a – opisywanymi np. przez Bronisława Malinowskiego w *Argonautach Zachodniego Pacyfiku* – „kulturami daru".

w tym argument prezentuje także Raymond, który dochodzi do wniosku, iż rozwidlanie projektów jest postrzegane jako złe, ponieważ „(...) wystawieni na ryzyko związane z ich reputacją członkowie projektów mogą kontrolować sytuację wyłącznie poprzez udział w obu projektach jednocześnie (po podziale)", co – jego zdaniem – jest w praktyce zbyt skomplikowane, rozpraszające i czasochłonne, aby dało się zrealizować [Raymond, 1999a].

Doceniając przedstawione przez Raymonda znaczenie norm dla faktu niewystępowania strategicznego rozwidlania, Weber w swej argumentacji zwraca szczególną uwagę na efekty sieciowe jako czynnik w tym kontekście kluczowy. Twierdzi on, iż jedną z podstaw funkcjonowania WOO jest to, że każdy „jadący na gapę" nie tylko nie stanowi szkody dla projektu, czy jest neutralny, ale wręcz przynosi korzyści. Zakładając odpowiednio dużą liczbę uczestników projektu, którzy jadącymi na gapę nie są, im więcej jadących na gapę, tym lepiej, gdyż każdy – nawet mimowolnie lub nieświadomie – wnosi coś do przedsięwzięcia (np. zauważa jakiś błąd w kodzie). Tymczasem koszt podzielenia się z nimi wspólnym dobrem (czyli programem) i tak zawsze wynosi zero. W związku z tym preferowane są jak największe wspólnoty, a zatem rozwidlanie jako działanie zmniejszające społeczności i ograniczające pozytywne efekty sieciowe, traci rację bytu [Weber, 2000].

Wreszcie, zdaniem Kenwood, przyczyną niewystępowania strategicznego rozwidlania może być brak bezpośrednich ekonomicznych bodźców do takiego działania, będący konsekwencją konstrukcji licencji GPL. Tym, co – zdaniem autorki – cementuje wiele projektów, jest obawa przed konkurującymi z *open source* firmami, przyjmującymi strategię „anty-WOO" [Kenwood, 2001]. Bodźce ekonomiczne i obawy to także jeden z argumentów cytowanego już wielokrotnie Raymonda, który – opierając się na badaniach wskazujących, że tylko niewielki procent „potomstwa" strategicznego rozwidlania w dłuższym okresie pozostaje na rynku ze znaczącym udziałem – dochodzi również do wniosku, iż do współpracy w ramach jednego tylko projektu skłania niepewność co do losów po ewentualnym podziale. Zdaniem tego autora zwycięża obawa przed byciem członkiem „przegranego" lub „zapomnianego" przedsięwzięcia, co oznacza też zmarnowanie pracy wykonanej przez społeczność [Raymond, 1999a].

To, co z jednej strony jest warunkiem koniecznym do wystąpienia strategicznego rozwidlenia, może być także postrzegane jako element powstrzymujący przed takim działaniem. Licencja GPL to zdaniem Lernera i Tirole jeden z kluczowych elementów w zapobieganiu podziałom, chociaż ich argumentacja zdaje się być uzasadniona głównie dla specyficznego przypadku, jakim jest rozwidlanie inicjowane przez prywatne korporacje, które – będąc producentami oprogramowania – mają w takim podziale interes lub po prostu chciałyby zastrzec część, lub całość wolnego kodu [Lerner, Tirole, 2004]. Argument ten potwierdzają rozważania Maurera i Scotchmer, w opinii których system operacyjny Linux jest doskonałym przykładem na to, jak GPL funkcjonuje jako środek zapobiegający rozwidlaniu. Chociaż to deweloperzy decydują, jaki kod będzie nosił nazwę Linux, użytkownicy mają za sprawą licencji swobodę rozwijania wersji pod własnymi nazwami. Zdaniem autorów, z punktu widzenia

suwerenności „konsumenta", może to być rozwiązaniem nawet lepszym niż proste zakazywanie rozwidlania [Maurer, Scotchmer, 2006]. Opinię tę potwierdza Kenwood, stwierdzając, iż licencja GPL wyeliminowała (przynajmniej w przypadku Linuksa) ekonomiczne motywacje dla fragmentacji [Kenwood, 2001].

Alternatywne podejście do problemu strategicznego rozwidlania

W swojej pracy Nakakoji et al. prezentują alternatywne podejście do kwestii tworzenia i rozwoju WOO. Ich studium przypadku dzieli modele współpracy w ramach społeczności *open source* – a co za tym idzie modele rozwoju – na trzy typy: „zorientowane na poszukiwanie" (których zasadniczym celem jest poszukiwanie nowych rozwiązań i rozwój oprogramowania poprzez dzielenie się innowacjami – np. projekt GNU), „zorientowane na użyteczność" (których celem jest wypełnienie jakiejś luki/braku w funkcjonalności – np. system Linux, aczkolwiek bez jądra, które jest projektem „zorientowanym na poszukiwanie") oraz „zorientowane na służenie" (które są nastawione na dostarczenie wszystkim członkom społeczności WOO oprogramowania stabilnego i niezawodnego – np. serwer Apache). Autorzy twierdzą, iż o ile w wypadku oprogramowania „zorientowanego na poszukiwanie" rozwidlanie jest raczej rzadkie, to programy „zorientowane na użyteczność" doświadczają wielu podziałów, a nowe projekty są rozwijane przeważnie poprzez ponowne użycie i modyfikowanie istniejących, a nie zastępowanie starych programów całkowicie nowymi. Oczywiście, niesie to za sobą wszystkie uprzednio wspominane konsekwencje, jednak Nakakoji et al. twierdzą, iż nie jest to szkodliwe, ponieważ projekty o podobnej użyteczności konkurują ze sobą, a wersja, która zdobędzie największe wsparcie społeczności, ostatecznie wyeliminuje pozostałe. Model ten nazywają oni „stylem turniejowym" [Nakakoji et al., 2002].

Wątpliwości i zagadnienia do dalszej analizy

Poza ewidentnymi sprzecznościami i niedostatkami istniejących analiz strategicznego rozwidlania, zarysowują się również zupełnie nieporuszone w dotychczasowych pracach zagadnienia. Dla przykładu: jedynie Reagle w swoim eseju wspomina w kontekście rozwidlania o tzw. dystrybucjach, czyli zestawach programów rozpowszechnianych łącznie i dających po zainstalowaniu gotowy do użytku produkt, podając najpopularniejszy przykład dystrybucji – dystrybucję Linuksa [Reagle, 2003].

Dystrybucji Linuksa, będących *de facto* gotowymi do użycia systemami operacyjnymi, są setki i należy się zastanowić, czy powołanie do życia kolejnej – oczywiście na bazie którejś z już istniejących, zwłaszcza największych i najlepiej rozwiniętych – nie jest szczególnym przypadkiem strategicznego rozwidlania. Przede wszystkim trzeba zauważyć, iż bariery technologiczne podjęcia takiej akcji są zdecydowanie mniejsze – „skomponowanie" dystrybucji w oparciu o istniejącą, przy użyciu specjalnie do tego celu stworzonych narzędzi jest o wiele prostsze, mniej czaso- i pracochłonne, wymaga też zdecydowa-

nie mniejszej wiedzy i doświadczenia niż tradycyjne programowanie. To, co łączy tworzenie nowych dystrybucji z rozwidlaniem, to także takie same, jak te zidentyfikowane wcześniej, warunki wystąpienia (licencja GPL, struktura organizacyjna projektów) oraz potencjalne bodźce do odłączenia się nowej dystrybucji od istniejącej (konflikty, poczucie braku wpływu na rozwój przedsięwzięcia, odmienne wizje tegoż rozwoju lub inne oczekiwania co do funkcjonalności projektu). Ważna jest również możliwość zidentyfikowania takich samych lub podobnych, jak w wypadku „zwykłego" dzielenia kodu, skutków: utraty kompatybilności dystrybucji, zmniejszenia oddziaływania pozytywnych efektów sieciowych, a co z tym związane – spadku jakości funkcjonowania społeczności (mniej wykrywanych błędów, słabsze wsparcie na forach, itd.).

Tym, co różni opisywane zjawisko od definicyjnego strategicznego rozwidlania jest jego nie do końca „programistyczny" charakter. Jak już wspomniano, w tym przypadku zdecydowanie mniej jest „pisania" kodu, a więcej różnego rodzaju zabiegów dodatkowych, jak stworzenie dla dystrybucji charakterystycznej oprawy graficznej czy ciekawej strony internetowej. Jednocześnie, biorąc pod uwagę bardzo dużą liczbę dystrybucji i ich – w wielu wypadkach – względne podobieństwo, należy sobie zadać pytanie, czy świadczy to o tym, iż *casus* ten nie da się sklasyfikować jako rozwidlanie czy może wprost przeciwnie, jest to jeden z niewielu dowodów (w przypadku słuszności podważający wiele fundamentalnych argumentów w tej kwestii) na to, że strategiczne rozwidlanie w społeczności *open source* zachodzi, a jedynie przyjmuje tę „łagodniejszą" formę.

Innym problemem, oczekującym na dalszą analizę, jest kwestia poprawek wprowadzanych do kodu. DiBona, Ockman i Stone podają przykład takich poprawek do jądra systemu Linux, które np. umożliwiają działanie tego systemu na komputerach z nawet najbardziej nietypowymi architekturami procesora, co bez tych usprawnień byłoby poza zasięgiem Linuksa. Zdaniem autorów takie poprawki mogą być rozpatrywane w kategoriach rozwidlania, chociaż zwykle nie są, gdyż jest to wąsko wyspecjalizowana nisza, która nie ma większego wpływu na całość społeczności danego projektu [DiBona, Ockman, Stone, 1999].

Podsumowując całość zawartych w niniejszym artykule rozważań, należy stwierdzić, iż o ile pojawiło się już sporo ciekawych prac z dziedziny ekonomiki Wolnego i Otwartego Oprogramowania, o tyle zjawisko strategicznego rozwidlania wciąż czeka na poważną i dogłębną analizę. Nie można nie wspomnieć, iż zawarta w niniejszej pracy próba, wykorzystująca praktycznie całą literaturę dotyczącą badanej kwestii, jest bodaj pierwszą na świecie podejmującą to zagadnienie jako temat sam w sobie. Głównie z tego względu praca ma charakter zdecydowanie deskryptywny, co pozostaje jej mankamentem. Być może w przyszłości uda się skonstruować sformalizowany model badanego fenomenu, co jednak w najmniejszym stopniu nie gwarantuje znalezienia odpowiedzi na nurtujące badaczy pytania.

Bibliografia

Baldwin C.Y., Clark K.B., [June 2003], *Does Code Architecture Mitigate Free Riding in the Open Source Development Model?*, working paper, Harvard Business School.

Crowston K., Howison J., [November 2004], *The social structure of Free and Open Source software*, Syracuse FLOSS research working paper.

Dalle J-M., David P.M., [2003], *The Allocation of Software Development Resources in "Open Source" Production Mode*, SIEPR Discussion Paper, No. 02-27.

DiBona C., Ockman S., Stone M., [1999], *Introduction to Open Sources: Voices from the Open Source Revolution*, [w:] DiBona C., Ockman S., Stone M. (red.), *Open Sources: Voices from the Open Source Revolution*, O'Reilly and Associates, Beijing.

Feller J., Fitzgerald B., [2000], *A Framework Analysis Of The Open Source Software Development Paradigm*, [w:] *International Conference on Information Systems, Association for Information Systems*, Atlanta, pp. 58-59.

Free Software Foundation, [1996], *The Free Software Definition*, Free Software Foundation Inc., Boston.

Gacek Ch., Lawrie T., Arief B., [January-February 2004], *The many meanings of Open Source*, IEEE, Vol. 21, pp. 34-40.

Ghosh R.A., [2002], *Clustering and Dependencies in Free/Open Source Software Development: Methodology and Preliminary Analysis*, A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21.

Gravelle H., Rees R., [2004], *Microeconomics*, Pearson Education, Harlow.

Henkel J., Tins M., [May 2004], *Munich/MIT Survey: Development of Embedded Linux*, Institute for Innovation Research, Technology Management and Entrepreneurship, University of Munich.

von Hippel E., [1988], *The Sources of Innovation*, Oxford University Press, New York.

von Hippel E., [April 1994], *'Sticky Information' and the Locus of Problem Solving: Implications for Innovation*, Management Science 40/4, pp. 429-439.

Hirsch F., [1976], *The Social Limits to Growth*, Routledge & Kegan Paul, London.

Johnson J. P., [May 2001], *Economics of Open Source Software*, Massachusetts Institute of Technology.

Kenwood C., [July 2001], *A Business Case Study of Open Source Software*, The MITRE Corporation.

Kieseppa I.A., [2002], *Open Source Software and Economics*, working paper, SoberIT, Helsinki University of Technology.

Kogut B., Metiu A., [2001], *Open-Source Software Development and Distributed Innovation*, „Oxford Review of Economic Policy", Vol. 17, No. 2, pp. 248-264.

Kuan J., [2002], *Open source Software as Lead User's Make of Buy Decision: A Study of Open and Closed Source Quality*, A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21.

Lancashire D., [2001], *Code, Culture and Cash: The Fading Altruism of Open Source Development*, First Monday 6.

Lanzara F.G., Morner M., [2003], *The Knowledge Ecology of Open-Source Software Projects*, Preliminary Draft presented at the 19th European Group of Organizational Studies (EGOS) Colloquium, Copenhagen, July 3-5.

Lerner J., Tirole J., [2002], *Some Simple Economics of Open Source*, „The Journal of Industrial Economics", No. 50, pp. 197-234.

Lerner J., Tirole J., [December 2004], *The Economics Of Technology Sharing: Open Source And Beyond*, Working Paper 10956, NBER Working Paper Series.

Levy S., [1984], *Hackers: Heroes of the Computer Revolution*, Penguin Books, New York.

Maurer S.M., Scotchmer S., [April 2006], *Open Source Software: The New Intellectual Property Paradigm*, Working Paper 12148, NBER Working Paper Series.
McGowan D., [2001], *Legal Implications Of Open-Source Software*, University of Illinois Law Review, No. 1, pp. 242-304.
Mundie C., [2002], *Security: Source Access and the Software Ecosystem*, A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21.
Nakakoji K., Yamamoto Y., Nishinaka Y., Kishida K., Ye Y., [2002], *Evolution Patterns of Open-Source Software Systems and Communities*, Proceedings of the International Workshop on Principles of Software Evolution.
Nuvolari A., [January 2003], *Open source software development: some historical perspectives*, Working Paper 03.01, Eindhoven Centre for Innovation Studies.
Pianon A., [February 2004], *Trade Secret Vs. Open Source: And The Winner Is...*, Erasmus Law and Economic Review, No. 1, pp. 47-75.
Raymond E.S., [1999a], H*omesteading the Noosphere*, First Monday 310.
Raymond E.S., [1999b], *The Magic Cauldron*, [w:] *The Cathedral and the Bazaar*, First Monday 33.
Raymond E.S., [2001], *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, Revised Edition, O'Reilly, Beijing.
Reagle Jr. J.M., [2003], *Socialization in Open Technical Communities*, working paper.
Rossi M.A., [April 2004], *Decoding the "Free/Open Source (F/OSS) SoftwarePuzzle" a survey of theoretical and empirical contributions*, Dipartimento di Economia Politica, Universita di Siena, Working Paper N. 424.
Saint-Paul G., [2002], *Growth Effects of Non-Proprietary Innovation*, A paper presented at the conference "Open Source Software: Economics, Law and Policy", Toulouse, France, June 20-21.
Stallman R.M., [2001], *The GNU project*.
Stenborg M., [2004], *Explaining Open Source*, The Researsch Institute of the Finnish Economy Discussion Papers.
Tuomi I., [May 2000], *Learning from Linux: Internet, Innovation, and the New Economy. Empirical and Descriptive Analysis of the Open Source Model*, KNEXUS Menlo Circle, Berkeley.
Varian H.R., Shapiro C., [December 2003] *Linux Adoption in the Public Sector: An Economic Analysis*, working paper.
Weber S., [June 2000], *The Political Economy of Open Source Software*, BRIE Working Paper 140.
Weber S., [December 2003], *Open Source Software in Developing Economies*, working paper, University of California, Berkeley.

STRATEGIC FORKING IN THE DEVELOPMENT OF FREE/OPEN-SOURCE SOFTWARE

Summary

The paper discusses a trend in the development of computer software known as "strategic forking." This trend is an intrinsic feature of today's "open-source community," according to Konat.

To begin with, the article defines the term "source code" in reference to software" and it also explains the terms "open source" and "free software." Moreover, it introduces the definition of Free/Open-Source Software (FOSS). In the following part of the article,

Konat offers a microeconomic analysis of "strategic forking" to determine the motives guiding software engineers taking part in FOSS projects. The problem is discussed from the perspective of the theory of public goods, the demand-side approach to innovation, and other theories concerned with issues such as "hackers' ethic" and "ego boosting." Konat pays special attention to describing the "strategic forking" phenomenon with the use of classical microeconomic and enterprise theory tools developed by researchers Jean Tirole and Josh Lerner.

The author follows up with a comprehensive analysis of strategic forking as a key to explaining a fundamental discrepancy in the assessment of the motives that guide programmers taking part in the development of open-source software. The analysis focuses on the definition of strategic forking, the conditions determining this trend and its direct causes. Konat also looks at the implications of strategic forking and the factors due to which this approach has not become more widespread around the world.

The article closes with a review of issues that require further analysis, according to Konat. These include the issue of software distribution.

Keywords: strategic forking, free/open-source software, software production branch, user-driven innovation, hackers' ethic, public goods, operating system