Projekty OPEN SOURCE

sposoby organizacji oraz źródła finansowania

- dlaczego firmy je finansują ?
- co motywuje uczestników projektu ?
- jakie struktury organizacyjne tam występują ?
- jakie można stosować modele biznesowe ?

Adam Walczak

Tytuł:
Projekty OPEN SOURCE – sposoby organizacji oraz źródła finansowania

Autor:
Adam Walczak

Oprogramowanie:
OpenOffice.org, GIMP, czcionka Liberation

Druk:
Wieland - www.wieland.com.pl

Copyright © Adam Walczak
Poznań 2011

Wszelkie prawa zastrzeżone

Wszelkich informacji udziela:
Adam Walczak
ul. Grunwaldzka 38a/2, 60-786 Poznań
tel. kom. 604 188 992
ja@adamwalczak.info

www.projektyopensource.pl/ksiazka

„Rozwój open source pogwałca niemal wszystkie teorie zarządzania”

Dr. Marietta Baba, Dziekan Kolegium Nauk Społecznych

Michigan State University

„Nie obchodzi mnie, jak dobrze prowadzisz swoją hi-tech firmę wytwarzającą zamknięte własnościowe oprogramowanie.

Możesz po prostu nie dać rady pchać własne technologie tak szybko do przodu, jak ludzie, którzy korzystają ze wspólnych standardów i mogą rozwijać technologie kolektywnie”

Bob Young, współzałożyciel korporacji Red Hat

Streszczenie

Niniejsza książka stanowi katalog praktyk zaobserwowanych w świecie projektów *open source* oraz *free software*. Początek ma za zadanie wprowadzić czytelnika w charakterystyczny żargon środowiska ludzi i przedsiębiorstw uczestniczących w tego typu przedsięwzięciach. W zwięzły sposób zostanie tu opisany proces kształtowania się wspomnianej społeczności na przestrzeni lat, występujące w niej wewnętrzne podziały, a wreszcie motywy kierujące jej uczestnikami. Następnie poprzez obserwację samych projektów wyłonimy różne metody współpracy społeczności wytwarzających otwarte oprogramowanie oraz sposób w jaki firmy czerpią korzyści z uczestnictwa w tych społecznościach. Mówiąc nieco bardziej precyzyjnie, będziemy opisywać struktury organizacyjne tego typu przedsięwzięć oraz budowane wokół nich modele biznesowe. Sprawdzimy także, w jaki sposób te dwa aspekty se sobą kolidują, jakie kombinacje występują najczęściej, a jakie się wykluczają. Na zakończenie sprawdzimy również, czy typ wytwarzanego oprogramowania ma wpływ na potencjalną efektywność projektu open source.

Dla kogo jest ta książka ?

Książka ta jest przeznaczona dla wszystkich, którzy mają kontakt ze światem otwartego oprogramowania i pragną zrozumieć rządzące nim reguły. Wiedza ta jest szczególnie przydatna dla programistów oraz menadżerów z branży IT, którzy rozważają zorganizowanie nowego lub przyłączenie się do istniejącego już otwartego projektu. Przedstawione w niniejszej książce struktury organizacyjne oraz sposoby licencjonowania doskonale obrazują korzyści, które mogą płynąć z tego sposobu wytwarzania oprogramowania, ale i z drugiej strony ostrzega ona przed związanymi z tym zagrożeniami. Informacje o modelach biznesowych oraz ogólnej kulturze panującej w społeczności *open source* i *free software* są dobrym elementarzem dla osób zajmujących się sprzedażą oraz promocją powiązanych z nimi produktów lub usług. W niniejszej książce ukazanych zostanie wiele niekoniecznie oczywistych możliwości czerpania zysku z oprogramowania, które to często pozostają niedostrzegane w firmach koncentrujących się w nazbyt dużym stopniu na sprzedaży tradycyjnych licencji. Ostatnią grupą odbiorców są szefowie firm oraz menadżerowie wyższego szczebla, dla których projekty open source mogą okazać się dobrymi narzędziami w osiąganiu celów strategicznych ich przedsiębiorstw. Wreszcie w książce tej przedstawione zostaną co ciekawsze ruchy większych graczy z branży IT, jak np. stymulowanie rozwoju otwartego oprogramowania w celu zwiększenia popytu na własne usługi, czy też podkopania dochodowych sektorów rynku opanowanych przez konkurenta.

Od autora

Na początku, Szanowny Czytelniku, pragnę Ci podziękować za zainteresowanie, którym uraczyłeś moją książkę. Włożyłem w nią mnóstwo pracy, poświęciłem dla niej wiele czasu i zainwestowałem ogrom własnej wiedzy, a wszystko po to, aby mogła ona trafić do szerszego grona odbiorców. Nawet nie wiesz jak bardzo cieszy mnie fakt, że trzymasz ją w swoich dłoniach. Pierwsze prace nad tym materiałem sięgają pewnego epizodu w moim życiu, związanego ze studiowaniem na Polsko-Japońskiej Wyższej Szkole Technik Komputerowych, gdzie mój promotor Pani dr inż. Ewa Stemposz zgodziła się bym, jako temat swojej pracy magisterskiej, obrał problematykę projektów *open source*. Było to dla mnie ogromnie wyzwanie, które to w ostatecznej fazie odsunęło mnie z życia zawodowego na ponad pół roku. Jednak w dniu obrony wiedziałem, że udało mi się stworzyć pracę unikalną, która wedle mej wiedzy wypełniła pewną istniejącą lukę w literaturze fachowej. Minęło trochę czasu i postanowiłem, że nie pozostawię tej pracy samej w sobie na uczelnianej półce. Dlatego zdecydowałem ostatecznie, że wydam ją samodzielnie. W związku z tym zagospodarowałem trochę czasu na dopracowanie materiału, samodzielnie zaprojektowałem okładkę, wybrałem również drukarnię i wynająłem redaktora. Postarałem się, aby koszt każdego egzemplarza był jak najmniejszy. Nie zamierzam zbić na tym tytule jakiegoś niewyobrażalnego majątku, pragnę jedynie pozyskać dodatkową motywację, ale i środki do dalszej rozbudowy poniższego materiału. Przyznam, że marzy mi się, by w przyszłości książka ta osiągnęła wręcz encyklopedyczny poziom. Mam również nadzieję, że w obecnej formie będzie dla Ciebie ciekawą lekturą, która pozwoli Ci zrewidować własne poglądy na temat świata *open source*. Dokładnie w taki sam sposób, w jaki pozwoliła je zrewidować i mi, w trakcie gdy ją przygotowywałem.

Patroni medialni

OSWorld.pl
Wiadomości z świata open source

4programmers.net
Programowanie w Delphi, C#, PHP, Java. Serwis zawiera materiały dydaktyczne dla programistów oraz niemal pół miliona postów na forum dyskusyjnym

ubuntu.pl
Polskie forum użytkowników Ubuntu Linux

linux.pl
Obszerny serwis poświęcony Linuksowi. Zawiera m.in. nowinki ze świata Open Source, poradniki, tłumaczenia.

sdjournal.pl
Miesięcznik dla programistów

di24.pl
Dziennik Internautów
internet w życiu i biznesie

Spis treści

1.Wstęp..11

2.Charakterystyka open source..15

2.1.Historia ruchu open source z biznesowego punktu widzenia............................16

2.2.Co oznacza free, a co open w free open source software ?...............................22

2.3.Co motywuje ludzi do angażowania się w projekty open source ?....................26

2.4.Typy licencji – poziomy wolności i ich wpływ na biznes.................................28

3.Struktury organizacyjne..37

3.1.Ogólna kultura organizacyjna...38

3.2.Społeczności z życzliwymi dyktatorami...51

3.3.Społeczności merytokratyczne...55

3.4.Proces wytwórczy wewnętrzny, sprzężenie zwrotne w społeczności................62

3.5.Proces społecznościowy oparty na specyfikacjach...65

3.6.Fork..73

3.7.Projekt parasolowy...77

3.8.Dystrybucja..78

4.Modele biznesowe..87

4.1.Ogólne praktyki..88

4.2.Podwójne licencjonowanie..90

4.3.Up-selling i Cross-selling...93

4.4.Zapewnianie powiązanych płatnych usług...98

4.5.Dotacje i inne formy uznaniowe..104

4.6.Cele biznesowe nie przynoszące bezpośredniego zysku........................108

5.Wpływ typu oprogramowania..115

6.Podsumowanie..127

Literatura..131

Dodatek A – badania ankietowe...135

Skorowidz..139

Notatki...145

Wstęp

Rozdział 1

Otwarte projekty są fenomenem, który wkroczył na arenę historii informatyki na przełomie lat osiemdziesiątych i dziewięćdziesiątych XX wieku, w postaci ruchu na rzecz wolnego oprogramowania (ang. *free software)*. W tym czasie biznesowi gracze branży IT postrzegali je jako fanaberię kilku członków środowiska akademickiego, która to nie ma racji bytu w warunkach komercyjnych. Wizja otwartych projektów, w ramach których oprogramowanie mogło być używane i kopiowane za darmo, a użytkownicy byliby traktowani jako współpracownicy, wydawała się nie mieć żadnych szans na opracowanie i udoskonalanie wartościowych produktów informatycznych. Nie do zaakceptowania wydawała się także idea otwarcia kodów źródłowych, które miałyby być publicznie dostępne oraz współdzielone z innymi otwartymi projektami. Mimo tego, społeczności zorganizowane wokół wolnego oprogramowania, wraz z upływem czasu, rosły w siłę i zaczęły produkować coraz większe ilości zaskakująco dobrej jakości oprogramowania. Dało to do myślenia menedżerom pracującym na rzecz zamkniętych, wewnątrzkorporacyjnych projektów. Z dnia na dzień na rynkach zdominowanych przez zamknięte oprogramowanie zaczynała się pojawiać konkurencja działająca w sposób niezrozumiały dla owych menedżerów: mimo że rozdawała swoje otwarte oprogramowanie zupełnie za darmo, znajdowała skuteczny sposób na dalsze finansowanie jego ciągłego rozwoju. Pojawiły się nawet głosy, które w sposób agresywny oskarżały konkurencję ze świata *open source* o dumping cenowy czy też psucie rynku. Sama organizacja pracy otwartych projektów również odbiegała w znacznym stopniu od tradycyjnych założeń i trudno było dostrzec to, co stanowiło podstawę dla jej efektywności.

Jeszcze większe zamieszanie, w przemyśle IT (pod koniec lat dziewięćdziesiątych), spowodowały debiuty firm, które zaczęły generować wielomilionowe zyski dzięki tworzeniu i finansowaniu projektów *open source*. Po bliższym przyjrzeniu się modelom biznesowym, na których oparta była działalność tego rodzaju, można dostrzec, że znaleziono wówczas inne sposoby na osiągnięcie zyskowności, aniżeli sprzedaż tradycyjnych licencji EULA[1] (ang. *end user license agreement*). Do grona takich praktyk należało opieranie się o świadczenie powiązanych płatnych usług lub tzw. podwójne licencjonowanie. Przy głębszej

1 EULA - zapis definiujący warunki udzielenia licencji końcowemu użytkownikowi. Zazwyczaj utożsamiany z warunkami licencyjnymi akceptowanymi podczas instalacji komercyjnego oprogramowania, które definiują z jakim zakresie możemy z owego oprogramowania korzystać (narzuca np. ograniczenie liczby stanowisk).

analizie organizacji pracy w projektach *open source* okazało się, że wiele elementów ich tworzenia zostało zaczerpnięte z nowopowstałych tzw. metodologii zręcznych (ang. *agile*). Zawierały one w sobie także wiele technik opisywanych w klasycznych pracach z dziedziny zarządzania przedsięwzięciami IT, które w korporacyjnych środowiskach były często postrzegane jako tzw. przerost inżynierii (ang. *overengineering*). Zrozumienie silnych stron otwartych projektów sprawiło, że również więksi gracze, tacy jak chociażby IBM czy też Sun Microsystems włączyli się do gry. Przejawami ich aktywności były na przykład „otwarcia" (w sensie uczynienia otwartym) niektórych większych produktów programistycznych oraz dotowanie wybranych fundacji wspierających rozwój otwartego oprogramowania. W dniu dzisiejszym projekty *open source* nie posiadają już wokół siebie takiej aury tajemniczości, jak na początku swojego istnienia i pewne podstawowe cechy, które przyczyniły się do ich sukcesu, są obecnie powszechnie znane.

Projekty *open source* nadal stanowią jednak dosyć istotny problem, gdy potrzebne jest dokonanie ich znacznie bardziej szczegółowej analizy lub też zaplanowanie stworzenia własnego przedsięwzięcia tego typu. Każdy menadżer lub programista, stający przed takim zadaniem bardzo szybko zacznie zadawać sobie pytania typu:

- Jakie formy organizacji otwartych projektów są powszechnie stosowane oraz które z nich wybrać?
- Jak budować społeczność wokół projektu i jak podejmować wraz z nią decyzje projektowe?
- Jak radzić sobie z ryzykiem „zbuntowania się" części społeczności poprzez utworzenie własnego projektu na bazie stworzonych w ramach pierwotnego przedsięwzięcia źródeł?
- Jak zachować wpływy i przywództwo w projekcie?
- Jaki model biznesowy wybrać, aby spieniężyć pracę włożoną w projekt *open source* jeżeli nie jest możliwa sprzedaż tradycyjnych licencji EULA?
- Dlaczego niektóre niszowe projekty, jak na przykład narzędzia programistyczne Qt, przynoszą krociowe zyski dla swoich twórców, a do niektórych popularnych produktów, takich jak pakiet biurowy OpenOffice, trzeba ciągle dopłacać?

Charakterystyka open source

Rozdział 2

2.1. Historia ruchu open source z biznesowego punktu widzenia

Powszechne praktyki lat 60' i 70'

Dzielenie się oprogramowaniem jest zjawiskiem tak starym, jak samo oprogramowanie. Jakkolwiek może się to wydawać nieprawdopodobne w dzisiejszym świecie, zdominowanym przez licencje EULA i liczne zabezpieczenia przeciw nieautoryzowanemu kopiowaniu, dzielenie się kodami źródłowymi było powszechną praktyką w latach sześćdziesiątych i siedemdziesiątych. W czasach tych oprogramowanie było wytwarzane i używane w głównej mierze przez naukowców oraz wysoko wykwalifikowanych techników. Wśród obu tych grup istniała stosunkowo otwarta kultura współdzielenia się własnością intelektualną oraz jej opiniowania. Wynikająca z tego wymiana wartości intelektualnych nie była oprawiana w żadną postać prawną, ani też limitowana postanowieniami licencyjnymi. Zresztą dostawcy sprzętu komputerowego mieli równie luźny stosunek do oprogramowania, gdyż nie postrzegali go jako istotnego aktywa biznesowego. Jak wyraźnie widać, branża IT w tamtych odległych o kilkadziesiąt lat czasach znacznie różniła się od obecnej. Jedną z kolejnych istotnych różnic była istniejąca wówczas duża liczba niestandaryzowanych architektur sprzętowych pozostających w powszechnym użyciu oraz brak języków programowania, które byłyby przenośne między tymi architekturami. Dlatego też dostawcy sprzętu komputerowego nie tylko nie ograniczali w żaden sposób współdzielenia się źródłami oprogramowania, a wręcz zachęcali do tego typu zachowań. Postawa taka wynikała z faktu, iż pomagało im to w dostosowywaniu oprogramowania do nowych architektur, a to z kolei zapewniało szybszą adaptację na rynku. Jednocześnie klienci tych dostawców wyrażali chęć do nawiązywania takiej współpracy, ponieważ tak czy inaczej konieczność zmuszała ich do pisania własnego oprogramowania, ściśle dopasowanego do wymogów prowadzonych przez nich badań. Zresztą moc obliczeniowa sama w sobie była wówczas znacznie ważniejsza od bogatego w opcje oprogramowania. Jednak wraz z dojrzewaniem tego przemysłu, w latach osiemdziesiątych zachodziły pewne zmiany, które zakończyły tzw. *pierwszą erę współtworzenia oprogramowania*[TES].

Eskalacja zamkniętego oprogramowania w latach 80'

Druga era, która rozpoczęła się w latach 80', jest określana mianem *eskalacja zamkniętego oprogramowania*. Jest to czas, w którym dostawcy sprzętu komputerowego zaczynają postrzegać oprogramowanie jako sprzedawalne dobro, towar sam w sobie podlegający wymianie handlowej, bądź chociażby w charakterze istotnego atrybutu przy sprzedaży sprzętu[POS]. Dwa najważniejsze czynniki, które doprowadziły do wspomnianej zmiany to powstanie wysokopoziomowych języków programowania oraz fakt, że rynek zaczął się skupiać na tylko kilku architekturach sprzętowych. Dzięki takiemu stanowi rzeczy wytwarzanie na szeroką skalę bogatych w opcje systemów informatycznych stało się opłacalnym przedsięwzięciem. Wytwórcy sprzętu zauważyli również, że wartość ich produktów, z perspektywy klienta, przesunęła się z oferowanej mocy obliczeniowej na dostępność oprogramowania. To z kolei sprawiło, że niektóre firmy zaczęły wymuszać przestrzegania własności intelektualnej w stosunku do produktów programistycznych, do których wcześniej wielu naukowców oraz techników ofiarowało własne poprawki. Jednym z najbardziej rozpoznawalnych przykładów tego typu był przypadek działań firmy AT&T wobec systemu operacyjnego UNIX[TES].

Praktyki te zerwały panujący cykl wymiany usprawnień w oprogramowaniu oraz obniżyły motywację społeczności IT do nieformalnego ofiarowania kolejnych poprawek. W trakcie, gdy trend ten rozprzestrzenił się, opanowując zdecydowanie większą część przemysłu komputerowego, kilka mniejszych społeczności (głównie powiązanych ze środowiskiem akademickim) skutecznie mu się opierało poprzez nie zamykanie swoich kodów źródłowych. Jedną z nich była grupa Computer Systems Research Group na Uniwersytecie Kalifornijskim Berkeley, która udostępniała swój pakiet oprogramowania Berkeley Software Distribution na zasadach bardzo liberalnej licencji BSD. Dzięki temu można było wykorzystywać ich kody źródłowe zarówno w projektach otwartych, jak i zamkniętych. Na początku Berkeley Software Distribution było zbiorem nowych programów oraz zamienników dla komercyjnego Sixth Edition Unix. Powoli jednak BSD zaczęło przeradzać się w niezależny system operacyjny. W międzyczasie niejaki Richard Stallman pojawił się na scenie branży IT jako jedna z najbardziej wpływowych oraz aktywnych osobowości opierających się trendowi zamkniętego oprogramowania. Rzucił on swoją pracę w laboratoriach AI w Massachusetts Institute of Technology, kiedy tylko przeszły one na zamknięte oprogramowanie i rozpoczął budowanie społeczności wokół projektu GNU oraz

fundacji Free Software Foundation[AGP]. Stallman przewodził obydwoma organizacjami budując prawne i ideologiczne podstawy nowej społeczności działającej na rzecz wolnego oprogramowania. Społeczności, która mogłaby dzielić się swoją pracą bez ryzyka, że ofiarowane przez nich kody źródłowe zostaną znów przekształcone w formę zamkniętego oprogramowania. Manifestacją tego było utworzenie licencji GNU GPL, która zezwalała na nieograniczoną dystrybucję niezmodyfikowanej wersji oprogramowania, ale jednocześnie nakładała pewne restrykcje jeżeli chodzi o dzieła powstałe z wykorzystaniem jego źródeł. Te restrykcje określały, że każde oprogramowanie, wykorzystujące źródła lub linkujące się z wolnym oprogramowaniem, musiało być również dostępne na zasadach licencji GPL. Ważnym efektem końcowym tego był fakt, że każde usprawnienie wykonane na wolnym oprogramowaniu stawało się automatycznie dostępne dla jego pierwotnych autorów. Projekt GNU Richarda Stallmana był pierwszym polem testowym dla zasad licencjonowania GPL. Budowała go społeczność z podobnymi celami jak ludzie związani z projektem BSD, jednak o wiele bardziej przesączona ideologią w swoich dążeniach. Dla członków zespołu BSD za otwartością projektu przemawiały względy praktyczne, podczas gdy dla ruchu wolnego oprogramowania była to kwestia natury moralnej. Ta różnica dała początek dwóm typom licencji *open source*: BSD-podobnym nierestrykcyjnym licencjom (nazywanymi także ang. *permissive licenses*) oraz restrykcyjnym licencjom zbliżonym do GPL (inaczej nazywanymi ang. *copyleft licenses*). Więcej informacji na temat licencji *open source* można znaleźć w rozdziale 2.4.

Jednocześnie w tych samych czasach zostały poczynione pierwsze próby czerpania zysku z wolnego oprogramowania. Jedną z nich było sprzedawanie kopii wolnego oprogramowania wykonywane przez Richarda Stallmana oraz FSF. Taki model biznesowy miał duży potencjał przed boomem internetowym i ewoluował później w sprzedaż dystrybucji oprogramowania dobieranego specjalnie pod konkretne zlecenie.

Innym przejawem rozwoju otwartego oprogramowania w tych czasach był projekt o nazwie *X Window System*. Stanowił on otwarte środowisko graficzne wykonane przez uczelnię MIT wraz z dostawcami sprzętu komputerowego, którzy byli zainteresowani dostarczaniem swoim klientom również systemu okienkowego. Projekt ten był jednakże daleki od idei sprzeciwu wobec zamkniętego oprogramowania, w końcu przecież i jego licencja X jawnie zezwalała na

wykonywanie zamkniętych rozszerzeń do tego środowiska. Każdy członek konsorcjum pragnął mieć możliwość dodania własnych modułów do domyślnego wariantu środowiska, aby móc zyskać przewagę nad pozostałymi członkami. X Window System został stworzony jako wolne oprogramowanie, a sam projekt stanowił prostą formę współpracy kilku dotychczas współzawodniczących ze sobą przedsiębiorstw nad wspólnym narzędziem[POS]. To pozwoliło im na drastyczne obniżenie kosztów wytworzenia produktu, który sam w sobie nie stanowił dla nich dobra podlegającego wymianie handlowej. Taka strategia pozwoliła wszystkim oszczędzić własne zasoby oraz rozproszyć ryzyko związane z projektem.

Poza przykładami wymienionymi powyżej, w tym samym czasie istniało wiele innych, mniej zauważalnych, otwartych projektów, posługujących się jeszcze innymi licencjami. Różnorodność otwartych licencji odzwierciedlała różnorodność istniejących motywacji. Wielu programistów, którzy decydowali się na wybranie licencji GPL dla swoich projektów, nie kierowało się w takim stopniu ideologią, jak chociażby osoby uczestniczące w projektach GNU. Niektórzy czuli moralny impuls, aby uwolnić świat od tzw. *software hoarding* (określenie Stallmana na nie-wolne oprogramowanie), ale inni byli bardziej motywowani poprzez techniczną fascynację i postrzegali metodologie stosowane w otwartych projektach jako efektywny sposób organizacji współpracy. Programiści mieli też inne powody, aby trzymać się razem na takich zasadach: wiele otwartych projektów wytwarzało bardzo dobrej jakości kod. Ta tendencja na pewno nie była czymś uniwersalnym, ale zarazem była coraz częściej spotykana. Nie mogło tego nie dostrzec i środowisko biznesowe. Menedżerowie wyższego szczebla nie zawsze są w pełni świadomi tego, co wykorzystują ich działy IT i często zdumieni dowiadują się z dużym opóźnieniem, że już używają wolnego oprogramowania w swoich codziennych operacjach. W związku z tym korporacje zaczęły obierać aktywną rolę w wolnych projektach poprzez oferowanie własnej siły roboczej lub sprzętu komputerowego, a czasem nawet poprzez finansowanie programistów. Takie inwestycje mogły, zgodnie z optymistycznymi scenariuszami, zwrócić się wielokrotnie w przyszłości. Mówiąc wprost: sponsor zatrudnia tylko małą liczbę pełnoetatowych ekspertów w projekcie, a czerpie korzyści z pracy ofiarowanej przez wszystkie podmioty w nim zaangażowane, w tym także przez pracujących bez finansowego wynagrodzenia wolontariuszy, jak i innych programistów opłacanych przez inne przedsiębiorstwa.

Rozwój internetu, masowej współpracy oraz open source od lat 90'

Jeżeli spojrzeć na wolne oprogramowanie jako metodologię, można zauważyć, że jest to łatwy sposób na stworzenie prostego środowiska do współpracy wszystkich podmiotów zainteresowanych projektem. Sposób, w którym ograniczamy biurokracje do absolutnego minimum. Otwarte projekty zazwyczaj chętnie akceptują wszystkich, którzy wyrażają chęć uczestniczenia w nich, oraz nie wymuszają, aby ich członkowie pracowali ciągle, tj. z dnia na dzień. Dzięki temu istnieje duża potencjalna grupa ludzi oraz przedsiębiorstw mogących dołączyć do tego typu projektów. Największą jednak barierą były tu początkowo ograniczenia geograficzne, które zanikły wraz z powstaniem i rozwojem internetu w latach dziewięćdziesiątych. Sieć sprowokowała społeczności zaangażowane w wolne oprogramowanie do stworzenia takich narzędzi, jak repozytoria plików, listy mailingowe, IRC chaty, trakery zadań, systemy kontroli wersji itp., a zarazem uczyniła je dostępnymi z każdego miejsca świata. To umożliwiło im w pełni wykorzystanie swojego potencjału i rozpoczęło trzecią erę, która trwa po dzień dzisiejszy[TES].

Narzędzia wymienione powyżej przyspieszyły rozwój sieci współpracy pomiędzy wieloma otwartymi projektami oraz inspirowały nowych twórców wolnego oprogramowania. Jednym z nich był Linus Torvalds, który wykorzystując narzędzia projektu GNU rozpoczął pracę nad projektem kernela Linux. Przy pomocy różnych programistów, którzy odnaleźli projekt w sieci, kernel Linusa dojrzewał w błyskawicznym tempie. Dwa lata od pierwszego wydania Linuxa w 1991 roku ekipa projektu GNU zdecydowała, aby użyć go do budowy swojego systemu operacyjnego. Stabilne jądro systemu było jedynym elementem jakiego im brakowało. Tym sposobem w roku 1993 światło dzienne ujrzała dystrybucja Debian, zwana także GNU/Linux, która była połączeniem kernela Linuxa i oprogramowania systemowego od GNU. Kombinacja ta okazała się wyjątkowo udana i zapewniła sobie rozpoznawalność w branży IT. W trakcie, gdy wolne oprogramowanie zyskiwało coraz szerszą akceptację na świecie, powstał nowy rynek odnoszący się do komercyjnego wsparcia dla tego typu oprogramowania oraz nowych dystrybucji GNU/Linuxa, po to by spełnić rosnące wymagania. Dwie z tych dystrybucji, które odniosły największy sukces, to SUSE oraz Red Hat, a firmy które je produkowały urosły do wielomilionowych spółek. Pierwsza z nich, obecnie pracująca pod grupą

Novell, niemal dominuje na rynku systemów operacyjnych dla superkomputerów[2]. Druga natomiast - Red Hat Inc., zyskała szeroką adaptację na rynku serwerów webowych i serwerów aplikacji. Te sukcesy pokazały przemysłowi IT, że wolne oprogramowanie może być źródłem znacznych dochodów.

Nowy trend traktował wolne oprogramowanie jedynie jako efektywny sposób wytwarzania i dystrybucji oprogramowania. Zmiana wizerunku z ideologicznego na bardziej praktyczny sprowadziła na scenę nowych aktorów, takich jak Larry Augustin, Jon Hall, Sam Ockman, Michael Tiemann i Eric S. Raymond, którzy zaczęli promować termin *open source*. Dzięki temu chciano odciąć się od ideologii negującej zamknięte oprogramowanie i promować aktywności o charakterze komercyjnym w ramach swojej społeczności. Bruce Perens i Eric S. Raymond utworzyli inicjatywę o nazwie Open Source Initiative, aby zapewnić merytoryczne i prawne wsparcie dla nowego ruchu. Wysiłek ten okazał się skuteczny, jako że sam termin *open source* zaczął zyskiwać na popularności, a na rynku pojawiły się nowe firmy czerpiące krociowe zyski z tego typu projektów. Jednym z najbardziej spektakularnych sukcesów była firma MySQL AB z ich low-endową bazą danych oraz Trolltech z zestawem narzędzi programistycznych Qt. Obie firmy wprowadziły nowy model biznesowy zwany *podwójnym licencjonowaniem* (patrz roz. 4.2), w którym za darmo udostępniali swoje produkty dla projektów *open source*, lecz jednocześnie wymagano opłat licencyjnych od tych, którzy chcieli je zintegrować z zamkniętym oprogramowaniem. Również najwięksi giganci branży IT dostrzegli zalety posiadania bogatego portfolio otwartych produktów. Od końca lat 90' można obserwować fale otwierania wielomilionowych produktów oraz tworzenia całych fundacji wspierających ruch open source. Przykładami takowych są Sun Microsystems, który otworzył niemal cały opracowany przez siebie stos technologiczny Java oraz wiele związanych z nim narzędzi. Obecnie wiele tych projektów jest aktywnie rozwijanych przez firmę Oracle po przejęciu w 2010 roku. Innym przykładem jest IBM, założyciel fundacji Eclipse Foundation mającej na celu rozwój otwartych narzędzi programistycznych. Swoje podejście zmienił także Microsoft, którego twórca Bill Gates w latach siedemdziesiątych agresywnie atakował ruch wolnego oprogramowania[3]. Korporacja z Redmond niedawno

2 Operating system Family share for 06/2009 - TOP500.org, http://www.top500.org/charts/list/33/osfam

3 Computer Hobbyists - Bill Gates, *Radio-Electronics* (New York NY: Gernsback Publications), 1976, http://www.swtpc.com/mholley/RadioElectronics/May1976/RE_May1976.htm

uruchomiła fundację CodePlex, zajmującą się rozwojem wolnego oprogramowania skoncentrowanego wokół swojego stosu technologicznego.

2.2. Co oznacza free, a co open w free open source software ?

Dla wielu osób zapoznających się ze światem otwartego oprogramowania mnogość zamiennie używanych terminów może być frustrująca. Możemy spotkać tutaj grupy programistów identyfikujących się z nazwami *open source*, ruch na rzecz wolnego oprogramowania (ang. *free software*) lub komercyjny (ang. *commercial-friendly* – dosłownie przyjazny dla komercji) *open source*. Mimo że na poziomie prawnym oraz technicznym społeczności te zorganizowane są w podobny sposób, wszystkie z nich mają wspólny zestaw zasad, których się trzymają, a do najważniejszych z nich należą[OSD]:

- **Swoboda redystrybucji** – od momentu, gdy dana osoba uzyska kopię oprogramowania ma pełne prawa, by je dalej kopiować i rozprowadzać bez potrzeby uiszczania dodatkowych opłat licencyjnych.
- **Otwarte źródła oprogramowania** – każdy rozprowadzający oprogramowanie *open source* musi rozprowadzać razem z nim jego kody źródłowe lub udostępnić je na szeroko dostępnym medium bez czerpania z tego korzyści materialnych.
- **Dzieła pochodne** – licencja musi zezwalać na modyfikację oprogramowania oraz na dystrybucję dzieł pochodnych bazujących na danym oprogramowaniu co najmniej na tych samych zasadach licencyjnych, na których było udostępnione to oprogramowanie.
- **Brak dyskryminacji** – oprogramowanie jest dostępne dla każdego, bez wyłączenia poszczególnych osób, grup czy też dziedzin zastosowania.
- **Dystrybucja licencji** – prawa, wraz z którymi rozprowadzane jest oprogramowanie muszą dotyczyć jego całości, bez wymagania stosowania dodatkowych licencji.

Zasada *swobodnej redystrybucji* jest często myląca ponieważ sprawia, że ludzie kojarzą otwarte oprogramowanie z oprogramowaniem freeware. Ta konfuzja jest częstokroć jeszcze większa, jeżeli zestawi się ze sobą terminy *free software* i *freeware*. Dlatego grupy, które określają się mianem grup *free software*, często spieszą z wyjaśnieniem, że[FSD]:

Free software (pl. wolne oprogramowanie) jest kwestią wolności, nie pieniędzy. Aby zrozumieć ten koncept należy postrzegać słowo free, jak free speech (pl. wolność słowa), a nie jako free beer (pl. darmowe piwo).

To oznacza, że *free software* jest czymś więcej niż oprogramowaniem, za którego kopię nie trzeba płacić. Tutaj uzyskuje się również dostęp do jego „wnętrza”, możliwość własnoręcznego wprowadzania modyfikacji odpowiednio do własnych potrzeb, a wreszcie prawo do wykorzystywania jego źródeł. Ale czy oznacza to, że ktoś mógłby pozyskać dany program typu *open source*, zmodyfikować go, a następnie zacząć sprzedawać jako zamknięte komercyjne oprogramowanie? Odpowiedź na to pytanie nie może być jednoznaczna: wszystko zależy od typu licencji *open source*, na której rozprowadzane jest dane oprogramowanie. Ogólnie licencje te dzielą się na nierestrykcyjne oraz na takie, które pozwalają na wykorzystanie źródeł tylko w innych otwartych projektach (patrz roz. 2.4).

Delikatne różnice między społecznościami programistów

Skoro terminy wymienione na początku rozdziału mają ze sobą tyle wspólnego, to czy możemy w ogóle traktować je rozłącznie? Czy w omawianych społecznościach mogą występować jakieś klarowne podziały? Dla osoby obserwującej to środowisko z boku mogłoby się wydawać, że nie. Przytoczmy tu jednak pewne wydarzenie, które miało miejsce w roku 1999 na konferencji LinuxWorld. Podczas tego wydarzenia Linus Torvalds, utożsamiany z ruchem *open source*, wręczał nagrodę Richardowi Stallmanowi. Stallman podczas przemówienia podsumował to poprzez żartobliwą analogię do sagi Gwiezdnych Wojen[4]:

Przekazanie nagrody Linus Torvalds Community Award dla fundacji Free Software Foundation to trochę jakby przekazać nagrodę Han Solo dla Floty Rebeliantów.

4 Revolution OS – film dokumentalny, 2001.

Free software jest terminem, który powstał jako pierwszy w latach osiemdziesiątych za sprawą Richarda Stallmana i jego fundacji Free Software Foundation (FSF). Ruch ten rósł jako opozycja w stosunku do nowych trendów w branży IT w czasach, które opisaliśmy w rozdziale 2.2. Jego zwolennicy uważają, że ważniejsze od posiadania potężnego oprogramowania jest tworzenie społeczności zaangażowanych w jego rozwój. Sam Richard Stallman postrzega oprogramowanie zamknięte wręcz jako amoralną koncepcję uniemożliwiającą użytkownikom dzielenie się poprawkami, które mogliby samodzielnie wykonać[AGP].

Jednak nie wszyscy członkowie ruchu wolnego oprogramowania widzieli to w ten sposób. Niektórzy uważali społecznościowe wytwarzanie oprogramowania zwyczajnie jako efektywną metodykę pracy. Takimi ludźmi byli Bruce Perens oraz Eric S. Raymond, którzy w późnych latach dziewięćdziesiątych stworzyli inicjatywę o nazwie Open Source Initiative[5] i rozpoczęli promocję terminu **open source**. Dzięki temu chcieli odciąć się od tej części ideologii, która negowała zamknięte oprogramowanie i skupić się w większym stopniu na aspektach technicznych oraz organizacyjnych. Wierzyli oni, że właśnie to jest źródłem spontaniczności rozwoju otwartych projektów.

Termin *open source* zyskał na popularności, jako że okazał się bardziej chwytliwy niż określenie *free software*. Prawdopodobnie stało się tak, poprzez brak mylącego podobieństwa do terminu *freeware*. Dzisiaj fraza *open source* jest używana przez większość ludzi jako sposób określania ogółu projektów i oprogramowania spełniających zasady wymienione na początku niniejszego rozdziału. Dlatego też wielu chcących się w widoczny sposób oddzielić od ideologii wolnego oprogramowania zaczęło stosować nazwę *komercyjny open source* (ang. *commercial open source* albo *commercial-friendly open source*).

Istnieją liczne projekty, których ideologie znajdują się gdzieś pomiędzy tymi dwoma grupami. Jednak dla łatwiejszego zrozumienia tych dwóch nurtów zestawimy w tabeli 2.2.A. zasadnicze różnice.

5 Oficjalna strona WWW Open Source Initiative, http://opensource.org/

Open source		
	Wolne oprogramowanie	**Komercyjne open source**
Ideologia	Moralny i społeczny wybór. Używanie zamkniętego oprogramowania jest określane jako niemoralne, jako że ogranicza swobodę do modyfikacji pozyskanej własności intelektualnej oraz dzielenia się poprawkami[AGP].	Techniczny i organizacyjny wybór. Metodyki oraz techniki stosowane w otwartym procesie wytwórczym pozwalają czasem bardziej efektywnie tworzyć oprogramowanie aniżeli w środowisku zamkniętym[CatB].
Preferowane licencje	Restrykcyjne licencje takie jak GNU GPL. Można korzystać z ich kodów źródłowych tylko jeżeli wytworzone oprogramowanie jest dostępne na tych samych zasadach[WhLG].	Mniej restrykcyjne, jak np. BSD, gdzie można wykorzystywać ich źródła w zamkniętych projektach lub tzw. podwójne licencjonowanie, gdzie np. jeden produkt jest dostępny zarówno na licencji GPL oraz na EULA dla zamkniętych projektów. (patrz roz. 4.2)
Inspirowane przez	Free Software Foundation, Richard Stallman	Open Source Initiative, firmy komercyjne
Przykłady	projekty GNU, Gimp	projekty Apache, MySQL

Tab. 2.2.A. Free software i open source

Więcej szczegółów odnośnie typów licencji oraz polityk kontroli praw autorskich zostanie omówionych w rozdziale 2.4.

2.3. Co motywuje ludzi do angażowania się w projekty open source ?

Jest wiele czynników motywujących przedsiębiorstwa oraz indywidualnych programistów do pracy w ramach projektów *open source*. Istnieje także mit, że większość ludzi biorących w nich udział jest wolontariuszami partycypującymi z powodów ideologicznych. Jednak, jak wykazały badania, owi woluntariusze stanowią średnio połowę programistów[GSH1], a w niemal każdym większym otwartym projekcie trzon grupy programistycznej jest opłacany przez jakiś powiązany podmiot. Również domniemana dominacja ideologi wśród wolontariuszy jest często znacznym uproszczeniem. Niektórzy twórcy udanych projektów *open source*, jak np. Marc Fleurys (autor serwera Jboss), wręcz uważają, że absolutnie nikt nie poświęca się w projektach *open source* pracując tam za darmo[6]. Każdy ma w tym jakiś interes.

Kilka z najczęstszych osobistych motywacji pośród indywidualnych programistów jest wymienionych poniżej:

- Radość tworzenia. Głównie dotyczy to studentów i hobbystów, jednak często odnosi się to również do programistów zawodowych. Jak wynika z ankiet[DA], 60% osób biorących aktywny udział w projektach *open source* i motywowanych tym czynnikiem pracuje również odpłatnie nad projektami zamkniętymi. Z obserwacji branży można również wysnuć wniosek, że zaskakująco często ludzie ci są wstanie wykonać bardzo stabilną i dobrze przemyślaną porcję kodu. Pierwsze wersje systemu operacyjnego Linux powstały, jak to określa Linus Trovalds, „dla zabawy", mimo że obecnie jest on zatrudniany przez Free Software Foundation, gdzie odpowiedzialny jest za rozwój projektu.

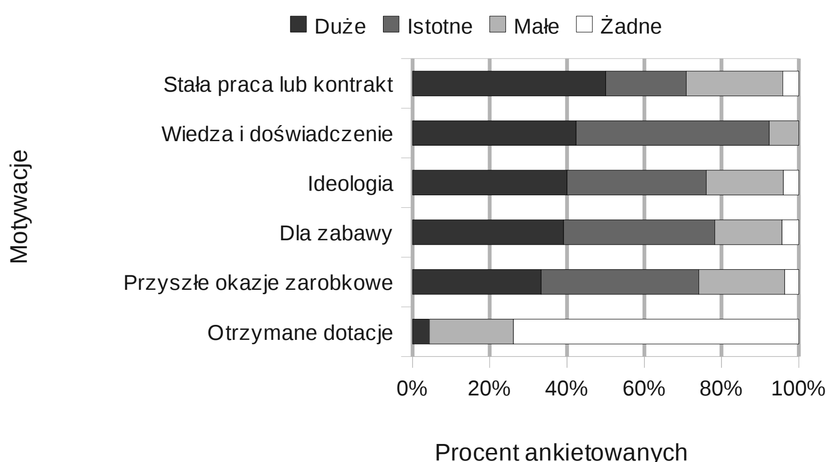
- Pozyskanie wiedzy i doświadczenia w ogólnie pojmowanym procesie wytwarzania oprogramowania.

- Pozyskanie specyficznej wiedzy projektowej, która może wspomóc przyszłe działania zawodowe. Często wiąże się to z chęcią zdobycia płatnych zleceń w zakresie danego produktu lub wytworzenia sobie marki konsultanta określanego jako tzw. specjalista produktu (patrz roz. 4.4).

6 „The Myth of Open Source", artykuł Bloomberg Businessweek, 08.07.2005, http://www.businessweek.com/technology/content/jul2005/tc2005078_5465_tc121.htm

- Stała praca lub płatny kontrakt obejmujący rozwój otwartego oprogramowania. Dotyczy to tych 50% wcześniej wspomnianych osób.
- Wiara w ideologię otwartego oprogramowania. Osoby utożsamiane z tym czynnikiem są często związane z ruchem na rzecz wolnego oprogramowania Richarda Stallmana. Mimo że ruch ten jest przeciwny zamkniętemu oprogramowaniu, z grupy osób ankietowanych[DA], która jest silnie motywowana przez ideologię, aż 47% bierze również udział w zamkniętych projektach. Nie można jednak odmówić wpływu, jaki ideologia, zarówno wolnego oprogramowania jak i programów *open source*, wywiera na programistów. Niemal wszyscy administratorzy tego typu projektów przebadani w ankietach[DA] uważają ten wpływ za znaczący, a aż 68% nawet za zasadniczy.
- Dotacje. Niektórzy programiści rozwijający małe projekty w pojedynkę, otrzymują drobne wsparcie materialne od swoich użytkowników.

Według ankiet[DA] prawie 35% szeroko pojętych informatyków oznajmiło, że wykonało mniejsze lub większe prace na rzecz projektów *open source*. Rozkład znaczenia powyżej wymienionych motywacji dla tej grupy przedstawia poniższe zestawienie (rys. 2.3.A).



Rys. 2.3.A. Motywacje programistów open source

Jak widać wszystkie wymienione motywacje, oprócz indywidualnych dotacji, pełnią tu dość porównywalną rolę. Nieco wyróżniająca wydaje się być podpisana umowa o pracę lub kontrakt, które angażują do pracy połowę ankietowanych. Za wynagrodzeniami tych programistów stoją rozmaite firmy oraz instytucje poświęcające swoje zasoby na rozwój otwartych projektów. Analizom motywacji tych podmiotów jest poświęcony cały rozdział 4. Ważny jest także podrozdział 4.6, który przedstawia wiele ukrytych w większym stopniu ról, jakie odgrywają projekty *open source*. Samo posiadanie sporej liczby użytkowników otwartego oprogramowania może zapewnić nieco ruchu wspierającego w pracach rozwojowych. Jak wykazują badania, 14-25% pracowników firm i instytucji wykorzystujących *open source* opracowuje w późniejszym czasie tak zwane łaty (ang. patch – tu słowo oznacza rodzaj dodatku do programu korygującego jego wady lub poprawiającego jego funkcjonalność) i oddaje je w ręce autorów projektu[7]. Natomiast według wyników ankiet[DA], 34% badanych osób oświadczyło, że wykonało mniejsze lub większe prace w otwartych projektach, a 9% określiło swój wkład jako istotny.

7 CIOInsight, CIOINSIGHT OSS survey, 2007.

2.4. Typy licencji – poziomy wolności i ich wpływ na biznes

Projekty *open source,* jak wszystkie inne projekty, podlegają systemowi prawnemu i rozpowszechniają swoje produkty razem z pewnego typu porozumieniami licencyjnymi. Powszechnie uznaje się, że licencje *open source* dają odbiorcom pełne prawa do swobodnego wykorzystywania, rozpowszechniania niezmodyfikowanych wersji oraz modyfikowania *software'u* według ich indywidualnych potrzeb. Jednakże istnieją znaczne różnice w wypadku integrowania otwartego oprogramowania z innym oprogramowaniem oraz w przypadku dystrybucji zmodyfikowanych kopii lub nowych produktów powstałych na ich bazie. Moment wyboru rodzaju licencji jest krytycznym dla programistów oraz menedżerów IT. Decyzja ta będzie miała istotny wpływ na strukturę organizacyjną przyszłego projektu oraz na model biznesowy firmy finansującej prace nad nim. Poprzez wybór licencji nie tylko określa się na jakich zasadach będzie możliwe wykorzystywanie oprogramowania, ale także precyzuje się czyje źródła będzie można wykorzystać w tak licencjonowanym produkcie. Poza tym poprzez obranie odpowiedniej polityki wobec praw autorskich będzie można odpowiedzieć na następujące pytania:

- Czy w trakcie rozwoju projektu razem ze społecznością *open source* zachowane zostaną prawa autorskie do całości kodów źródłowych?
- Czy owe prawa do tego oprogramowania, jako całości, będą powoli „rozpływały się” między programistami biorącymi udział w projekcie?

Kto określa, które licencje są open source?

Jak zostało już powiedziane w rozdziale 2.2, licencje muszą spełniać kilka podstawowych zasad, aby móc być nazwanymi licencjami *open source*. Zezwalanie jedynie na wgląd do kodów źródłowych to za mało. Inicjatywa Open Source Initiative jest jednym z najszerzej akceptowanych punktów odniesienia w tej materii i posiada bardzo precyzyjną definicję *open source*[OSD]. Bazuje ona na zasadach[8] stworzonych przez administratorów repozytoriów oprogramowania Debiana. Repozytoria te zawierają tysiące pakietów oprogramowania udostępnionych na dziesiątkach licencji,

8 Debian Free Software Guidelines - artykuł projektu Debian, 2004, http://www.debian.org/social_contract#guidelines

z których wszystkie muszą być zgodne z wizją wolnego oprogramowania. Dlatego też jeżeli licencja, którą wybrano, jest zgodna z zasadami Debiana lub zatwierdzona przez OSI to można być pewnym, że ma miejsce poruszanie się w ramach społeczności *open source*. Inicjatywa Open Source Initiative utrzymuje listę oficjalnie zatwierdzonych licencji[9], które są dobrym miejscem do rozpoczęcia poszukiwań tej odpowiedniej dla danego projektu.

Typy licencji

Ogólnie rzez biorąc licencje *open source* dzielą się na bardziej lub mniej restrykcyjne[TES]. Restrykcyjne licencje zezwalają na wykorzystanie kodów źródłowych tylko w takim oprogramowaniu, które rozprowadzane jest na zasadach kompatybilnej restrykcyjnej licencji *open source*. Chroni to otwarte źródła przed niepożądaną sprzedażą w postaci składników zamkniętych produktów. Wymusza to jednocześnie udostępnienie wykonanych modyfikacji społeczności *open source*, jeżeli strona modyfikująca chce rozprowadzać zmodyfikowane źródła. Mimo wszystko istnieje możliwość modyfikacji oprogramowania na restrykcyjnej licencji bez publikowania źródeł wprowadzonych modyfikacji. Należy jednak wtedy zachować warunek, że powstałe oprogramowanie jest wykorzystywane wyłącznie do użytku wewnętrznego i nie podlega dystrybucji. W takiej sytuacji wybór czy owe modyfikacje mają być otwarte czy zamknięte należy do strony modyfikującej.

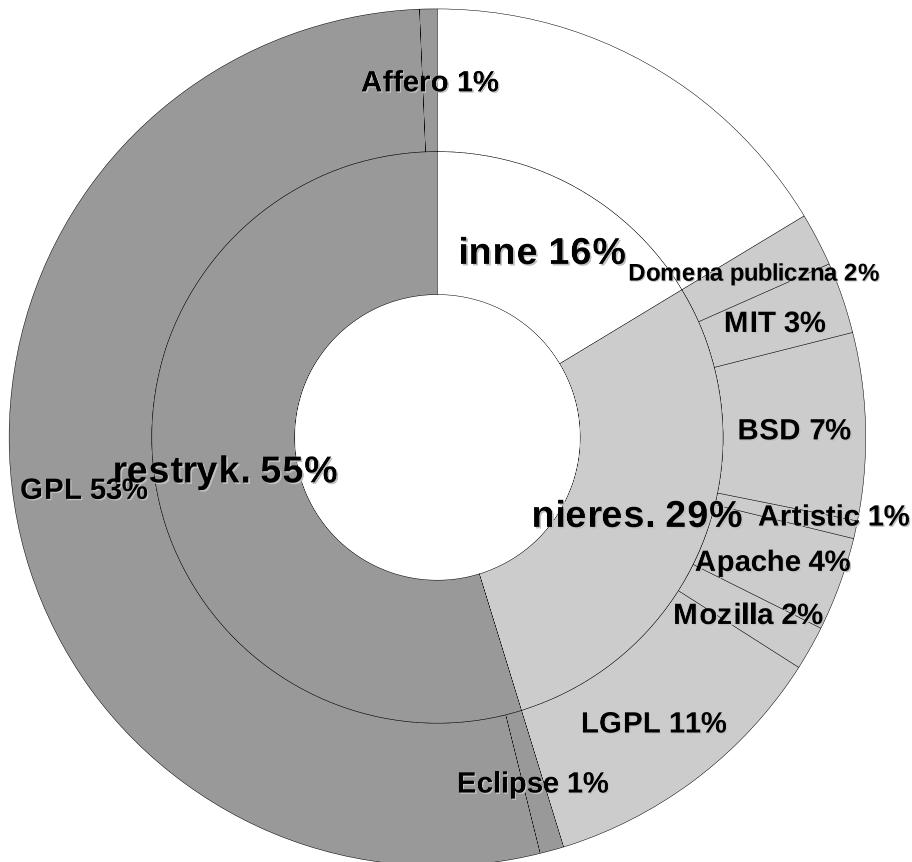
Restrykcyjne licencje są również nazywane angielskim określeniem *copyleft* (angielska gra słowna; chodzi tu o stworzenie słowa przeciwnego do *copyright*, czyli terminu oznaczającego prawa autorskie). Jedną z pierwszych oraz obecnie najpopularniejszych licencji tego typu jest GNU General Public License (w skrócie GPL). Innym typem licencji *open source* są tak zwane licencje nierestrykcyjne, określane często w języku angielskim nazwą *permissive licenses* (dosłownie licencje permisywne). Ich zasady są zbliżone w znacznym stopniu do koncepcji domeny publicznej (ang. *public* domain). Dwie z najstarszych i najszerzej stosowanych licencji tego typu to licencje LGPL oraz BSD. Zezwalają one na wykorzystanie źródeł tak licencjonowanego oprogramowania zarówno w projektach otwartych, jak i komercyjnych.

9 List of officially approved open source licenses - The Open Source Initiative, http://www.opensource.org/licenses

Mniej restrykcyjne licencje są zazwyczaj bardzo proste. Często są jedynie zezwoleniem na dowolne wykorzystanie kodów źródłowych z adnotacją, że autor nie ponosi żadnej odpowiedzialności za to oprogramowanie. Licencja MIT jest tego dobrym przykładem, jako że zajmuje tylko około 20 linii standardowego druku na kartce A4. Tego typu licencje są często preferowane dla projektów wytwarzających narzędzia programistyczne lub *middleware*, rozwijanych przez kilka przedsiębiorstw IT. Takie podejście pozwala szybko zorganizować bardzo liberalny i sterowany przez społeczność projekt. Często przyciąga to więcej firm oraz indywidualnych programistów, ponieważ owoce swojej pracy będą mogli wykorzystać zarówno we własnych otwartych, jak i w zamkniętych projektach. Innym ważnym aspektem jest brak wymogu transferu praw autorskich od osób i firm, które ofiarowały swoje poprawki. Można w przypadku tak licencjonowanego oprogramowania zezwolić, aby prawa autorskie do całości źródeł powoli rozpraszały się pomiędzy ludźmi, którzy włożyli w nie swoją pracę. Wszyscy uczestnicy tego typu projektów ofiarują swoje poprawki i usprawnienia na tej samej nierestrykcyjnej licencji, a także zezwalają na wykorzystywanie ich zarówno w zamkniętych, jak i otwartych przedsięwzięciach.

Jednak jeżeli przedsiębiorstwo, które otwierając dany projekt będzie chciało oprzeć swój model biznesowy na podwójnym licencjonowaniu (patrz roz. 4.2), to zachowanie przez nie pełnych praw autorskich będzie obowiązkowe. Tylko wtedy dane przedsiębiorstwo będzie mogło rozprowadzać produkty projektu na licencji *open source*, takiej jak GPL oraz typowej licencji komercyjnej EULA. Tego typu polityka wobec praw autorskich może być również wymagana przy tak zwanym up-sellingu zamkniętych produktów, które bazują na źródłach projektu otwartego (patrz roz. 4.3).

Popularność poszczególnych licencji wśród 28 000 najaktywniejszych projektów na SourceForge.net przedstawia się zgodnie z rysunkiem 2.4.A.



Rys.2.4.A. Popularność licencji

Kompatybilność licencji

Termin *kompatybilność licencji* odnosi się do problemu, w którym kilka produktów oprogramowania jest rozpowszechnianych na licencjach mogących zawierać wzajemnie wykluczające się wymagania. Dla przykładu załóżmy, że mamy dwie licencje A i B, w których znajdują się następujące nakazy:

A) zmodyfikowane wersje muszą wspominać o autorach oprogramowania w swoich materiałach reklamowych;

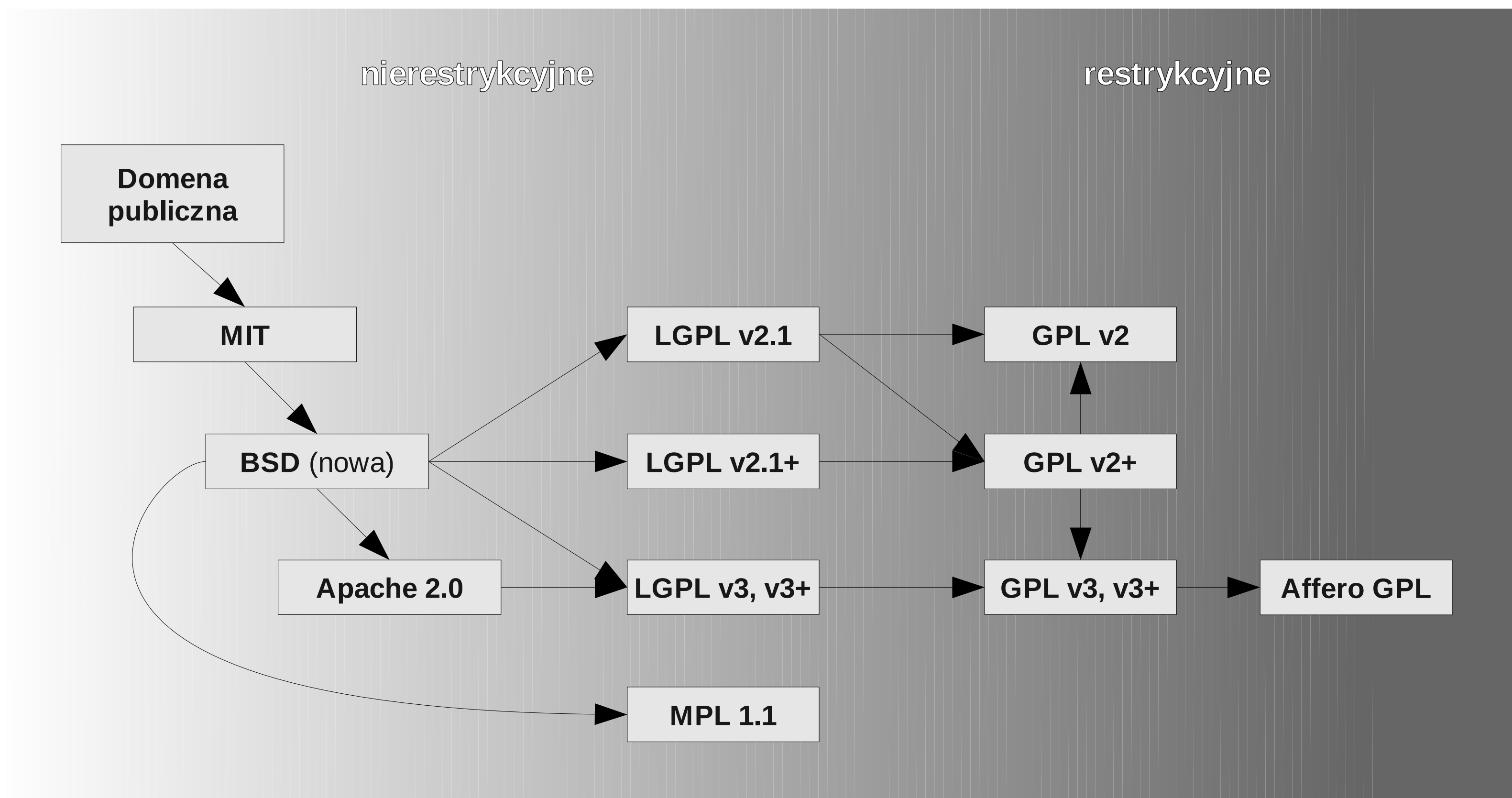
B) zmodyfikowane wersje nie mogą nakładać żadnych dodatkowych zobowiązań na mocy swojej licencji.

Jeżeli ktoś stworzyłby oprogramowanie zawierające pewien komponent udostępniony na licencji A oraz pewien komponent udostępniony na licencji B, wtedy nie miałby on żadnej legalnej możliwości rozprowadzania tego oprogramowania. Zasad obydwu licencji nie da się spełnić jednocześnie.

Sprawa jest prosta w przypadku, gdy rozstrzyga się kompatybilność licencji *open source* z tak zwanymi licencjami własnościowymi stosowanymi w zamkniętym oprogramowaniu. Licencje własnościowe nie zapewniają żadnego dostępu do źródeł oprogramowania oraz wszelkiej jego modyfikacji. Są one niekompatybilne w obydwie strony z restrykcyjnymi licencjami open source. Jeżeli chodzi o licencje nierestrykcyjne, to można używać ich w zamkniętych projektach, jeżeli spełni się specjalne dodatkowe wymagania, które czasami w sobie zawierają.

Sytuacja jest nieco bardziej skomplikowana, gdy rozpatruje się kompatybilność między samymi licencjami *open source*. Ogólna zasada jest taka, że mniej restrykcyjne licencje są zazwyczaj kompatybilne z bardziej restrykcyjnymi, ale nigdy w drugą stronę. Na przykład dopuszczalne jest wykorzystanie kodów źródłowych na nierestrykcyjnej licencji MIT w oprogramowaniu na licencji GPL i wydanie całego produktu na zasadach restrykcyjnej licencji GPL, ale nie jest dozwolone postępowanie odwrotne, czyli opublikowanie wszystkiego na licencji MIT. David A. Wheeler stworzył schemat, który bardzo dobrze ilustruje, jak wygląda kwestia kompatybilności pomiędzy najpopularniejszymi licencjami *open source*[10]. Ilustracja ta jest przedstawiona poniżej (rys.2.4.B).

10 The Free-Libre / Open Source Software (FLOSS) License Slide - David A. Wheeler, 2007, http://www.dwheeler.com/essays/floss-license-slide.html



Rys.2.4.B. Kompatybilność licencji

Prostokąty na rys. 2.4.B reprezentują różne licencje *open source*. Strzałka od prostokąta A do prostokąta B oznacza istnienie możliwości łączenia ich pod warunkiem, że nowe oprogramowanie zostanie wydane na licencji B z ewentualnymi dodatkowymi postanowieniami wynikającymi z licencji A. Aby zweryfikować, czy oprogramowanie na dwóch licencjach może być połączone, należy sprawdzić czy istnieje wspólny prostokąt, do którego można dojść z obu licencji, przechodząc wedle schematu przedstawionego na ilustracji. Na przykład, z prostokąta licencji Apache 2.0 oraz licencji GPL v2 można dojść do prostokąta GPL v3. Zatem można połączyć oprogramowanie na obu licencjach pod warunkiem, że powstały produkt zostanie wydany na licencji GPL v3. Ponadto można także wydać ten produkt na licencji Affero GPL v3.

Łatwo zauważyć po złożoności przedstawionego tutaj diagramu, że zachowanie kompatybilności wielu licencji nie jest zadaniem trywialnym. Zwłaszcza w systemach wykorzystujących wiele otwartych komponentów. Z ankiet[DA] wynika, że niemal połowa personelu IT uważa możliwość złamania warunków różnych licencji *open source* za istotne ryzyko. Niemal identyczny wynik uzyskano dla grupy osób twierdzących, że posiadają wiedzę z zakresu organizowania projektów *open source*, jak i dla grupy zupełnie nie obeznanej w tej tematyce.

Struktury organizacyjne

Rozdział 3

3.1. Ogólna kultura organizacyjna

Kultura organizacyjna otwartych projektów odbiega w wielu aspektach od tego, co można spotkać w zamkniętych przedsięwzięciach wewnątrzfirmowych. *Open source* nie polega jedynie na wydawaniu oprogramowania na zasadach otwartej licencji. Jeżeli poprowadzi się tego typu projekt w typowym korporacyjnym stylu, wtedy nie zyska się tych wartości, jakich dostarczają otwarte metodologie. Poza ujawnianiem „wnętrza” swojego oprogramowania, otwartość ta powinna przejawiać się w bardziej publicznej formie podejmowania decyzji. Takie podejście często zmusza do ujawnienia pewnych wewnątrzprojektowych postanowień, które lepiej byłoby zachować w tajemnicy. Z drugiej strony, jako że społeczność użytkowników otwartego oprogramowania jest świadoma tego faktu, skutkuje to często większym zaufaniem z ich strony.

Budowanie społeczności

Zwiększony wpływ na projekt daje użytkownikom motywację, aby spędzać więcej czasu na przygotowaniu raportów o błędach. Można również oczekiwać od nich aktywnej pomocy w precyzyjnym lokalizowaniu przyczyn błędów. Z reguły w projektach *open source* użytkownicy są bardziej aktywni w dyskusjach na temat przyszłości oprogramowania oraz zapewniają bardziej szczegółową informację zwrotną na temat jego obecnej użyteczności. Aby uzyskać taki poziom zaangażowania, trzon ludzi odpowiedzialnych za rozwój projektu powinien unikać dokonywania prywatnych decyzji wewnątrz własnego grona. Nowe pomysły oraz plany dalszego rozwoju powinny być upubliczniane tak szybko, jak tylko się pojawią[POS].

Istotna część dobrze kierowanej społeczności z reguły oferuje swoją pomoc w testowaniu wydań rozwojowych oprogramowania. Aby dać im tę możliwość należy wypuszczać kolejne wydania często i w pakietach, które są łatwe do uruchomienia. W projektach *open source* sprawdziła się zasada, że im bardziej złożony projekt, tym częściej powinny ukazywać się nowe wydania, chociażby w postaci wydań oznaczonych jako rozwojowe lub niestabilne. Niektóre projekty, jak np. Apache ActiveMQ oraz Ekiga[11], automatyzują ten proces do stopnia, w którym mogą

11 Post na oficjalnym blogu Ekiga, odnośnie przywrócenia tzw. ang. nightly builds, http://blog.ekiga.net/?p=90

zapewnić codziennie tzw. migawki (ang. *snapshot*). Są to wydania budowane z ich najbardziej aktualnych źródeł w postaci paczki gotowej do uruchomienia „od zaraz”. Tego typu praktyki pozwalają użytkownikom na przyłączenie się do testów bez większego wysiłku z ich strony. Wnioski wynikające z tego typu testów potrafią niekiedy obalić kluczowe decyzje projektowe zanim będzie za późno, aby w prosty sposób się z nich wycofać. Dlatego właśnie posiadanie dużej liczby testerów w świecie *open source* nigdy nie jest postrzegane jako redundancja pracy w projekcie[CatB]. Koncepcja częstego wydawania nowych wersji oprogramowania toruje sobie drogę w świecie zamkniętego oprogramowania i jest postrzegana jako odejście od tradycyjnych metodologii na rzecz bardziej zręcznych (ang. *agile*) i ekstremalnych. Jednym z podstawowych założeń tego typu metodologii jest fakt, że to czego klient oczekuje początkowo od oprogramowania nie jest zazwyczaj tym czego naprawdę potrzebuje. Klient często poznaje swoje prawdziwe potrzeby dopiero, gdy oprogramowanie jest już częściowo napisane i zaczyna wchodzić w fazę testów[EPM]. Zasady metodyk zręcznych mówią, że w im mniejszym stopniu jest możliwe jasne określenie naszego celu lub sposobów na jego osiągnięcie, tym bardziej należy być przygotowanym na częste wydawanie oprogramowania oraz jego prototypownie.

Jak można zauważyć, otwieranie kodów źródłowych oraz uzewnętrznienie procesu wytwarzania oprogramowania nie tylko zmienia sposób, w jaki zorganizowana jest praca, ale także zmienia relacje między ludźmi. Między innymi pozwala na traktowanie użytkowników oprogramowania jako współpracowników i podnosi wartość beta-testerów w porównaniu z projektami zamkniętymi[CatB]. W świecie *open source* ogólne praktyki zalecają angażować jak największą liczbę osób w proces analizy poszczególnych błędów oprogramowania. Określane jest to często tzw. prawem Linusa, które mówi, że[CatB]:

Dając wystarczająco dużą liczbę obserwujących, rozwiązania wszystkich bugów stają się oczywiste.

To poprawia proces zachowania jakości, ale też zwiększa narzut wysiłku poświęconego na komunikację wewnątrz projektu. Aby okiełznać ten dodatkowy nakład, projekty *open source* muszą wykorzystywać w sposób efektywny narzędzia, takie jak trackery, fora dyskusyjne, archiwizowane listy mailingowe, itp. Komunikacja na tak szeroką skalę zazwyczaj była postrzegana jako negatywny

czynnik w zamkniętych przedsięwzięciach. Jednak w świecie *open source* okazała się ona skutecznym sposobem na przyspieszenie procesu wytwórczego.

Inną ważna konsekwencją takiej formy komunikacji, a raczej narzędzi w niej wykorzystywanych, jest fakt, że niemal każda rozmowa na temat danego problemu jest na stałe zapisywana i indeksowana w internecie. Daje to każdemu programiście lub użytkownikowi, który natrafił na dany problem, dużą szansę na dotarcie do innych ludzi, którzy również go napotkali. Wartość, jaka się w tym kryje, często jest niedoceniana. Taka forma przeprowadzania konwersacji wewnątrz projektu sprawia, że rozwiązania lub obejścia dotyczące większości problemów, które napotkamy, można od razu odnaleźć poprzez wyszukiwarkę internetową.

Open source zmienia relacje publiczne w jeszcze jednym wymiarze. Od momentu, w którym dany projekt zostaje otwarty, ludzie zaczynają go postrzegać bardziej jako dobro publiczne, a mniej jako cudze aktywa do robienia pieniędzy. Dla różnego typu otwartych przedsięwzięć jest to w większym, bądź mniejszym stopniu prawdą, nawet jeżeli jakieś firmy zarabiają na zainwestowanych tam środkach. W większości przypadków zwiększa to motywację pracowników zatrudnianych przez założyciela projektu oraz przyciąga programistów z zewnątrz wykazujących inne motywy niż wynagrodzenie pochodzące bezpośrednio od niego. Najważniejsze z nich omówione zostały w rozdziale 2.3.

Kultura pracy programistów ze środowiska open source

Poza wyższą motywacją i poczuciem większej swobody w projekcie, zazwyczaj spotyka się również odmienną kulturę pracy pośród programistów, w stosunku do projektów zamkniętych. Dobra kombinacja tych czynników częstokroć sprawia, że przeciętna porcja kodu od programisty w *open source* jest jakościowo lepsza od tego wykonanego przez osobę zatrudnioną w zamkniętym projekcie. Potwierdzają to badania gęstości defektów w kodzie wykonane na źródłach jądra Linux[12] oraz bazy danych MySQL[13]. Jedną z przyczyn takiego stanu rzeczy może być fakt, że programiści *open source* zazwyczaj sami przydzielają się do zadań, których

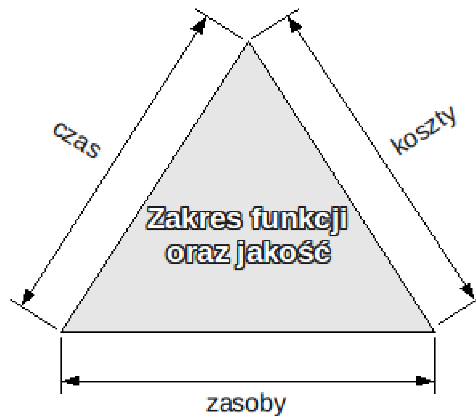
12 Reasoning Inc. A Quantitative Analysis of TCP/IP Implementations in Commercial Software and in the Linux Kernel.

13 Reasoning Inc. How Open Source and Commercial Software Compare: Database Implementations in Commercial Software and in MySQL.

się podejmują, nawet jeżeli są zatrudnieni lub mają podpisany kontrakt z jakąś inną stroną, której zależy na konkretnych pracach w projekcie. Kolejną przyczyną z pewnością jest faworyzowanie tak zwanego programowania bez ego (ang. *egoless programming*). Podejście to polega na unikaniu rozgraniczania kodu między programistami oraz na motywowaniu do niepersonalnej krytyki cudzych zmian w repozytorium[CatB]. Jednym z przejawów tego jest powszechna praktyka recenzowania kodu (ang. *code review*) w świecie open source[POS].

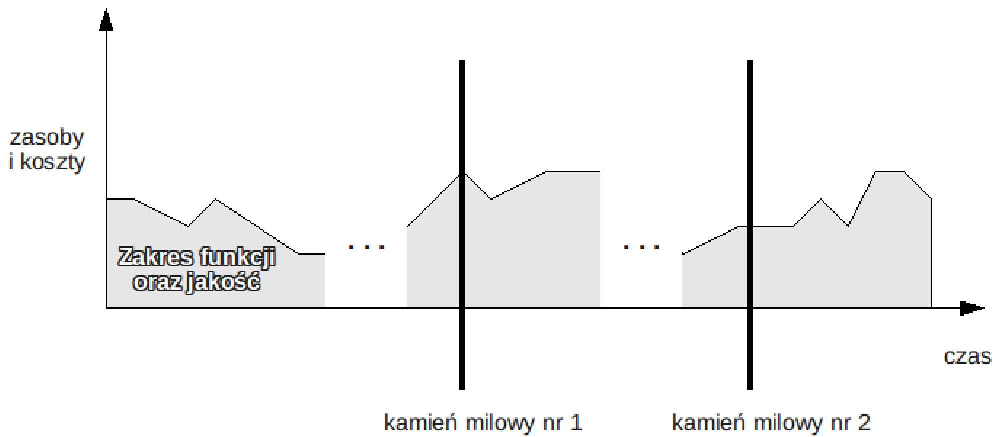
Innym aspektem, który podnosi jakość kodu, jest częsty brak ścisłych terminów wydań oprogramowania (ang. *deadline*). Im bardziej projekt jest sterowany przez społeczność (zwłaszcza w społeczności merytokratycznej, patrz roz. 3.3) tym mniej prawdopodobne, że będzie zezwolenie na ponaglanie pracy ze względu na jakikolwiek termin. Jeżeli chodzi o społeczności zorganizowane wokół pojedynczych firm (patrz roz. 3.4), to terminy wydań są nadal obecne i często zsynchronizowane z taktyką firmy. Jak wiadomo z teorii zarządzania projektami, istnieją cztery podstawowe własności każdego kolejnego wydania oprogramowania: zakres nowych funkcji, jakość, termin oraz dostępne zasoby. Wiadomo również, że nie można maksymalizować ich wszystkich jednocześnie. Jeżeli intencją jest wydanie oprogramowania we wcześniejszym terminie, to należy zawrzeć w nim mniej funkcjonalności lub poświęcić jego jakość i *vice versa*. Ponieważ w wewnątrzfimowych projektach terminy i zakres nowych funkcji są często wyznaczane przez strategie marketingowe, jakość pozostaje wtedy jedyną własnością, którą można zaniżyć[CatB]. Wiele projektów sterowanych przez społeczność podąża za prostą zasadą: *będzie wydane, kiedy będzie gotowe*, a jakość jest ostatnią właściwością, którą członkowie owej społeczności będą skłonni manipulować. Istnieje jeszcze czynnik poziomu dostępnych zasobów, który można zmodyfikować, aby utrzymać planowany poziom jakości. Jednak ich zmiana w trakcie projektu jest dosyć trudnym zadaniem z powodu dwóch zjawisk. Pierwszym z nich jest tak zwana *pułapka nadziei*, w której współpracownicy kłamią na temat ich faktycznego postępu łudząc się, że niepostrzeżenie nadrobią zaległości[EPM]. Efekt ten wzrasta proporcjonalnie do zakresu projektu, liczebności zespołu oraz do dalszej odległości terminu wydania. Jeżeli już wpada się w tę pułapkę często istnieje konieczność modyfikacji budżetu oraz zatrudniania nowych programistów na stosunkowo późnym etapie. W tym momencie można spotkać się z drugim zjawiskiem nazywanym prawem Brooksa[MMM]: *dodanie siły roboczej do spóźnionego projektu sprawia, że będzie on jeszcze bardziej spóźniony*.

Pojęcie zasobów oraz kosztów same w sobie jest ciekawym zagadnieniem w świecie *open source*. Każdy zamknięty lub otwarty projekt, który miałby mieć ściśle określony termin wydania, musiałby mieć jednocześnie określony poziom kosztów oraz listę dostępnych zasobów. Te czynniki determinują, jaki zakres funkcjonalności będzie można zawrzeć w przyszłym wydaniu i na jakim poziomie jakości. Jest to zazwyczaj ilustrowane w postaci trójkąta, w którym czas, zasoby i koszty to jego krawędzie, a zakres funkcji oraz jakość to jego pole[EPM] (rys. 3.1.A).



Rys. 3.1.A. Właściwości projektu

Jednak w świecie *open source* koszty i zasoby potrafią być czynnikami nieprzewidywalnymi z powodu spontanicznej natury otwartego modelu wytwarzania oprogramowania. Ma to miejsce zwłaszcza w bardzo społecznościowych projektach, na przykład mających merytokratyczną strukturę organizacyjną (patrz roz. 3.3), dlatego właśnie brak *deadline'ów* nie jest strategią wyłącznie opartą na preferencjach programistów *open source*. Większość otwartych projektów radzi sobie z tym założeniem poprzez planowanie swojej pracy z pomocą, tak zwanej, mapy drogowej (ang. *roadmap*). Ów plan grupuje zadania w tak zwane kamienie milowe (ang. *milestones*), do których nie przypisuje się żadnych terminów, ale muszą być realizowane w określonej kolejności (rys. 3.1.B).



Rys.3.1.B. Projekt z kamieniami milowymi

To daje projektom *open source* poczucie kierunku, w którym zmierzają. Odpowiednie równoważenie zadań pomiędzy kamieniami milowymi daje administratorom możliwość oceny, jak bardzo stabilny jest poziom aktywności społeczności projektowej.

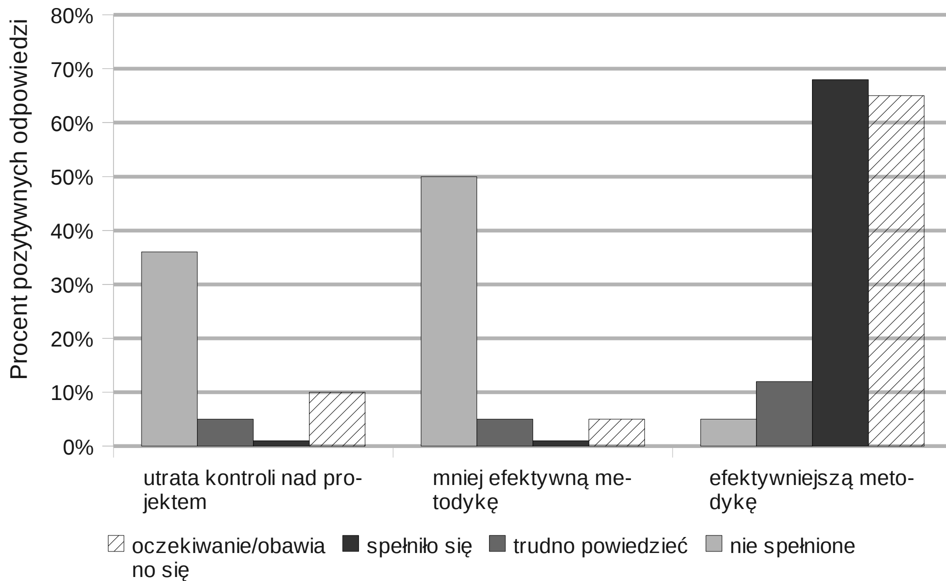
Efektywność otwartych metodyk zdaniem pracowników IT

W badaniach ankietowych[DA] pośród pracowników IT podzielono respondentów na trzy grupy, w zależności od ich dotychczasowego uczestnictwa w projektach *open source*. Wszystkich zapytano co myśleliby o otwarciu projektów, w których właśnie uczestniczą. Zestawienie tych odpowiedzi przedstawiono poniżej (rys. 3.1.C).

Rys.3.1.C. Opinie pracowników IT na temat otwarcia

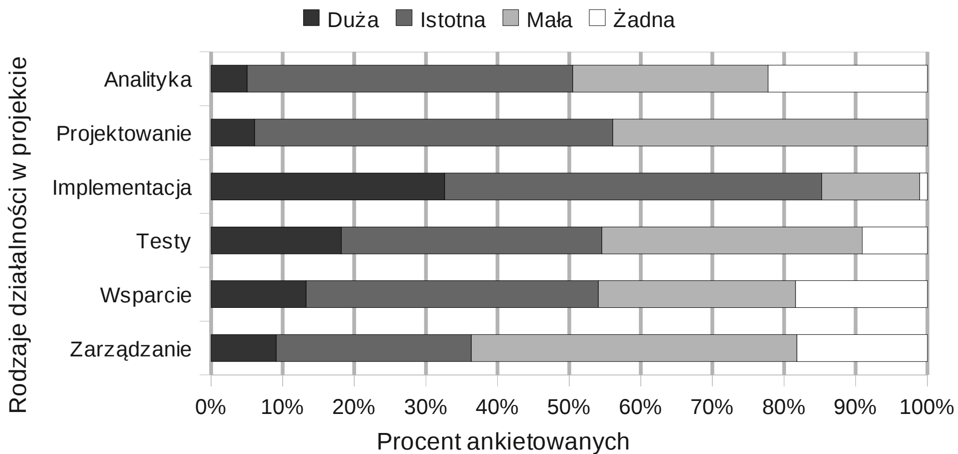
Z zestawienia wynika, że wizja efektywności jest przeciwna w porównaniu grupy o dużym stopniu zaangażowania do grupy generalnie nie uczestniczącej w projektach *open source*. Analizując te dane można jednak zauważyć, że ponad 60% aktywnych programistów *open source*, bierze także udział w projektach zamkniętych. Może to oznaczać, że mają większą możliwość porównania obu światów. Innym ważnym czynnikiem jest tu fakt, że ponad 50% ankietowanych nie biorących znaczącego udziału w otwartych projektach nie posiada większej wiedzy na temat tego, jak są one zorganizowane, a mimo niskiej oceny efektywności niemal 90% z nich ma pozytywną opinię o otwartych produktach.

Spójrzmy na wyniki ankiet[DA] w zakresie wątpliwości oraz oczekiwań, jakie administratorzy pokładali w organizacyjnych aspektach otwartych projektów jeszcze przed ich rozpoczęciem. Zestawione zostaną one z informacją o tym, które z oczekiwań administratorzy uważają za spełnione.



Rys.3.1.D. Opinie właścicieli projektów *open source* na temat otwarcia

Jak widać, oczekiwania te są podobne do ogółu programistów *open source*, a opinie po otwarciu projektów są bardzo pozytywne. Niewielka część administratorów oczekiwała negatywnych skutków na organizację prac po otwarciu, lecz w rzeczywistości niemal w żadnym przedsięwzięciu one nie wystąpiły. Przyglądając się dalej wynikom ankiety, możemy sprawdzić, jak badani administratorzy oceniają efektywność otwartych projektów w poszczególnych rodzajach jego działalności. Zestawienie na rys. 3.1.E przedstawia, jaki procent ankietowanych administratorów określiło efektywność otwartej metodologi w danym obszarze prac.



Rys.3.1.E. Efektywność w poszczególnych rodzajach działalności projektu open source

Podejmowanie decyzji

Podejmowanie decyzji jest dość subtelnym zagadnieniem w świecie *open source*. W otwartych projektach proces ten znacznie różni się od mocno zhierarchizowanego łańcucha zarządzania stosowanego w większych firmach i korporacjach. Jednym z głównych czynników wymuszających inne podejście jest brak wyłącznych praw do rozwoju oraz dystrybucji danego oprogramowania. Każda grupa, która sprzeciwia się pierwotnym założycielom projektu, może rozpocząć nowy projekt bazujący na ich pracy. Tego typu przedsięwzięcie w mowie potocznej programistów nazywane jest „forkiem” (od angielskiego słowa oznaczającego rozwidlenie) i w zależności od kultury danej społeczności może mieć pozytywny lub negatywny wpływ na „sforkowany” projekt. Więcej na temat tego zjawiska zostanie powiedziane w rozdziale 3.6. Paradoksalnie otwarcie liberalnej furtki dla tworzenia „forków” jest czymś, co spaja ze sobą członków danej społeczności projektu i motywuje ich do rozwiązywania konfliktów przez konsensus, a nie, jak to zwykle bywa, przez autorytet[POS].

„Forkowalność” nadaje moc zasadzie, że największy wpływ na podejmowanie decyzji jest dany ludziom, którzy wykonują najwięcej pracy na rzecz projektu. W pewien sposób daje to komercyjnym firmom możliwość utrzymania lub

zdobywania przywództwa poprzez, na przykład, finansowanie głównych programistów. Jednak firmy muszą być świadome faktu, iż „forkowalność” sprawia, że decyzje w projektach open source są zazwyczaj podejmowane w pozycji wertykalnej, tj. od dołu do góry (ang. *bottom-up*). O projektach zamkniętych wewnątrz korporacji można powiedzieć tyle, że decyzje z reguły są w nich podejmowane na górnym poziomie przez menedżerów i analityków, a następnie wdrażane na niższym poziomie przez ekipę programistów. W świecie *open source* angażowani są najpierw kluczowi programiści, a decyzje które podejmują są akceptowane lub też odrzucane przez społeczność projektu. Zresztą sami menedżerowie oraz analitycy mogą być częścią tej społeczności. Proces w stylu *bottom-up* jest często pożądany, z tego względu, iż większość dobrze wykonanej pracy w oprogramowaniu rozpoczyna się poprzez pobudzenie prywatnego zainteresowania programisty w danym zagadnieniu[CatB]. Projektanci oraz menedżerowie projektu *open source* powinni być świadomi, że od posiadania w pojedynkę dobrych pomysłów znacznie ważniejsza jest tutaj zdolność do rozpoznawania dobrych pomysłów wewnątrz większej społeczności projektu.

Niektóre projekty *open source* są bardzo ściśle powiązane z pojedynczą firmą i zorganizowane w strukturę z wewnętrznym procesem wytwórczym oraz sprzężeniem zwrotnym w ramach społeczności (patrz roz. 3.4). Pozwala to na zaaplikowanie bardziej tradycyjnych metodologii zarządzania oraz modeli biznesowych, takich jak na przykład *podwójne licencjonowanie* (patrz roz. 4.2). Jednym z efektów ubocznych tego typu organizacji jest niższy udział programistów niezatrudnionych bezpośrednio przez założyciela projektu w porównaniu do tych prowadzonych w sposób bardziej społecznościowy. Przykładem tych ostatnich jest społeczność z życzliwymi dyktatorami (patrz roz. 3.2). Struktura ta jest nieco bardziej zdecentralizowana, jednak nadal zachowuje jasne poczucie przywództwa, które najczęściej skupia się w jednej osobie założyciela projektu lub głównego programisty. Projekty zorganizowane w ten sposób mają z reguły bardzo nieformalną strukturę i są oparte na zaufaniu do swoich dyktatorów.

Pieniądze to delikatny temat

Projekty open source, które posiadają łatwo rozpoznawalne centrum przywództwa, często posiadają do dyspozycji zarówno programistów zatrudnianych przez to centrum, jak i programistów wolontariuszy, którzy dołączyli do projektu

dobrowolnie. Pierwsza grupa jest motywowana w głównej mierze przez owe wynagrodzenie i reprezentuje interes finansującego podmiotu w projekcie. Członkowie drugiej grupy są natomiast motywowani głównie przez samo posiadanie częściowej władzy nad projektem lub są zatrudnieni przez inne przedsiębiorstwa, które wykazują potrzebę pozyskania takich wpływów. Jeżeli przywódca projektu nie będzie posiadał umiejętności sprawiedliwego rozdzielania władzy pomiędzy te dwie grupy, to wtedy kwestia pieniędzy może podzielić społeczność na grupy pierwszej oraz drugiej kategorii[POS]. Jeżeli nieopłacani wolontariusze poczują, że decyzje projektowe są po prostu podejmowane przez najwyżej płacącego gracza, to wtedy odejdą poszukać bardziej merytokratycznie zorganizowanych projektów. Zjawisko to może być długo niezauważalne z perspektywy założyciela, jako że wolontariusze ci mogą nigdy nie protestować publicznie. Zamiast tego w projekcie będą stopniowo zanikać głosy z zewnętrznych źródeł[POS]. Najczęściej jest to spotykane w projektach z wewnętrznym procesem wytwórczym i sprzężeniem zwrotnym w ramach społeczności. Natomiast projekty zorganizowane w społecznościach o charakterze merytokratycznym są odporne na ten problem w największym stopniu, jako że tego typu organizacja jest systemem, który determinuje wpływ danej osoby tylko na podstawie jej wcześniejszego wkładu w sam projekt.

Firmy, które chcą się przyłączyć do projektu *open source,* nie powinny być zbyt zachłanne podczas wdrażania tam swojej siły roboczej. Większość projektów ma ścisłe wytyczne co do przyznawania zezwoleń na wykonywanie zmian bezpośrednio na źródłach projektu. Przyprowadzanie dużej liczby programistów z ogromną ilością łat kodów źródłowych nie zapewni im autoakceptacji w społeczności. Zwłaszcza w społecznościach merytokratycznych powszechną normą jest, że każdy członek ekipy wynajętej przez firmę będzie musiał zyskać dla siebie prawa do bezpośredniego wprowadzania zmian, dokładnie w taki sam sposób, jak każdy inny członek społeczności danego projektu. Najczęściej osiąga się to poprzez zapewnianie odpowiedniej liczby małych i dobrze wykonanych łat, które zostaną pozytywnie zrecenzowane. Jest to postrzegane jako dobra praktyka przez większość społeczności i nie przestrzeganie jej mogłoby skierować projekt w powszechnie nieakceptowalnym kierunku. Bez zachowania opisywanej tutaj ostrożności nieopłacani programiści mogliby się poczuć niedowartościowani. Dobrym przykładem, w którym pracownicy zewnętrznej firmy zdobywali stopniowo i uczciwie swoje uprawnienia był przypadek

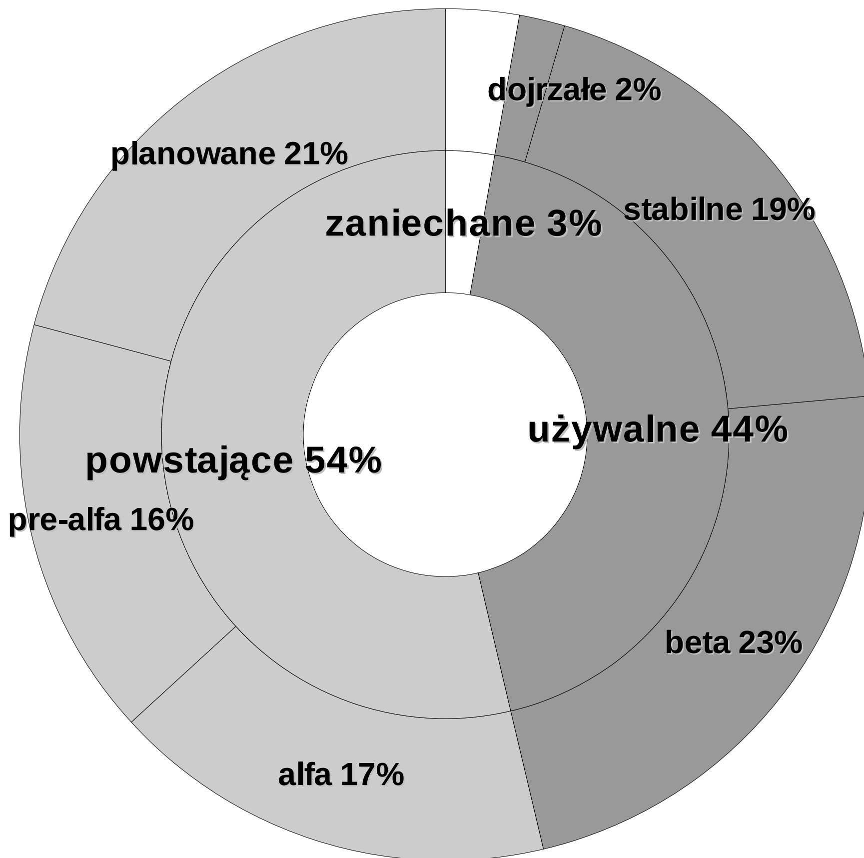
firmy Sun Microsystems i projektu Apache Tomcat[14]. Zasady wymagające, aby fundatorzy projektu grali według tych samych zasad co wszyscy, są nieco trudniejsze do wyegzekwowania w strukturze organizacyjnej opartej na życzliwych dyktatorach. Zwłaszcza jeżeli owi dyktatorzy są zatrudnieni przez wspomnianych fundatorów[POS].

W przypadku, gdy pewna firma lub osoba chce zakontraktować programistę do dodania jakieś funkcjonalności do danego oprogramowania, obie strony muszą być świadome w jaki sposób postępuje proces podejmowania decyzji wewnątrz danego projektu. W idealnej sytuacji zakontraktowany programista powinien wykonać łatę kodów źródłowych, oddać ją w ręce społeczności i doprowadzić do jej akceptacji w przyszłym wydaniu. Jednak jeżeli ta łata jest kontrowersyjna, źle przetestowana lub nie wpasowuje się w ogólną wizję oprogramowania wytwarzanego przez tę społeczność, wtedy może zostać odrzucona. W tym przypadku powstanie konieczność utrzymywania tej łaty samodzielnie oraz integrowanie jej z kolejnymi wydaniami oprogramowania. Dlatego też dobrą praktyką jest przeprowadzanie dyskusji, nad proponowanymi zmianami, jeszcze przed podjęciem prac. Sam kontrakt nie może wymagać od zleceniobiorcy, aby *patch* został zaakceptowany, ponieważ oznaczałoby to sprzedawanie czegoś, co nie jest na sprzedaż[POS].

Ryzyko niepowodzenia

Jak widać, tworzenie oprogramowania w świecie *open source* cechuje się szeregiem niuansów natury organizacyjnej. Często nasuwa się także pytanie: jakie jest faktyczne ryzyko niepowodzenia otwartego projektu? W wielu krążących po sieci artykułach z reguły więcej słyszy się o sukcesach, aniżeli o porażkach *open source*. Jednak przeglądając losowe przedsięwzięcia na jednym z wielu portali zapewniających infrastrukturę otwartym projektom, zauważa się, że około 90% z nich zostało porzuconych lub wykazuje śladową aktywność. Analizując bazę 215 000 projektów prowadzonych na SourceForge.net (stan z początku roku 2010) w rozbiciu na etap rozwoju, deklarowany przez właścicieli owych projektów, otrzymuje się poniższy podział (rys. 3.1.F).

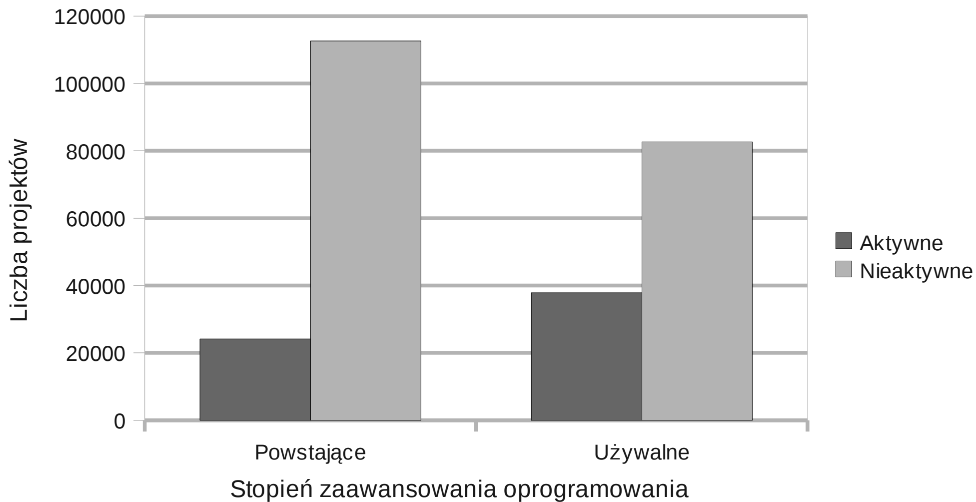
14 A strong OpenOffice.org community is the key - Danese Cooper's (pracownik Sun Mircosystems w tym czasie), wpis na blogu, 16.09.2004, http://blogs.sun.com/DaneseCooper/date/20040916



Rys.3.1.F. Stopnie rozwoju projektów

Interpretując te wyniki, przy wykazaniu dużej ilości dobrej woli, można by dojść do wniosku, że autentyczne ryzyko niepowodzenia wynosi jedynie 3%, a 54% oprogramowania we wczesnej fazie rozwoju świadczy o nadchodzącym boomie w branży *open source*. Opierając się jednak nie na deklaracjach właścicieli projektów, a na współczynniku aktywności obliczanego przez SourceForge na podstawie ruchu w repozytoriach kodu, listach dyskusyjnych oraz trackerach, można dojść do diametralnie innych wniosków. Przyjmując, że stopień aktywności mniejszy od 80% określa się jako zaniechanie przedsięwzięcia, można zauważyć, że przedsięwzięć zaniechanych jest około 160 000, co stanowi 75% całego hostingu. Przeciętny projekt z wynikiem w okolicy 80 % cechuje się niemal brakiem jakichkolwiek aktywności na wymienionych wcześniej narzędziach w ostatnich 3 latach. Gdy rozbijemy ten wynik

na stopień zaawansowania oprogramowania, otrzymamy następujący wykres, przedstawiający liczbę projektów aktywnych i nieaktywnych w zależności od stopnia rozwoju wytwarzanego oprogramowania (rys. 3.1.G).



Rys.3.1.G. Aktywność a poziom rozwoju

Na powyższym wykresie (rys. 3.1.G) widać, że projekty na wczesnym etapie rozwoju mają niemal trzykrotnie większe ryzyko porzucenia w stosunku do tych projektów, którym udało się rozwinąć swój produkt do używalnej postaci.

W niniejszym rozdziale ryzyko postrzegane jest wyłącznie jako zamarcie prac nad oprogramowaniem. W świecie *open source* zazwyczaj jest ono spowodowane zanikiem aktywności społeczności projektu lub odcięcia dofinansowania przez podmioty opłacające programistów. Istnieje też ryzyko niespieniężenia pracy włożonej w oprogramowanie *open source* przez firmy budujące swoje modele biznesowe na jego bazie. Ten aspekt będzie omawiany w rozdziale 4.

3.2. Społeczności z życzliwymi dyktatorami

Społeczności z życzliwymi dyktatorami (ang. *benevolent dictators*) są dokładnie tym, co podpowiada intuicja tuż po usłyszeniu tego terminu. Ostateczne zdanie podczas podejmowania decyzji leży w gestii kilku lub tylko jednej osoby,

której pryzmat charakteru lub doświadczenia zapewnia autorytet w tej dziedzinie. Pomimo że termin „*benevolent dictator*” (w skrócie BD) jest w powszechnym użyciu, bardziej trafnym byłoby używanie określenia: *arbiter zatwierdzony przez społeczność* lub *sędzia*. Dyktatorzy z reguły nie podejmują większości decyzji w projekcie. Są jednak ostatecznym głosem rozstrzygającym kwestie, w których społeczność nie może dojść do konsensusu[POS]. Czasami również wykorzystują swój autorytet, aby zawetować decyzje, które uważają za wyjątkowo nietrafne. Jednak zazwyczaj takie weto jest jedynie formą gry na czas, w której dyktatorzy próbują przekonać społeczność do swojego punktu widzenia.

Społeczność z życzliwymi dyktatorami jest naturalną strukturą organizacyjną dla większości raczkujących projektów open source rozpoczętych przez małą grupę lub jednego programistę. Z badań ankietowych[DA] wynika, że stanowią oni 80% założycieli projektów. Wedle tych samych wyników wiemy także, że tego typu struktury organizacyjne znajdują zastosowanie w 33% projektów, w których jest mniej niż 30 aktywnych członków. Założyciele naturalnie zyskują autorytet wśród społeczności powstającej wokół ich oprogramowania z powodu występowania dwóch oczywistych czynników. Po pierwsze, w danym momencie ich wkład pracy jest największy. Po drugie, znają swoje oprogramowanie jak nikt inny, dzięki czemu ich decyzje niosą ze sobą największą potencjalną wartość. Efektywność omawianej tutaj struktury organizacyjnej jest wysoce zależna od umiejętności społecznych i technicznych posiadanych przez życzliwych dyktatorów, co wpływa na sposób odbioru przywództwa i jego zakres w danym projekcie. Jest to pomocne, między innymi, w zachowaniu spójnej wizji architektury oprogramowania oraz kierunku, w którym ono podąża. Bardzo często jest to efekt pożądany, który pozwala projektom zorganizowanym w omawiany sposób sukcesywnie rosnąć, nawet po przejściu już okresu dojrzewania oraz po zbudowaniu dużej społeczności.

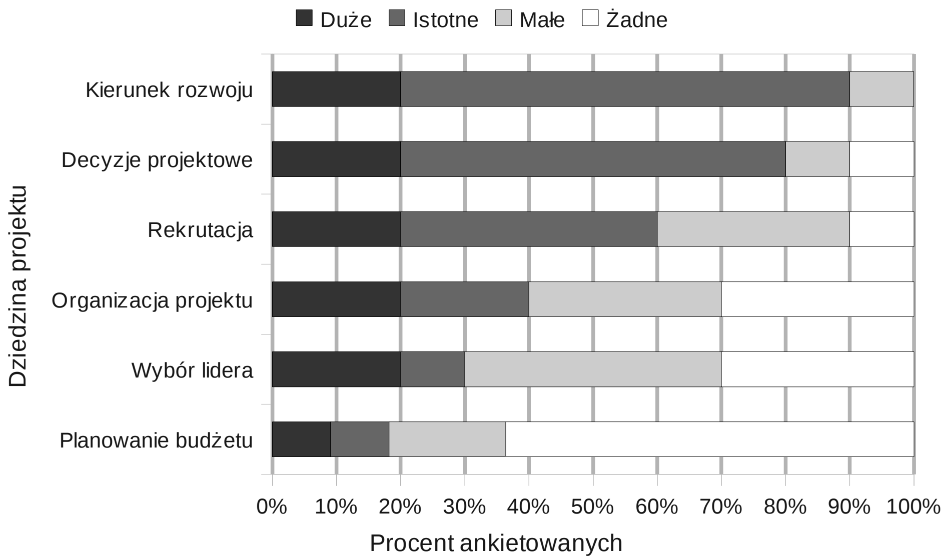
Budowa stabilnej społeczności projektowej przez życzliwego dyktatora wymaga od niego, jak już zostało wspomniane, dobrego połączenia umiejętności społecznych i technicznych. Z technicznego punktu widzenia musi on być dobrym projektantem, zdolnym do rozpoznawania dobrych pomysłów projektowych podsuwanych przez społeczność[CatB]. Z perspektywy społecznej powinien być subtelnym rozjemcą konfliktów, w których nie można dojść do konsensusu. Dyktator musi być także przywódcą, wobec którego ludzie wykazują pewien stopień przywiązania oraz który przyciąga nowych członków do projektu. Eric S. Raymond

stawia nawet umiejętności społeczne wyżej niż techniczne. Według niego przywódca otwartego projektu powinien mieć przynajmniej zdolność do zrzeszania i zjednywania sobie ludzi[CatB]. Może to właśnie dlatego wielu dyktatorów udanych projektów *open source* zyskuje wśród swoich społeczności dożywotni tytuł *Benevolent Dictator For Life* (w skrócie BDFL – dożywotni życzliwy dyktator). Termin ten został po raz pierwszy przypisany twórcy języka Python oraz przywódcy całego projektu Guido van Rossum[15].

Jednym z najważniejszych minusów tego typu struktury organizacyjnej jest potencjalne ogromne zamieszanie w projekcie, które może być spowodowane rezygnacją życzliwego dyktatora, bądź utratą jego autorytetu. Prostym wyjściem z pierwszej sytuacji jest mianowanie następcy, co jest uważane jako ich ostatni obowiązek wobec społeczności[CatB]. Jednak takie przekazanie władzy jest łatwe tylko wtedy, gdy w społeczności istnieją ogólnie akceptowani kandydaci, których umiejętności będą w stanie utrzymać prace w projekcie na poziomie porównywalnym z poprzednim. Zły wybór wprowadza w społeczności takie same napięcia, jak utrata autorytetu przez dyktatorów, co może w ostateczności skończyć się „forkiem" projektu (patrz roz. 3.6). W obu przypadkach najczęstszą tendencją jest ewolucja społeczności w merytokratyczną strukturę organizacyjną. Społeczności, które przyjęły taką formę organizacji z reguły już nigdy nie wracają do koncepcji życzliwych dyktatorów[POS]. Struktura merytokratyczna zapewnia większą stabilność oraz rozprasza proces podejmowania decyzji. Jednak decyzje same w sobie mogą stracić na spójności oraz ciągłości w porównaniu do tych podejmowanych z udziałem życzliwych dyktatorów (patrz roz. 3.3).

Mimo że ostateczna władza leży w rękach życzliwych dyktatorów, to według badań ankietowych[DA] poziom swobody w decyzjach projektowych takiej społeczności jest wysoki. Na poniższym wykresie (rys. 3.2.A) przedstawiono, jaki procent życzliwych dyktatorów uważa swobodę społeczności za: dużą, istotną, małą lub żadną, w określonej dziedzinie swojego projektu.

15 Origin of BDF - Guido van Rossum, 01.08.2008, http://www.artima.com/weblogs/viewpost.jsp?thread=235725



Rys.3.2.A. Władza społeczności w poszczególnych rejonach

Przykładowe projekty

Jak wcześniej wspomniano, struktura oparta na życzliwych dyktatorach jest jedną z najpopularniejszych wśród mniejszych i początkujących projektów *open source,* choć zarazem często społeczności pozostają przy tej formie organizacji, nawet wówczas, gdy projekt rozwinie się do większych rozmiarów. Jednymi z najbardziej rozpoznawalnych projektów zorganizowanych w ten sposób to:

- Python – wysokopoziomowy język programowania ogólnego przeznaczenia, którego rozwojem przewodzi Guido van Rossum,
- Blender – aplikacja do modelowania 3D oraz reytraceing'u, której rozwojowi przewodzi Ton Roosendaal,
- Slackware – jedna z najstarszych dystrybucji systemu Linux, którą utrzymuje Patrick Volkerding.

3.3. Społeczności merytokratyczne

Społeczność merytokratyczna to struktura organizacyjna, wewnątrz której wykorzystywany jest formalny system głosowania w sprawach, w których społeczność nie mogła dojść do konsensusu. Mimo że głosowania stanowią istotny element tej struktury, społeczności starają się stosować je najrzadziej jak to tylko możliwe[POS]. W związku z tym tego typu forma organizacji jest często nazywana demokracją opartą na konsensusie. Należy jednak zauważyć, że określenie „demokracja” jest w tym przypadku lekkim nadużyciem, jako że nie wszystkie oddane głosy muszą mieć równą wagę, a i nie każdy członek projektu może brać udział w głosowaniu. Społeczność kierująca się zasadami merytokracji musi mieć ściśle zdefiniowany zestaw zasad określających, kto może głosować, a także jaką wagę posiada głos danej osoby w danym temacie. Głównym celem tych zasad jest dowartościowanie w największym stopniu głosów od tych osób, które mogą wnieść najwięcej wartości merytorycznej do podejmowanej problematyki. Aby to osiągnąć, prawo do głosu zazwyczaj mają osoby biorące aktywny udział w projekcie, a waga ich głosu jest zależna od recenzji włożonej przez nich pracy. Ta sama idea powinna dotyczyć głosowań zarówno w decyzjach technicznych, jak i personalnych, podczas wybierania przywódców projektu lub awansowania poszczególnych członków społeczności.

Dosyć subtelną kwestią jest to, w jakich okolicznościach głosowanie powinno być wykonane, jako że zazwyczaj niewiele sytuacji w rzeczywistości tego wymaga. Głosowanie nie powinno być uważane za stosowny mechanizm do rozstrzygania codziennych debat, ponieważ kończy ono nie tylko samą dyskusję, ale i też kreatywne myślenie nad rozwiązaniem omawianego problemu. Dopóki dyskusja jest kontynuowana, dopóty istnieje możliwość, że ktoś zasugeruje rozwiązanie, na które wszyscy się zgodzą. Nawet, gdy nie znajdzie się żadna powszechnie akceptowalna opcja, często lepszym rozwiązaniem jest próba dojścia do kompromisu, aniżeli nadmierne zarządzanie głosowania. Zgoda na kompromis pozostawia wszystkich uczestników sporu z małą dozą nieusatysfakcjonowania. Mimo wszystko jest to lepsza sytuacja niż w przypadku rozstrzygnięcia głosowania, w którym głosujący są podzieleni na przegranych i wygranych. Częste głosowania mogą zniechęcić niektórych programistów do projektu, z tego względu, iż po kilku przegranych sondach mogą stracić oni poczucie więzi ze społecznością oraz poczucie dążenia do wspólnego celu. Jednak poza tymi negatywnymi aspektami głosowanie daje

ostateczną odpowiedź w trudnych tematach i pozwala całemu projektowi iść naprzód. Dlatego też należy używać tego narzędzia tylko wtedy, gdy nie ma już żadnych szans na osiągnięcie konsensusu, a niepodjęcie decyzji będzie miało negatywny wpływ na postępy w ramach projektu.

Następnym ważnym aspektem jest prawo oddania głosu. Zasady, które określają sposób doboru osób uprawnionych do głosowania, powinny promować tych członków społeczności, których opinie są najbardziej znaczące w danym temacie. Powinno się jednocześnie unikać ludzi, którzy mogą nie być w nim biegli, a własną decyzję oparliby w większym stopniu na osobistych uprzedzeniach, niż na technicznych faktach. Jednym z uproszczonych sposobów na osiągnięcie tego jest dopuszczenie do głosu ludzi, którzy brali aktywny udział w danej dyskusji. Nieco bardziej skuteczną metodą jest przyznawanie głosu tym członkom projektu, którzy wykonali stosowną ilość pracy w danym obszarze. Na przykład jeżeli dyskusja tyczy się danego modułu w systemie, można by wybrać głosujących spośród ludzi którzy otrzymali pełen dostęp do repozytorium owego modułu. Należy pamiętać, że zgodnie z zasadami tej struktury organizacyjnej, decyzja o przyznaniu pełnego dostępu do repozytorium musi być również podjęta poprzez konsensus lub głosowanie. Kolejnym rozszerzeniem zasad doboru głosujących może być przydzielenie wybrańcom różnych wag głosu. Jest to element trudny do osiągnięcia, jako że potrzebne są do tego narzędzia, które przekalkulują realny wkład pracy danego członka projektu i zaprezentują go pod postacią liczby. Można to osiągnąć, na przykład, poprzez dodanie systemu zbierającego oceny procesu recenzowania kodu. Innym ciekawym narzędziem są dane udostępniane przez serwis Ohloh.net, który monitoruje zaangażowanie zarejestrowanych programistów w niemal 20 000 projektów *open source*. Ostatnią rzeczą, jaką należy zrobić po zebraniu głosów od wcześniej wybranych osób oraz po ewentualnym obliczeniu ich wag, jest zanotowanie ostatecznej decyzji oraz przebiegu procesu jej uzyskania. Każdy projekt powinien mieć ogólnodostępny rekord wszystkich decyzji podjętych w ten sposób.

Modele biznesowe

Firmy zaangażowane w projekty sterowane merytokratycznie zazwyczaj realizują w ten sposób cele biznesowe nie zapewniające bezpośrednich zysków. Przykładami tego może być obniżanie kosztów wytwarzania wewnętrznych narzędzi lub realizacja strategii *loss-leader* (patrz roz. 4.6). Same projekty są natomiast często

zarządzane przez utworzone specjalnie pod nie fundacje, które utrzymują się z dotacji (patrz roz. 4.5). Ponieważ w środowisku merytokratycznym, władza jest znacznie bardziej rozproszona i płynna, trudniej jest zbudować efektywny model biznesowy na bazie produktów wypływających z tego typu projektów. Choć jednocześnie należy zauważyć, iż zdarzają się przykłady *cross-sellingu* i *up-sellingu* bazujące na oprogramowaniu wytwarzanym w ten właśnie sposób (patrz roz. 4.3). Jednym z nich są produkty klasy MOM (ang. *message orientated middleware*) firmy Fuse, bazujące na projekcie Apache ActiveMQ.

Przykładowy projekt

Jedną z modelowych organizacji prowadzonych w sposób merytokratyczny jest fundacja Apache, której projekty są nakierowane głównie na rozwój narzędzi programistycznych oraz *middleware*. Nazwa Apache utożsamiana jest zazwyczaj z nazwą serwera HTTP, nad którym prace rozpoczęły się w roku 1994. Oficjalnie Apache Software Foundation została założona w roku 1999 właśnie wokół tego projektu. Do dnia dzisiejszego w jej ramach zrzeszonych jest kilkadziesiąt większych projektów, w tym wiele szeroko stosowanych narzędzi programistycznych opartych na Javie, jak chociażby Apache Ant, Apache Commons, log4j, Tomcat, Xerces, czy też własna implementacja Javy SE o nazwie Harmony.

Samą fundację można postrzegać jako projekt parasolowy (patrz roz. 3.7), który ogranicza swoją rolę jedynie do zapewniania infrastruktury projektom prowadzonym w sposób merytokratyczny, a także których produkty wpasowują się w obecne portfolio oprogramowania marki Apache. Projekty pod szyldem powyższej organizacji mają narzucany obowiązek licencjonowania wytwarzanego oprogramowania na zasadach licencji Apache. ASF za główne cele stawia sobie[16]:

- zapewnianie fundamentów pod otwarte projekty, w postaci infrastruktury sprzętowej, komunikacyjnej oraz biznesowej;
- stworzenie niezależnej osobowości prawnej, dla której firmy oraz poszczególne osoby mogą ofiarować swoje zasoby i być pewnymi, że zostaną one wykorzystane dla dobra publicznego;

16 How the ASF works - oficjalny artykuł od Apache Foundation, http://www.apache.org/foundation/how-it-works.html

- zapewnienie środków umożliwiających ochronę poszczególnych wolontariuszy od oskarżeń prawnych kierowanych przeciwko projektom fundacji;
- ochrona przed nadużyciami marki „Apache”.

Fundacja Apache ma charakter zdecentralizowany, a prowadzone w jej ramach projekty są ze sobą stosunkowo luźno powiązane. Jednak w przeciwieństwie do innych organizacji, zapewniających infrastrukturę projektom *open source*, ASF wymaga, aby prawa intelektualne do oprogramowania zostały w pełni przekazane w ręce fundacji, zanim stanie się ono jej oficjalnym projektem. W dodatku świeżo przyjęte projekty są z reguły oznaczane jako „inkubowane”, zanim społeczność zdecyduje, że osiągnięty został odpowiedni poziom dojrzałości.

Patrząc globalnie na całą fundację można podzielić jej społeczność na następujące grupy:

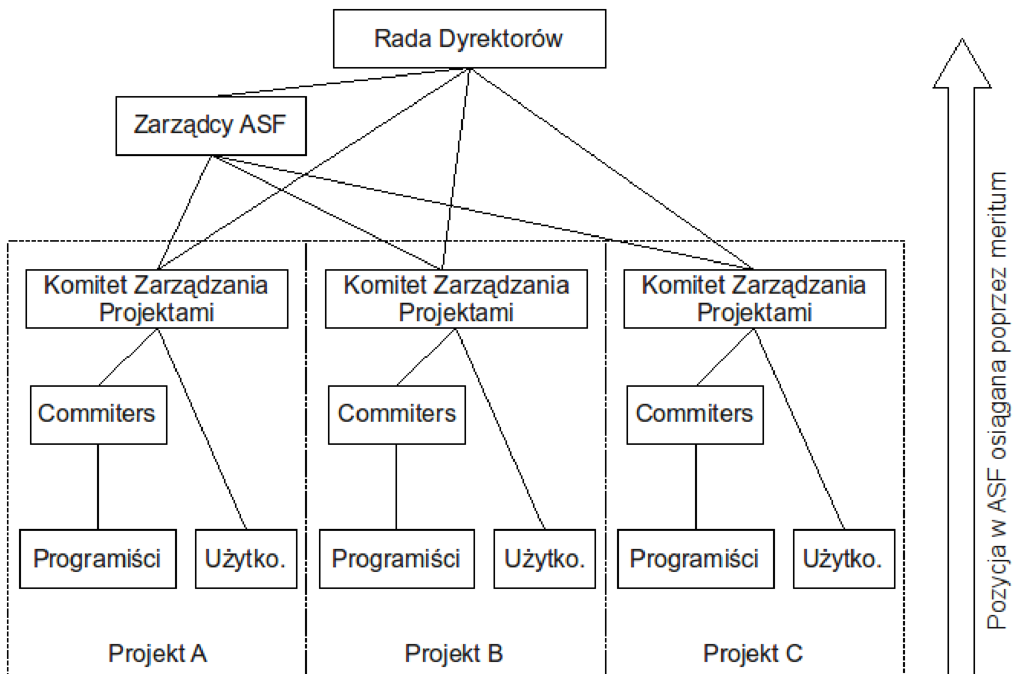
- **Użytkownicy**, kategoryzowani następnie jako:
  - *aktywni*, czyli zapewniający projektowi wartościowe sprzężenie zwrotne (ang. *feedback*) oraz wspierający innych użytkowników;
  - *pasywni*, ograniczający się jedynie do użytkowania oprogramowania, określa się ich często słowem zaczerpniętym z języka angielskiego - *lurkers.*
- **Programiści**, którzy ofiarują projektom swoją pracę.
- **Commiters**[17], którzy są programistami posiadającymi bezpośredni dostęp do repozytoriów kodu. Programiści bez tego statusu są poddawani obowiązkowemu recenzowaniu kodu zanim trafi on do repozytoriów.
- **Członek PMC -** programista, który poprzez swoje walory merytoryczne wykazane w danym projekcie, został wybrany do zarządzenia nim w ramach **Komitetu Zarządzania Projektem** (ang. *Project Managment Committtee*, w skrócie PMC).

17 Brak polskiego tłumaczenia. Bezpośrednim odpowiednikiem byłby wyraz *ofiarodawcy,* jednak nie oddaje on w pełni znaczenia, z jakim utożsamiane jest określenie *commiters.*

- **Członek ASF -** osoba nominowana przez obecnych członków i wybierana na podstawie wkładu merytorycznego w rozwój fundacji. Członkowie fundacji zajmują się rozwojem fundacji jako takiej.
- **Zarządcy ASF** zajmujący się codziennymi sprawami fundacji. Zarządcy są wybierani przez **Radę Dyrektorów**.

Centralnym punktem tej struktury organizacyjnej jest Rada Dyrektorów, która składa się z 9 członków ASF wybieranych corocznie przez ową grupę. Jest ona odpowiedzialna za zarządzanie oraz nadzór nad przedsięwzięciami korporacji, zgodnie z zasadami fundacji, w tym także rozporządzanie jej własnościami oraz alokacją zasobów pomiędzy poszczególnymi projektami.

Rada nie bierze natomiast znaczącego udziału w sprawach technicznych oraz związanych z wyborem kierunku rozwoju danego projektu. Za tego typu kwestie odpowiedzialne są poszczególne Komitety Zarządzania Projektem, co wprowadza wspomnianą wcześniej decentralizację władzy. Komitety są ustanawiane przez Radę w celu aktywnego zarządzania jedną lub kilkoma społecznościami projektowymi. W skład PMC zawsze wchodzi jeden zarządca ASF, sprawujący rolę przewodniczącego, oraz zazwyczaj jeden lub kilku członków ASF. Przewodniczący jest mianowany przez Radę, a zarazem jest przed nią odpowiedzialny. W jego gestii leży misja zwerbowania reszty komitetu oraz istota ustanowienia reguł i procedur codziennego zarządzania danym projektem. Całościowy pogląd na społeczność Apache przedstawia poniższy rysunek 3.3.A.



Rys.3.3.A. Struktura społeczności Apache

Wszystkie projekty składają się z ochotników i żaden z nich, nawet członkowie ASF oraz zarządcy, nie są opłacani przez samą fundację. Wielu członków społeczności Apache jest natomiast opłacanych przez firmy czerpiące korzyści z rozwoju oprogramowania fundacji. Kilka projektów powstało nawet z oprogramowania darowanego na rzecz fundacji, jak na przykład Apache Derby z bazy danych Couldspace od IBM.

Projekty są zazwyczaj samozarządzane przez ludzi, którzy poświęcają im swój czas. W fundacji często określa się to mianem *do-ocracy*, czyli władza tych, którzy „robią” (*ang. do*). Polega to na tym, że poszczególni programiści podejmują tym śmielej samodzielne decyzje, im większy jest ich procentowy wkład w danym obszarze projektu. W sytuacji, gdy potrzebna jest konsolidacja z resztą zespołu, stosuje się wówczas różne techniki podejmowania decyzji. Jedną z nich jest tzw. „leniwy konsensus”, który wykorzystuje się najczęściej w trakcie podejmowania codziennych decyzji przez małe grupy osób. W tym podejściu opis decyzji wymagającej konsolidacji jest publikowany na liście mailingowej danego projektu.

Wolontariusze wedle własnego uznania wystawiają danemu żądaniu pozytywne, negatywne lub zerowe głosy. Liczba kilku pozytywnych głosów, bez żadnego negatywnego, wystarczy aby uznać dane żądanie jako zaakceptowane. Brak żadnego głosu przez dłuższy okres czasu jest także uznawane za tzw. *cichą akceptację*. Każdy negatywny głos blokuje natomiast decyzję. Reguły fundacji wymagają w tym wypadku, aby osoba negująca zawsze dołączała do swojego głosu szczegółowe wyjaśnienie wątpliwości lub propozycję alternatywnego rozwiązania. W czasie, gdy żądanie jest zablokowane, jego autor ma czas na przekonanie właścicieli negatywnych głosów do swoich racji. Jeżeli uda mu się ta sztuka wobec wszystkich negujących, decyzja zostanie przyjęta. W przypadku wybrania propozycji alternatywnego rozwiązania, podlega ono nowemu głosowaniu.

Jak już nadmieniliśmy tzw. *leniwy konsensus* jest częstą formą podejmowania pomniejszych decyzji w fundacji. Jednak to, jakie techniki wypracowywania decyzji mogą zostać podjęte w danej kwestii, zależy od jej kontekstu. W fundacji wyróżniamy trzy rejony podejmowania decyzji[18]:

- **Modyfikacje kodu:** decyzje są podejmowane na ogólnych zasadach leniwego konsensusu. Zasada cichej akceptacji jest stosowana, jeżeli autor modyfikacji tego sobie zażyczy. Bez niej wymagane są trzy pozytywne głosy, bez żadnego negatywnego, aby zatwierdzić zmianę.
- **Proceduralne:** decyzje te z reguły sprowadzają duże grono zainteresowanych i są rozstrzygane normalną większością głosów. W przypadku, gdy okaże się, że grupa głosujących jest mała i niereprezentatywna, wówczas można wprowadzić zasadę leniwego konsensusu.
- **Wydanie produktu:** decyzja dotycząca wypuszczenia kolejnego wydania produktu opracowywanego w ramach projektu podlega nieco inny zasadom, które są opisane poniżej.

W systemie głosowań fundacji Apache istnieje jeszcze pojęcie tak zwanych *głosów wiążących*. Głosy tego rodzaju należą jedynie do członków PMC i mają zastosowanie przy podejmowaniu kluczowych decyzji, jak np. ogłoszenie kolejnego wydania. W sytuacji, gdy członkowie PMC chcą skorzystać z tego przywileju,

18 Consensus Gauging through Voting, oficjalny artykuł fundacji Apache, http://www.apache.org/foundation/voting.html

pozostali członkowie projektu są często proszeni o powstrzymanie się od głosowania, aby ograniczyć szum w temacie lub ich głosy są traktowane jedynie jako element doradczy. Decyzja o ogłoszeniu kolejnego wydania wymaga obowiązkowo przynajmniej trzech pozytywnych głosów wiążących.

W świecie *open source* istnieje kilka innych organizacji wzorowanych na strukturze Apache Software Foundation. Jedną z nich jest fundacja Eclipse założona przez IBM w 2001 roku. Wyróżnia ją nieco bardziej sformalizowany proces wytwórczy oraz większy stopień ingerencji w projekty wykonywane pod jej patronatem.

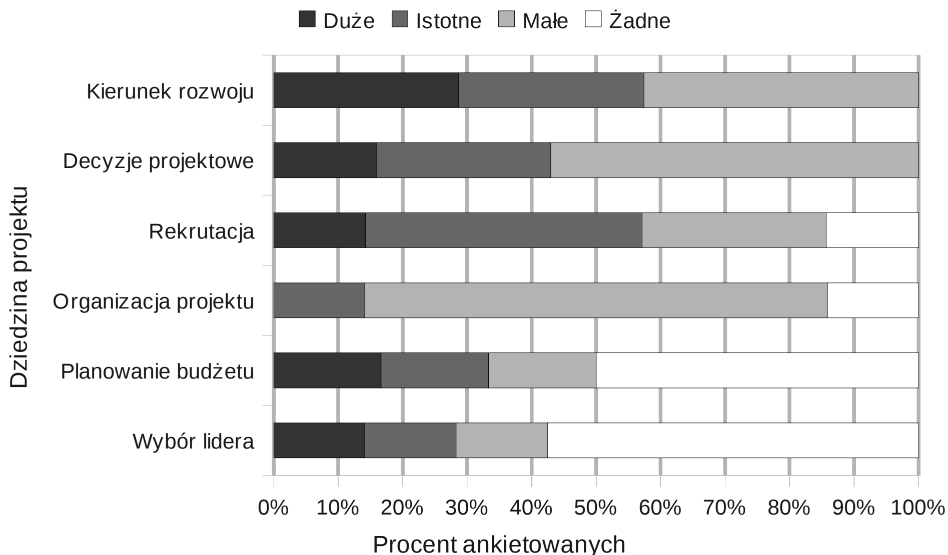
3.4. Proces wytwórczy wewnętrzny, sprzężenie zwrotne w społeczności

Określenie proces wytwórczy wewnętrzny, sprzężenie zwrotne w społeczności odnosi się do struktury organizacyjnej, w której istnieje wyraźny podział pomiędzy społecznością a zespołem projektowym. W języku angielskim stosuje się termin *in-house development, community feedback*. Oznacza to zazwyczaj, że większość prac rozwojowych jest wykonywanych przez pracowników opłaconych przez firmę fundującą danych projekt, a społeczność wokół niego pełni raczej rolę doradczą oraz wspiera proces zapewniania jakości oprogramowania. Tego rodzaju struktura jest kombinacją otwartych oraz zamkniętych metodyk wytwarzania oprogramowania. Według badań ankietowych[DA], ponad 30% projektów jest organizowanych właśnie w ten sposób, niezależnie od ich wielkości. W trakcie, gdy zarządzanie zespołem programistycznym odbywa się w owej strukturze według bardziej tradycyjnych metodologii, zarządzanie produktem jest z reguły prowadzone w sposób otwarty. Wymuszają to wpływy ogólnej kultury organizacyjnej ruchu *open source*, takie jak np. wyższe zaangażowanie użytkowników w proces wytwórczy oraz możliwość „sforkowania” projektu (patrz roz. 3.1 i 3.6). W związku z tym menedżerowie produktu muszą być w mniejszym stopniu autorytarni podczas określania kierunków rozwoju. Ponadto muszą być przygotowani na proces przekonywania społeczności do swoich pomysłów oraz na spontaniczne kontrybucje nowych funkcjonalności.

Jednym z kluczowych elementów, w którym społeczność bierze aktywny udział w omawianej strukturze organizacyjnej, są testy wczesnych wydań

rozwojowych. Jest to ważny aspekt *open source*, który wedle wcześniejszych spostrzeżeń wspierany jest poprzez automatyzację generowania wczesnych wydań oraz poprzez traktowanie swoich użytkowników jako współpracowników (patrz roz. 3.1). Dobrze stymulowana społeczność beta-testerów może stać się najwartościowszym zasobem każdego projektu[CatB]. Wielu projektom udaje się też pozyskać tzw. użytkowników świadomych kodu (ang. code-aware users). Jest to grupa użytkowników ze zdolnościami programistycznymi, którzy w razie napotkania błędu, są w stanie przeanalizować go na poziomie kodu lub nawet dostarczyć gotową poprawkę. Suma wkładu mniej lub bardziej zaawansowanych członków społeczności projektowej może gwałtownie zmniejszyć czas potrzebny na wprowadzenie kolejnego wydania produktu na rynek, wraz z zachowaniem stabilnego poziomu jakości[POS].

Na rysunku 3.4.A przedstawiono, zgodnie z wynikami ankiet[DA], poziom swobody dany społeczności w poszczególnych rejonach projektu. Zawarta jest w nim informacja, jaki procent administratorów projektu, wykorzystujących opisywaną w tym rozdziale strukturę organizacyjną, uważa swobodę społeczności za: dużą, istotną, małą lub żadną, w określonej dziedzinie swojego projektu.



Rys.3.4.A. Władza społeczności w poszczególnych rejonach

Porównując te wyniki ze strukturą zbudowaną wokół życzliwych dyktatorów z rozdziału 3.2 można zauważyć, że poziom władzy społeczności jest tutaj nieco niższy.

Modele biznesowe

Struktury organizacyjne z wewnętrznym procesem rozwojowym oraz sprzężeniem zwrotnym w społeczności są często wykorzystywane przez firmy, których większość produktów jest wytwarzanych w metodologii otwartej. Zazwyczaj firmy te stosują modele biznesowe oparte na podwójnym licencjonowaniu oraz zapewnianiu usług powiązanych, aby spieniężyć włożoną w nie pracę (patrz roz. 4.2 i 4.4). Realizują to poprzez dystrybucję dwóch wariantów ich oprogramowania: otwartej dla projektów *open source* oraz na komercyjnej licencji, dostarczając dodatkowe prawa lub usługi. Wiele spośród firm, które zajmują się wytwarzaniem zarówno oprogramowania open source, jak i własnościowego, stosuje względem nich *cross-selling* oraz *up-selling* (patrz roz. 4.3). Istnieją także inne cele biznesowe, które idą w parze z taką strukturą organizacyjną. Są one opisane w rozdziale 4.6.

Dzięki elastyczności w sferze biznesowej, fundatorzy projektów często wybierają opisywaną w tym rozdziale strukturę kosztem ograniczenia korzyści w domenie procesu wytwórczego. Szacuje się, że niemal 50% dochodów wygenerowanych przez produkty *open source* będzie pochodziło z tej formy organizacji[19]. Projektami często utożsamianymi ze strukturą na zasadzie wewnętrznego procesu rozwojowego i sprzężenia zwrotnego w społeczności są: baza danych MySQL i Qt toolkit.

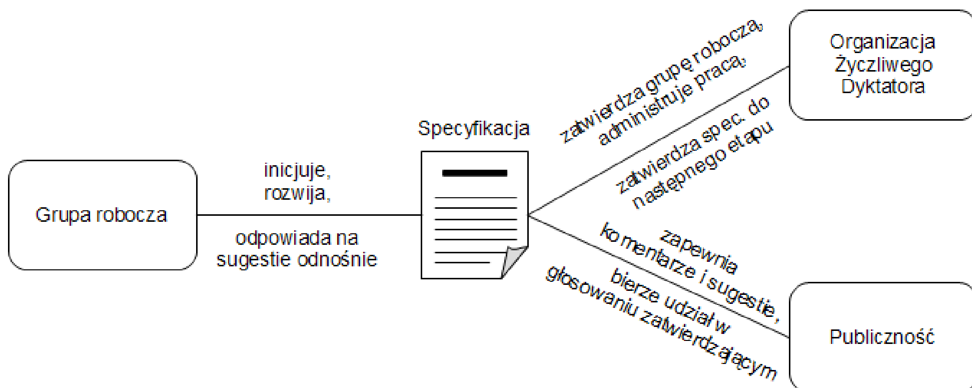
3.5. Proces społecznościowy oparty na specyfikacjach

Proces społecznościowy oparty na specyfikacjach jest formalną strukturą organizacyjną, nastawioną na tworzenie specyfikacji produktów informatycznych. Zachodzi to w myśl ściśle zdefiniowanej formuły, która jest otwarta na publiczne recenzje oraz komentarze. Dwoma najbardziej znanymi procesami tego typu są Java Community Process (w skrócie JCP), zajmująca się specyfikacjami dla platformy Java, oraz Internet Standards Process, zajmujący się standaryzacją protokołów

19 Gartner, Inc. “Predicts 2009: The Evolving Open Source Model.”, 2008.

internetowych. Członkowie społeczności o takiej formie organizacji dzielą się z reguły na trzy grupy względem pełnionych przez siebie funkcji:

- Organizacja pełniąca funkcję życzliwego dyktatora – ogólnie poważany autorytet, który administruje procesem tworzenia standardów przemysłowych w danej dziedzinie.
- Grupy robocze konkretnych specyfikacji – stanowiące grupy ochotników zatwierdzonych przez dyktującą organizację do pracy nad konkretną specyfikacją.
- Publiczność – ludzie zainteresowani procesem tworzenia danej specyfikacji, którzy zarazem wspierają pracę poprzez zgłaszanie własnych sugestii, a także biorą niekiedy udział w głosowaniu na rzecz przyjęcia, bądź też odrzucenia pewnej specyfikacji.



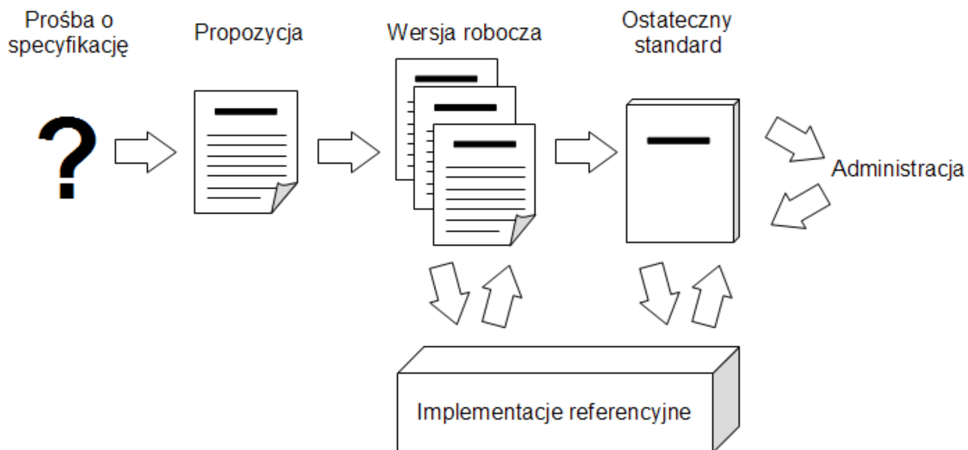
Rys.3.5.A. Podział ról w kontekście specyfikacji

Jak widać, struktura organizacyjna w postaci procesu społecznościowego opartego na specyfikacjach jest w pewnym stopniu podobna do społeczności z życzliwymi dyktatorami oraz społeczności merytokratycznych (patrz roz. 3.2 i 3.3). Organizacja, która administruje procesem opisywanym w tym rozdziale nie posiada wyłącznych praw do tworzenia danego typu specyfikacji. Dlatego też musi liczyć się z możliwością powstania „forka” na bazie jej dzieła (patrz roz. 3.6). Organizacja pełniąca rolę życzliwego dyktatora musi ciągle zatwierdzać swój autorytet w danej

dziedzinie poprzez wartość swoich decyzji, przy wsparciu umiejętności dochodzenia do konsensusu z zainteresowanymi stronami. We wcześniej wspominanym JCP rolę życzliwego dyktatora pełni Program Managment Office mianowany przez Oracle, natomiast w Internet Standards Process autorytetem jest Internet Engineering Steering Group. Podczas pracy nad specyfikacją definicja procesu zakłada przejście przez kilka etapów realizacji. Decyzja o przejściu do następnego etapu jest podejmowana w formie merytokratycznego głosowania. Największa wagę głosu mają zazwyczaj osoby reprezentujące organizację życzliwego dyktatora oraz najbardziej aktywni członkowie społeczności.

Przebieg procesu

Proces tworzenia specyfikacji przebiega według ogólnego schematu przedstawionego na poniższym rysunku 3.5.B. Pod nim znajduje się opis poszczególnych kroków procesu.



Rys.3.5.B. Podział ról w kontekście specyfikacji

1. Prośba o specyfikację – publiczność sygnalizuje potrzebę standaryzacji pewnego tematu.
2. Propozycja – pomysł na specyfikację jest wysłana przez najbardziej zainteresowane podmioty do organizacji pełniącej rolę dyktatora.
3. Wersja robocza – utworzona zostaje grupa robocza zajmująca się się pracami nad roboczą wersją specyfikacji poprzez jej uszczegóławianie, publikowanie aktualnej wersji oraz reagowanie na opinie i sugestie publiczności.
4. Ostateczny standard – kiedy wersja robocza specyfikacji osiągnie określony poziom dojrzałości, członkowie organizacji dyktatora oraz publiczność mogą podjąć decyzję o mianowaniu jej ostatecznym standardem.
5. Administracja – część grupy roboczej, która administruje specyfikacją poprzez zatwierdzanie drobnych poprawek oraz utrzymywaniu erraty.

Podczas opracowywania wersji roboczej, a także już po utworzeniu ostatecznego standardu, wiele firm trzecich opracowuje tzw. implementacje referencyjne. Stanowią one zalążek przyszłego produktu oraz służą do weryfikacji rzeczywistej jakości samej specyfikacji.

Przykład Java Community Process

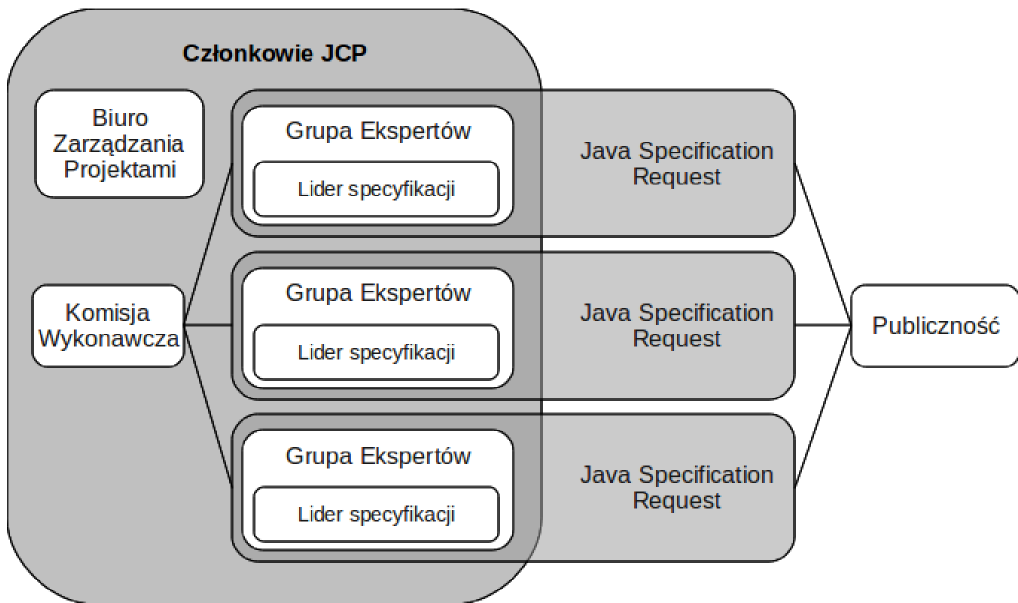
Java Community Process, zwany w skrócie JCP, powstał w roku 1998 jako sformalizowany proces współpracy między zainteresowanymi podmiotami nad specyfikacjami przyszłych wydań platformy Java. Program ma charakter otwarty, co oznacza, że każdy może swobodnie opiniować powstające w jego ramach specyfikacje oraz ubiegać się o członkostwo. Członkowie JCP mogą natomiast za zgodą władz JCP opracowywać własne pomysły na usprawnienie platformy. Współpraca ta opiera się na formalnych dokumentach nazywanych Java Specification Requests, w skrócie JSR, które są poddawane publicznej recenzji. Specyfikacja jest ogłaszana jako ostateczna, gdy osiągnie poziom dojrzałości zgodny z oczekiwaniami Komisji Wykonawczej JCP. Zanim tak się jednak stanie, wymagane jest często, aby grupa pracująca nad daną specyfikacją dostarczyła także jej implementacji referencyjnej oraz narzędzi do weryfikacji kompatybilności innych implementacji. Sam proces JCP jest konstruowany jednocześnie według zasad, które opisuje. Obecna jego wersja opisana w JSR-215 to 2.6[20].

Społeczność biorąca udział w procesie JCP podzielona jest na kilka grup i patrząc na nie od góry hierarchii jej uczestnicy prezentują się następująco[21]:

- **Biuro Zarządzania Programem** (ang. *Program Managament Office*, w skrócie **PMO**) - stanowi grupę z wewnątrz, poprzednio Sun Microsystems, a obecnie Oracle, która jest odpowiedzialna za administrację JCP oraz przewodnictwo w Komisji Wykonawczej,
- **Komisja Wykonawcza** (ang. *Executive Committee*, w skrócie **EC**) - grupa członków JCP, która kieruje rozwojem platformy Java jako całością; składa się z reprezentantów największych podmiotów zainteresowanych rozwojem Javy oraz innych członków społeczności tej platformy.
- **Grupa Ekspertów** składająca ze specjalistów zaproszonych przez Lidera Specyfikacji (ang. *Spec Lead*), który sam jest jednocześnie członkiem tej grupy.

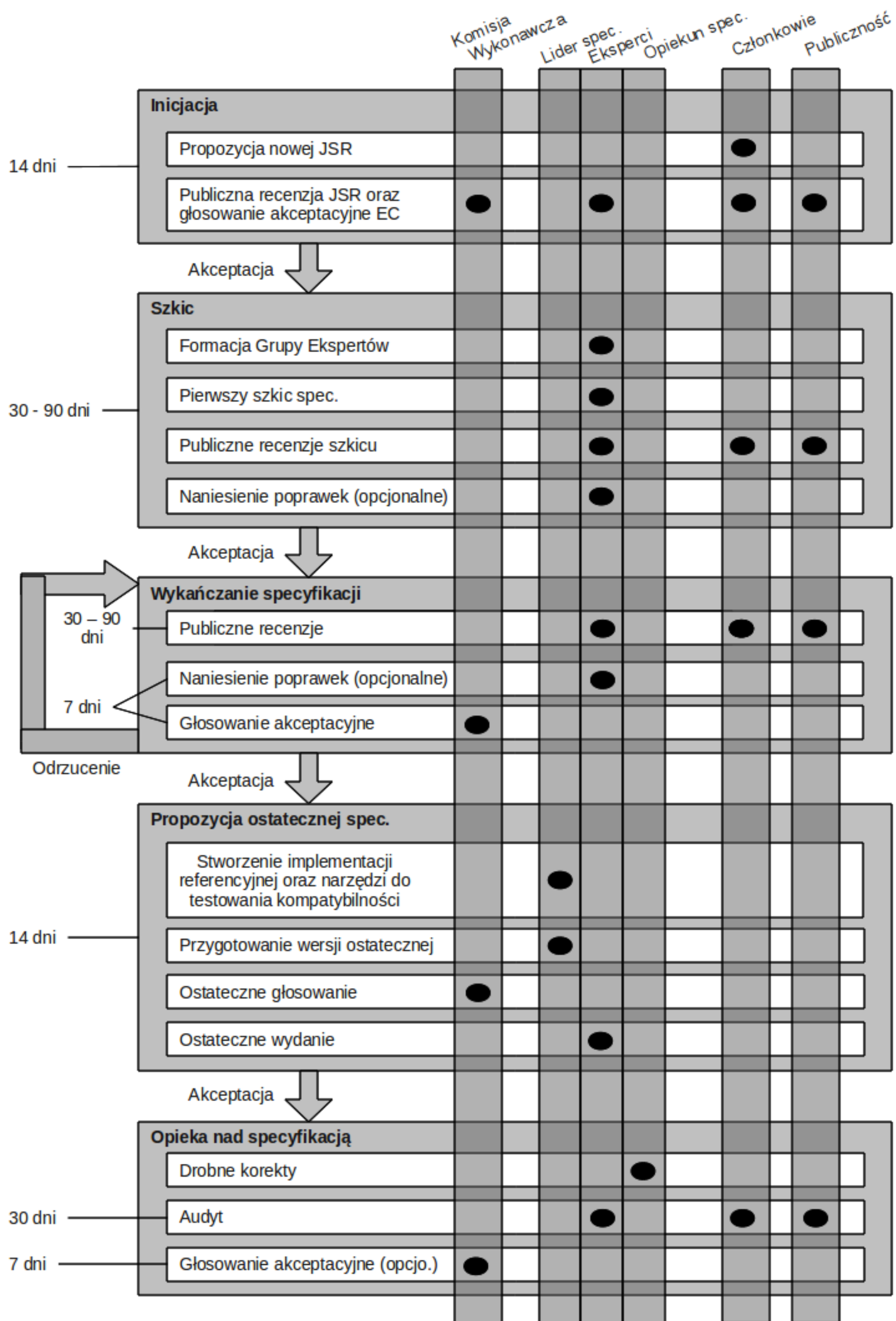
20 JSR 215: Java Community ProcessSM version 2.6, http://jcp.org/en/jsr/detail?id=215
21 JCP 2: Process Document - JCP Web site, Version 2.7, 15.05.2009, http://jcp.org/en/procedures/jcp2

- **Członkiem JCP** mogą być firmy, organizacje oraz pojedyncze osoby, które zgłosiły się do programu oraz podpisały *Java Specification Participation Agreement*.
- **Publiczność**, czyli każdy kto nie jest członkiem, ale wykazuje zainteresowanie daną JSR oraz udzielił swoich opinii lub sugestii w trakcie trwania procesu.



3.5.C. Struktura społeczności JCP

Przebieg procesu JCP jest skoncentrowany wokół Grupy Ekspertów i dzieli się na 5 etapów. Decyzje o przejściu z jednego etapu do kolejnego podejmuje Komisja Wykonawcza. Ideowy schemat całego procesu przedstawia poniższa ilustracja rys. 3.5.D.



3.5.D. Etapy JCP

Poszczególne etapy przebiegają w następujący sposób:

1. **Inicjacja**. Jedna osoba lub grupa członków JCP zgłasza żądanie utworzenia zupełnie nowej lub kolejnej wersji już istniejącej specyfikacji do PMO. Po akceptacji żądanie specyfikacji jest publikowane zgodnie ze standardem JSR oraz poddane do wewnętrznej recenzji przez okres od 2 do 4 tygodni. W tym czasie wybrane grupy zainteresowanych mogą brać udział w dyskusji poprzez nowo utworzoną listę mailingową. Członkowie mogą też w tym czasie ubiegać się o przydzielenie do Grupy Eksperckiej związanej z daną specyfikacją. Podczas ostatnich 14 dni recenzowania JSR głosowanie wewnątrz Komisji Wykonawczej decyduje o akceptacji i przejściu do kolejnego etapu. W przypadku braku akceptacji zespół zajmujący się specyfikacją ma 14 dni na uwzględnienie zastrzeżeń zgłoszonych przez komisję oraz na zgłoszenie JSR do kolejnego głosowania.

2. **Szkic specyfikacji.** Po zaakceptowaniu przez Komisję Wykonawczą eksperci powinni rozpocząć pracę nad uwzględnieniem wszelkich dostarczonych wymagań, dokumentów, opisów technologii oraz opinii w celu zrewidowania obecnego kształtu specyfikacji. Jest to najlepszy czas na konsultacje z publicznością, w tym także z firmami oraz środowiskami akademickimi, zainteresowani tematem danej JSR. Głównym celem tego etapu jest nakreślenie wymagań, jakie będą stawiane przed ostateczną wersją specyfikacji. Okres ten trwa od 30 do 90 dni i jest cyklicznie powtarzany. Każdy taki cykl kończy się głosowaniem akceptacyjnym w PMO nad kolejną rewizją specyfikacji.

3. **Wykańczanie specyfikacji.** Na tym etapie specyfikacja jest oficjalnie ogłaszana oraz poddawana publicznej krytyce. Zarówno członkowie, jak i publiczność, mogą zgłaszać opinie oraz sugestie na oficjalnej liście mailingowej. Lider Specyfikacji jest odpowiedzialny za rozpatrywanie wszystkich uwag oraz prowadzenie dialogu z publicznością. Etap ten jest cyklicznie powtarzany, a jeden jego cykl trwa od 30 do 90 dni. Przed końcem każdego cyklu specyfikacja jest zamrażana na 7 dni i wysyłana do PMO w celu weryfikacji naniesionych zmian. Etap ten kończy wynik publicznego głosowania wszystkich stron, które brały udział w pracach i opiniowaniu specyfikacji.

4. **Propozycja ostatecznej wersji specyfikacji.** Po publicznej akceptacji grupa ekspertów przygotowuje propozycję finalnej wersji specyfikacji. Wersja ta powinna zostać wysłana przez lidera do PMO, które oficjalnie ogłosi ją na łamach JCP i podda pod kolejną publiczną recenzję. Ponadto w tym okresie wymagane jest od Lidera Specyfikacji wykonanie implementacji referencyjnej oraz narzędzi do testowania kompatybilności. Podczas tych prac zazwyczaj wykrywane są obszary specyfikacji, które nie zostały opisane w odpowiedni sposób. Lider, wraz z grupą ekspertów, musi doprecyzować wykryte braki oraz regularnie przysyłać sprawozdanie zmian do PMO. Kiedy eksperci uznają, że narzędzia do testowania kompatybilności pokrywają odpowiedni obszar specyfikacji, a sama implementacja referencyjna zdaje owe testy, całe oprogramowanie wraz z niezbędnymi dokumentami jest wysyłane do PMO. O uczynieniu nadesłanych materiałów ostatecznym standardem decyduje głosowanie w ramach Komisji Wykonawczej. W przypadku odrzucenia grupa ekspertów ma 30 dni na uwzględnienie zastrzeżeń komisji oraz poddanie JSR pod kolejne głosowanie. Natomiast jeżeli specyfikacja zostanie zaakceptowana, staje się wtedy oficjalnym standardem platformy Java. Pracę grupy eksperckiej uznaję się za zakończoną, a Lider Specyfikacji jest z reguły mianowany jej opiekunem i sprawuję nad nią pieczę w ostatnim etapie.

5. **Opieka nad specyfikacją.** Opiekun specyfikacji jest odpowiedzialny na tym etapie za utrzymywanie erraty oraz uwzględnianie żądań o klaryfikację, interpretację lub dodanie drobnych usprawnień. W przypadku większych zmian wymagane jest utworzenie nowego JSR.

3.6. Fork

„*Fork*”[22] jest terminem z języka angielskiego określającym sytuację, gdy programiści postanawiają wykorzystać źródła jednego projektu do rozpoczęcia na ich podstawie nowego przedsięwzięcia, wytwarzającego ich własny produkt. Otwarte oprogramowanie z definicji zezwala na tzw. „forkowanie”, bez potrzeby uzyskania zgody od jego pierwotnych twórców. Zjawisko to pojawiło się również kilka razy w świecie zamkniętego oprogramowania, jak na przykład łączenie kliku większych

22 Z racji braku odpowiedniego tłumaczenia terminu, używany jest on w spolszczonej wersji, z zastosowaniem polskiej deklinacji.

systemów Unix'owych przewodzonych przez zespoły odpowiedzialne za System V Release 4.0 i SunOS[23]. Jednak w przeciwieństwie do oprogramowania *open source*, licencjonowane „forki” oprogramowania własnościowego powstają zazwyczaj wskutek strategicznych porozumień pomiędzy firmami oraz często wiążą się ze sporymi opłatami licencyjnymi. Natomiast w otwartych projektach, jak już zostało wcześniej wspomniane, powstają one zazwyczaj wskutek buntu przeciwko administratorom projektu lub oddzielnej wizji rozwoju danego oprogramowania.

„Fork” jest utożsamiany z czymś negatywnym, jako że z reguły niepotrzebnie dzieli zasoby pracy między dwie inicjatywy. Chociaż, jak pokazała historia, jego powstanie nie musi zawsze wywierać negatywnego wpływu na oryginalny projekt. Często podzielonym społecznościom udaje się wypracować swoistego typu symbiozę i współpracować razem nad wspólnymi celami. Zdarzają się nawet przypadki, gdy projekty połączyły się z powrotem, po tym jak jeden z nich udowodnił swoje racje.

Eric Raymond określił w swoim eseju[CatB], że:

Najważniejszą cechą charakterystyczną forka jest to, że tworzy dwa współzawodniczące ze sobą projekty, które później nie mogą wymieniać kodu między sobą, dzieląc w ten sposób potencjał społeczności.

Obecnie punkt widzenia na to zjawisko uległ delikatnej zmianie. Forki są postrzegane jako złe, lecz nie dlatego, że mogą zmarnotrawić część pracy ich społeczności. Przy dużej liczbie uczestników projektów *open source* „forkowanie” ciągnie ze sobą dużą dozę konfliktów oraz walki w nowo powstającej społeczności o ich zasadność, cele i sukcesję władzy[24].

Relacje z projektem matką

W zależności od okoliczności oraz atmosfery w jakiej dana społeczność utworzyła „forka”, może ona mieć dwa różne stanowiska w stosunku do swojego projektu-matki. Najbardziej drastyczną postawą jest całkowite odcięcie

23 Several major computer and software companies announce strategic commitment to AT&T'S UNIX System V, Release 4.0. - informacja prasowa, Amdahl, Control Data Corporation, 18.11.1988, http://groups.google.com/group/comp.unix.questions/msg/2e02a599c5c62848

24 The Jargon File - version 4.4.7, http://www.catb.org/jargon/

i ustanowienie zupełnie niezależnego projektu. Dzieje się tak zazwyczaj wśród społeczności, które podzieliły się w trakcie podejmowania fundamentalnych decyzji technicznych lub personalnych. Łatwo jest ogłosić „forka” tego typu, ale wymaga to ogromnego wysiłku, aby utrzymać jego rozwój oraz wsparcie użytkowników. Bez adekwatnych zasobów, podobne inicjatywy mogą szybko zatracić swoją pierwotną aktywność, zasilaną początkowym buntem. Przykładem takiego projektu był „fork” projektu GNOME o nazwie GoneME (angielska gra słów, którą można przetłumaczyć jako „zniknąłem” lub „zniknij mnie”), wykonany przez ich byłego programistę[25]. Był on przeciwnikiem wielu decyzji podejmowanych przez administratorów projektu, zwłaszcza w zakresie utrzymywania nadmiernej prostoty GUI wytwarzanych tam aplikacji. Mimo że „fork” zyskał duży rozgłos i przyciągnął ku sobie sporo publiczności, nie minęło dużo czasu zanim aktywność w nim zamarła.

Istnieją jednak też takie „forki”, którym udało się odnieść sukces i uzyskać przewagę nad swoimi pierwotnymi projektami. Jednym z takich przykładów jest X.Org, który po „sforkowaniu” Xfree86 zyskał szerokie wsparcie ze strony programistów i znacznie przyspieszył rozwój serwera X11.

Innym ciekawym przykładem jest historia, w której „fork” okazał się być na tyle sprawniejszy, że administratorzy matczynego projektu poprosili „buntowników” o powrót do pierwotnych struktur. Mowa tu o EGCS (the Experimental/Enhanced GNU Compiler System) powstałym na bazie zarzutów, że ekipa GNU zarządza projektem GCC w zbyt „katedralny” sposób[CatB]. Mianem tym często są określane projekty zarządzane w tradycyjny sposób, z długimi cyklami wydawniczymi i ze ściśle kontrolowanym zespołem programistów. Administratorzy GCC uważali, że taka mniej otwarta forma wytwarzania oprogramowania powinna zachować odpowiednią jakość dla tego typu projektów. Jednak powstała społeczność wokół EGCS okazała się utrzymywać porównywalny poziom oraz rozwijać ich kompilatory znacznie szybciej niż miało to miejsce w ekipie GNU. Zespół pozostały przy GCC nie zignorował wniosków, jakie z tego płynęły, dlatego też postanowił połączyć rozdzielone społeczności z powrotem, pod pierwotną nazwą i przyjmując formę organizacji z projektu EGCS.

25 Strona Ali Akcaagac, inicjatora GoneME - http://www.akcaagac.com/index_goneme.html

Podobną historię ma również projekt Firefox, który rozpoczął swoje istnienie jako nieoficjalny projekt wewnątrz fundacji Mozilla i niedługo po swoim powstaniu przejął rolę Mozilla Suite, jako główny produkt fundacji.

Odmiennym podejściem są „forki", które od początku są budowane na zasadzie symbiozy z pierwotnym projektem. Powstają one zazwyczaj w sytuacji, gdy część społeczności ma wizję, która nie odbiega fundamentalnie od pierwotnego projektu, ale z różnych przyczyn nie ma szans na realizację wewnątrz jego struktur. „Forki" tego typu zakładają, że będą podążały za ogólnym tokiem rozwoju, ale będą jednocześnie niezależnie utrzymywać swoje rozszerzenia dostępne jedynie w ich produkcie. Wydaje się to być prostym rozwiązaniem, nie mniej jednak utworzenie pozytywnych relacji oraz efektywnego przepływu współpracy z pierwotnym projektem może okazać się sporym wyzwaniem. Jednym z rejonów, w którym się to najczęściej udaje, są dystrybucje oprogramowania (patrz roz. 3.8). Najlepszym przykładem tego zjawiska są setki dystrybucji systemów operacyjnych z rodziny GNU/Linux, z których niemal wszystkie są pochodną jednego z 12 czołowych projektów[26], takich jak: Debian, Slackware, Red Hat, Gentoo lub OpenSuse. Dzieje się tak prawdopodobnie dlatego, że zarządzanie dodatkowymi pakietami do macierzystych repozytoriów jest łatwiejszym zadaniem w porównaniu do utrzymania niezależnych łat lub gałęzi kodów źródłowych w repozytoriach. Wyjątkiem są tutaj sytuacje, gdy obydwa projekty rozwijają mniej więcej te same zespoły, które synchronizują daty kolejnych wydań oprogramowania. Jest to często spotykane w firmach up-selling'ujących produkty budowane na bazie własnych projektów *open source* (patrz roz. 4.3).

Przykład Debian – Ubuntu

Dobrym przykładem symbiotycznego „forka" są projekty Debian i Ubuntu. Debian jest dystrybucją systemu GNU/Linux, która z założenia ma być stabilnym systemem operacyjnym składającym się wyłącznie z wolnego oprogramowania. Cele zespołu z Ubuntu są jednak nieco inne. Dla nich najważniejsze jest dostarczenie użytkownikom końcowym możliwie najbardziej przyjaznego systemu wspierającego „out of the box" szerokiej gamy sprzętu oraz formatów multimedialnych. Osiągnięcie tych założeń wymaga zmniejszenia rygoru zasad stosowanych podczas selekcji

26 Mind Map of Linux - Version 2, http://linuxhelp.blogspot.com/2006/04/mind-map-of-linux-distributions.html

oprogramowania wchodzącego w skład ich dystrybucji. Zasady te muszą zezwalać na włączenie wielu własnościowych sterowników oraz kodeków. W tym celu większość swojego produktu budują na bazie repozytoriów oprogramowania Debiana i rozszerzają je o własne repozytoria z pakietami, które nigdy nie zostałyby zaakceptowane przez właścicieli pierwotnego projektu. Jak widać, obie drużyny nie rozdzielają tutaj swoich wysiłków między dwa projekty, a jedynie koncentrują się na osobnych rejonach zastosowań. W przypadku problemów wpływających na obie dystrybucje, programiści Ubuntu często uczestniczą w procesie wytwórczym Debiana, aby je rozwiązać.

3.7. Projekt parasolowy

Projekt parasolowy jest centralnym punktem pewnej szerszej inicjatywy, która zrzesza pod sobą wiele podprojektów. Często także inkubuje nowe, aby osiągnąć swoje cele statutowe. Praca w tego typu projekcie koncentruje się głównie w następujących rejonach:

- zarządzanie inicjatywą jako całością,
- integrowanie oraz wspieranie komunikacji między projektami,
- podejmowanie globalnych decyzji,
- dostarczanie określonych zasobów oraz infrastruktury potrzebnej w procesie wytwórczym.

Projekty parasolowe są często organizowane jako fundacje, których cele statutowe pokrywają się z celami projektu. Projekty parasolowe powstają często również pod patronatem korporacji oraz różnego typu instytucji.

Bardzo duże projekty parasolowe posiadają podprojekty, które jednocześnie same są projektami parasolowymi. Ich cele statutowe są z reguły pewną specjalizacją celów ich „rodzica” lub pasują do jego ogólnej strategii. Tego typu struktura buduje hierarchię projektów w postaci drzewa.

Przykładowe projekty

Jednymi z bardziej znanych projektów parasolowych są:

- GNU – jeden z pierwszych otwartych projektów parasolowych, którego celem było utworzenie, a obecnie rozwój w pełni wolnego oraz otwartego systemu operacyjnego. Jednym z jego największych podprojektów jest Gnome, który również jest zorganizowany w postaci projektu parasolowego, nakierowanego na bardziej specjalistyczny cel, jakim jest dostarczanie w pełni wolnego środowiska desktopowego.
- NUI Group – jest raptownie rozwijającą się społecznością nakierowaną na promowanie oraz inkubowanie systemów używających tzw. *naturalnej metodologii interakcji z interfejsem użytkownika.*
- Fundacja Apache – mająca na celu zrzeszanie projektów produkujących narzędzia programistyczne oraz *middleware*, które są prowadzone w sposób merytokratyczny (patrz rozdział 3.3, w którym znajduje się również opis funkcjonowania fundacji).

3.8. Dystrybucja

Dystrybucja oprogramowania, często określana w skrócie jako *distro*, stanowi kolekcję produktów uprzednio skompilowanych, skonfigurowanych i przygotowanych do wspólnej instalacji. Pakiety wchodzące w skład dystrybucji mogą pochodzić z projektów zorganizowanych wokół jednego projektu parasolowego albo mogą być rozwijane zupełnie niezależnie. W trakcie, gdy przedsięwzięcia niezależnych społeczności zajęte są rozwojem swoich produktów, zespół odpowiedzialny za dystrybucję koncentruje się na integracji oraz stabilizacji, aby wydać na ich bazie jedno spójne wydanie. Aby to osiągnąć, społeczność dystrybucji zajmuje się głównie:

- testowaniem pakietów oprogramowania jako elementów większej całości,
- dostarczaniem poprawek do pierwotnych projektów,
- rozwojem własnych mniejszych pakietów oprogramowania, które wypełniają pewne luki w dystrybucji.

Sam termin „dystrybucja” jest najczęściej utożsamiany z rodzinami unixopodobnych systemów operacyjnych, które korzystają z pewnej wspólnej bazy. Na rynku jest obecnie rozwijanych około 300 większych tzw. distr[27], z czego większość z nich oparta jest na oprogramowaniu systemowym pochodzącym od społeczności GNU i BSD. Jednymi z największych projektów tego typu są:

- z rodziny GNU/Linux: Debian, Ubuntu, Red Hat, SUSE;
- z rodziny *BSD: NetBSD, FreeBSD, OpenBSD;
- Solaris oparty na Unix System V.

Projekty dystrybucji oprogramowania są często inicjowane po to, aby tworzyć kompletne stosy narzędzi programistycznych oraz *middleware*. Jest to powszechna praktyka w świecie Javowych serwerów aplikacji. Przykładami są:

- Apache Geronimo – certyfikowany serwer J2EE 5, który łączy w sobie oprogramowanie udostępniane na licencji Apache[28] (ale nie tylko pochodzące z projektów fundacji Apache),
- Glassfish – implementacja referencyjna serwera J2EE łącząca w sobie głównie komponenty pochodzące z projektów open source organizowanych początkowo przez Sun Microsystems, a obecnie rozwijanych przez Oracle. Sam serwer jest także bazą dla up-sellowanego produktu zwanego Oracle GlassFish Server.

Innym przykładem dystrybucji stosu technologicznego jest projekt WAMP, którego celem jest dostarczanie łatwego w instalacji środowiska dla aplikacji webowych, dla rodziny systemów MS Windows. Stos ten składa się z serwera HTTP Apache, bazy danych MySQL oraz interpretera PHP. Imituje to ogólnie znany stos LAMP (Linux, Apache, MySQL, PHP...), który sam w sobie nie potrzebuje oddzielnego projektu dystrybucji, jako że jego komponenty są dostępne w postaci łatwo instalowanych pakietów w repozytoriach większości dystrybucji linuxowych.

27 DistrWatch Statistics - http://distrowatch.com/stats.php?section=popularity
28 Open Source Community Overview - Apache Wiki, http://cwiki.apache.org/GMOxPMGT/open-source-community-overview.html

Spotykane są też projekty dystrybucyjne składające się wyłącznie z aplikacji dla użytkownika końcowego. Portable Apps jest jednym z nich, a jego przeznaczeniem jest dostarczanie swoim użytkownikom kompletnego zestawu aplikacji codziennego użytku w postaci tzw. przenośnej paczki (ang. *portable pack*). Umożliwia to uruchamianie ich bez jakiejkolwiek instalacji oraz ingerencji w pliki systemowe.

Przykładowy projekt Debian

Debian jest projektem tworzącym kompletny system operacyjny składający się wyłącznie z pakietów wolnego oprogramowania. Bazą systemu jest jądro Linux oraz oprogramowanie systemowe GNU, dlatego też często nazywany jest dystrybucją GNU/Linux, mimo że dostępne są także warianty z jądrem Hurd oraz FreeBSD[29]. Projekt założył w 1993 roku Ian Mardock na bazie Softlanding Linux System[30], nazywanego w skrócie SLS. Debian rozwijał się powoli w pierwszych latach swojego istnienia. Zwiększenie aktywności projektu można zauważyć na przełomie roku 1996, wraz z wydaniem 1.1 buzz dla architektury x86 zawierającego 474 pakiety oprogramowania. Od tego czasu, Debian nabrał rozpędu i co 2 lata niemal podwaja swoje repozytoria oprogramowania. Wraz z wydaniem wersji 5.0 lenny projekt może się pochwalić około 23 tysiącami pakietów dostępnych dla 12 architektur, takich jak x86 i x86-64, SPARC, PowerPC lub AVR. Z powodu tak wielkiego zakresu Debian jest chętnie wybierany jako baza dla innych bardziej specjalizowanych „forków” dystrybucji, takich jak np. Ubuntu, MEPIS, Damn Small Linux (patrz roz. 3.6).

Projekt Debian jest organizacją składającą się jedynie z wolontariuszy nie opłacanych bezpośrednio przez jego władze. Fundamentami tej struktury organizacyjnej są trzy dokumenty:

- Debian Social Contract, który definiuje charakter oraz podstawowe zasady działania projektu[31],

29 Combining Debian and FreeBSD; Pushing the Envelope of FOSS – Linux Magazine, 09.03.2009, http://www.linux-mag.com/id/7295
30 Debian and the grass roots of Linux – 31.10.2008, http://www.itpro.co.uk/applications/features/135084/debian-and-the-grass-roots-of-linux.html
31 Debian Social Contract – Debian, 21.11.2008, http://www.debian.org/social_contract

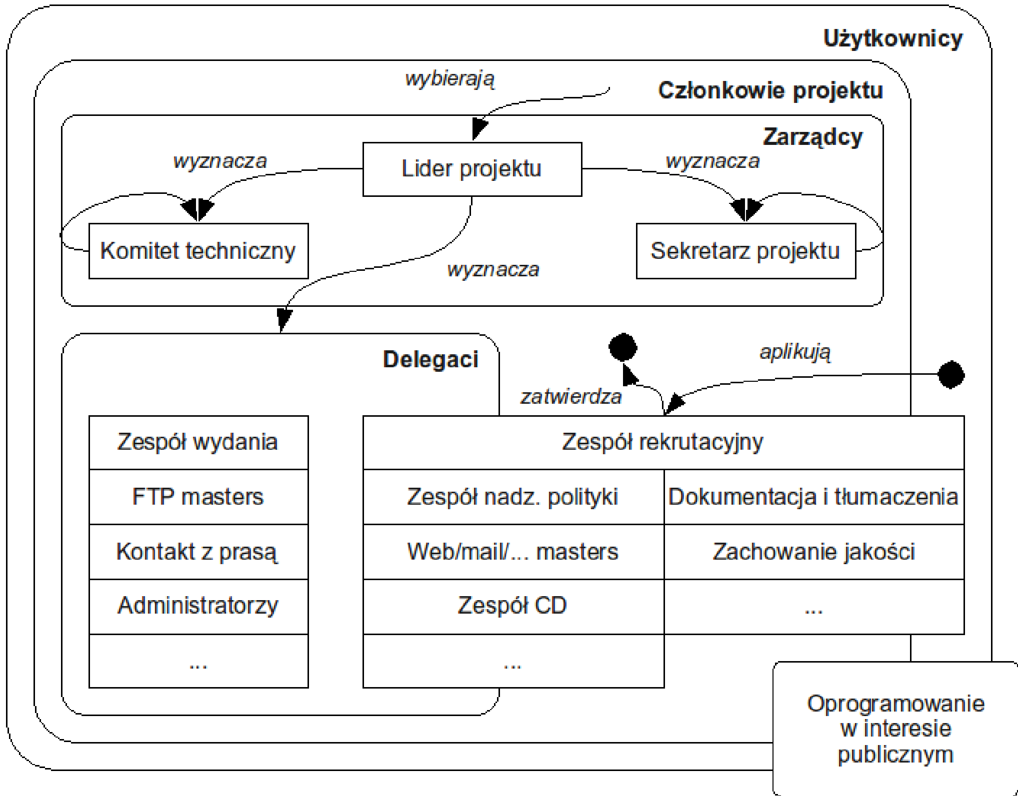
- Debian Free Software Guidelines zawierający definicję wolnego oprogramowania, które może trafić do oficjalnych repozytoriów (patrz roz. 2.4),
- The Debian Constitution, który określa przebieg procesu podejmowania decyzji w projekcie oraz wyróżnia rolę lidera projektu, sekretarza oraz programistów.

Obecnie w projekcie bierze udział około kilku tysięcy programistów, których praca organizowana jest wokół konkretnych pakietów oprogramowania. Istnieje również podział na zadania związane z dystrybucją jako całością, takie jak chociażby: tworzenie dokumentacji, utrzymywanie infrastruktury projektu, zachowywanie jakości oraz koordynacja oficjalnych wydań.

Proces podejmowania decyzji w projekcie Debian jest rozproszony i zazwyczaj odbywa się wewnątrz grup odpowiedzialnych za utrzymywanie konkretnych pakietów. Często w tym procesie biorą udział lub są proszeni o opinię oryginalni autorzy owych pakietów, jako że większość oprogramowania wspieranego przez Debian nie jest wytwarzane w jego ramach. W kwestiach spornych lub wymagających szerszych konsultacji, społeczność może podjąć ogólną rezolucję lub zarządzić głosowanie. System głosowania oparty jest na metodzie Schulzego, która stanowi tzw. ordynację preferencyjną. Głosowanie odbywa się przez wpisanie liczby przy każdej opcji. Wyborca oddaje głos, oznaczając wybrane opcje numerami: zaznaczając 1 obok najbardziej preferowanej opcji, 2 obok następnej w kolejności preferencji, itd. Jeżeli w zestawieniach kandydatów parami w wyniku tych zestawień jeden z nich jest preferowany, metoda Schulzego gwarantuje, że ten kandydat wygra wybory. Według tych samych zasad wybierany jest corocznie Lider Projektu. Ten ostatni ma kilka dodatkowych sposobów na egzekwowanie swojej władzy, jednak nie są one, ani niezaprzeczalne, ani też często stosowane. Przy pomocy generalnej rezolucji programiści mogą między innymi: odwołać lidera, jego pojedynczą decyzję, wyznaczonych przez niego delegatów lub nawet zmienić fundamentalne dokumenty opisujące relacje w projekcie. Z tych cech jasno wynika, że podejmowanie decyzji w projekcie Debian ma charakter merytokratyczny (patrz roz. 3.3).

Lider projektu często wykorzystuje swoją władzę, aby delegować programistów do wyspecjalizowanych zadań. Delegowani programiści zazwyczaj

zawiązują grupę roboczą wokół konkretnego zadania, a także określają jej rozmiar, zgodnie z własnym uznaniem. Rolą o podobnym znaczeniu, jak lider projektu, jest menedżer wydania. Jest on odpowiedzialny za nadzorowanie całego procesu wytworzenia kolejnego wydania w projekcie oraz podejmuje decyzję, kiedy to wydanie ma nastąpić[32]. Podsumowanie ogólnego podziału ról w projekcie przedstawia poniższy rysunek - 3.8.A.



Rys.3.8.A. Struktura społeczności Debian

Jak już wcześniej wspomnieliśmy, życie w projekcie toczy się głównie wokół poszczególnych paczek oprogramowania. Nowe wersje pakietów trafiają z reguły najpierw do repozytoriów oznaczonych jako niestabilne lub eksperymentalne. Nazwa może być nieco myląca: w repozytorium niestabilnym znajdują się głównie pakiety uznane za wydania stabilne przez swój projekt macierzysty. Różnią się one jednak

32 Debian organization web page – Debian, 01.11.2008, http://www.debian.org/intro/organization

drobnymi modyfikacjami wykonanymi przez zarządców pakietu oraz otrzymanym specjalnym paczkowaniem, zgodnym ze standardami Debiana. Jeżeli oprogramowanie będzie leżało odpowiednio długi okres czasu w repozytorium niestabilnym i nie zostaną w nim wykryte żadne poważne luki, wtedy jest ono przenoszone do repozytoriów testowych. Tam po raz kolejny oprogramowanie ma czas na odpowiednie testy oraz na zaaplikowanie łat na wykryte błędy. Kiedy oprogramowanie z repozytoriów testowych osiągnie odpowiedni poziom dojrzałości, staje się ono *de facto* nowym wydaniem systemu Debian. Użytkownik systemu Debian może sam decydować, czy jego system ma korzystać tylko z repozytorium stabilnego, czy także z pozostałych, aby otrzymywać najświeższe oprogramowanie. Sam projekt natomiast dostarcza zawsze aktualizację bezpieczeństwa dla pakietów poprzedniego wydania systemu z repozytorium stabilnego, a od niedawna również dla testowego.

Każdy pakiet oprogramowania posiada co najmniej jednego opiekuna w ramach projektu Debian. Osoba ta odpowiedzialna jest za:

- śledzenie zmian w wydaniach z macierzystego projektu pakietu,
- synchronizację informacji w bug trackerze Debiana z bug trackerem projektu macierzystego,
- zapewnianie, że paczka jest zgodna z polityką projektu Debian, jest koherentna z resztą dystrybucji, a także czy spełnia określone wymogi jakościowe.

Co jakiś czas opiekun pakietu wprowadza jego nowe wydanie do repozytoriów, a dzieje się to za pośrednictwem zautomatyzowanej kolejki, która sprawdza czy paczka została zbudowana zgodnie z wymaganiami technicznymi projektu. Z owej kolejki pakiet jest rozsyłany na wiele serwerów lustrzanych z repozytoriami niestabilnymi i przygotowywany do dystrybucji. Aby pakiet mógł trafić do repozytoriów testowych poza warunkami, które zostały wspomniane wcześniej, musi także spełniać inne wymogi:

- nie może mieć więcej znanych niekrytycznych bugów niż obecna wersja pakietu z repozytoriów testowych,

- być skompilowany i przetestowany na wszystkich wspieranych przez pakiet architekturach,
- wszystkie inne pakiety wymagane do działania danego pakietu muszą mieć także odpowiednią wersję w repozytoriach testowych,
- operacja instalacji pakietu do testowych repozytoriów nie może uszkadzać żadnego innego pakietu, który już się tam znajduje.

Co pewien czas menedżer wydania publikuje informacje na jakim etapie znajduje się projekt i kiedy opiekunowie mogą spodziewać się przygotowań do kolejnego wydania. Przygotowania te rozpoczynają się od momentu kiedy ów menedżer uzna, że wszystkie najważniejsze pakiety zostały odpowiednio zaktualizowane dla wszystkich wspieranych architektur oraz że zostały osiągnięte cele następnego wydania. Cykl życia pojedynczego pakietu w projekcie Debian przedstawia poniższy diagram 3.8.B.

Rys.3.8.B. Przepływ pracy w projekcie Debian[33]

33 The Debian System - Concepts and Techniques - Prace Martin F. Krafft i Kevin Mark, http://debiansystem.info/ lub http://arch.madduck.net/cgi-bin/archzoom.cgi/madduck@debian.org--2005-debian-misc/package-cycle--main--LATEST--LATEST

Modele biznesowe

Rozdział 4

4.1. Ogólne praktyki

Czerpanie zysków z oprogramowania jest zazwyczaj utożsamiane ze sprzedażą licencjonowanych pakietów instalacyjnych. Dlatego też trudno jest przekonać ludzi do możliwości czerpania zysków z produkcji oprogramowania *open source*, w stosunku do którego nie można zastosować tej formy sprzedaży. Patrząc jednak na dane statystyczne z USA można zauważyć, że nie więcej niż 25% kopii oprogramowania jest sprzedawanych w formie licencjonowanych pakietów instalacyjnych[TEO]. Również faktyczna efektywność tego modelu często okazuje się być o wiele niższa, niż mogłaby sugerować intuicja. Można oszacować, że wymagana będzie stopa zwrotu rzędu dziesięcio- do dwudziestokrotnej wielkości kosztu prac rozwojowych, aby opłacało się wytwarzać oprogramowanie, które ma być później sprzedawane w tej formie[TEO].

Open source można postrzegać jako znacznie lżejszą metodykę wprowadzania produktów na rynek w stosunku do licencjonowanych pakietów instalacyjnych. Dzięki otwartemu procesowi wytwórczemu można pozwolić sobie na:

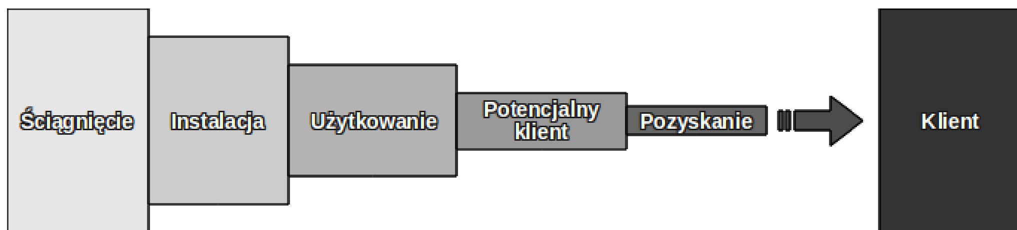
- wcześniejsze rozpoczęcie dystrybucji oprogramowania, jako że wymagania klientów tego typu projektów są z reguły mniej wygórowane,
- dystrybucję kosztów oraz ryzyka niepowodzenia projektu na wczesnym etapie,
- ograniczenie kosztów marketingu.

Zasadniczym problemem jest także konieczność odejścia od tradycyjnych modeli biznesowych, opartych na pełnej wyłączności do praw intelektualnych oraz tajemnic handlowych. Dlatego też praca włożona w projekty *open source* musi zostać spieniężona często w innych rejonach aniżeli sprzedaż licencji. Na takim właśnie założeniu opiera się większość modeli biznesowych otwartych projektów, takich jak *up-selling*, *cross-selling*, czy sprzedaż powiązanych usług (patrz roz. 4.3 i 4.4). Są one zgodne z ogólnymi trendami w branży IT, jak na przykład:

- zorientowanie na usługi, wraz z modelami typu SaaS i IaaS (ang. Software/Infrastructure as a Service),

- praktyka kuszenia darmowymi produktami, które klient będzie skłonny później wymienić na tzw. wersje *enterprise*.

Modele biznesowe oparte na oprogramowaniu *open source* są atrakcyjne na rynku ponieważ z perspektywy klienta zmniejszają koszty związane z ryzykiem wyboru niewłaściwego produktu. Otwarte oprogramowanie minimalizuje również możliwość stosowania tzw. *vendor lock-in*, czyli ograniczania klienta do usług i produktów jednego dostawcy. Jednak z drugiej strony, w modelach biznesowych *open source* należy się pogodzić w faktem, że większość użytkowników otwartego oprogramowania nie zapłaci jego twórcom w żadnej formie. Przyczyną tego faktu jest dłuższy proces, jaki z reguły przechodzi użytkownik zanim stanie się klientem firmy powiązanej z projektem. Drogę ku temu ilustruje poniższy rysunek (rys. 4.1.A).



Rys.4.1.A. Proces od użytkownika do klienta[34]

Negatywnym skutkiem powyższego procesu jest ograniczenie potencjalnych zysków. Jednak ma on także pewien pozytywny skutek. Otóż proces ten potrafi zmniejszyć koszty związane z pozyskaniem klienta, ponieważ wszyscy zainteresowani są w stanie we własnym zakresie zbadać możliwości produktu i nie muszą być „przekonywani" przez różne działania marketingowe, które de facto są bardzo kosztowne[COS]. Wymusza to przyjęcie strategii, w której 5% klientów generuje 90% zysku. Według niektórych oszacowań przeciętny otwarty projekt musi się liczyć nawet z procentem użytkowników generujących zysk dla przedsięwzięcia na poziomie zaledwie 0,5-2%[35].

34 Smoothing the On-ramp to Commercial – Larry Augustin, prezentacja z 2008 Open Source Business Conference, http://www.infoworld.com/event/osbc/08/
35 User to Customer Conversion Rates - Jacob Taylor, 03.2009.

Niektórym projektom *open source* udaje się pozyskać finanse przy pomocy dotacji oraz różnego typu form uznaniowych (patrz roz. 4.5). Otwarte metodyki wytwarzania oprogramowania są również często wybierane, z tego powodu iż pozwalają one na realizację wielu innych celów biznesowych, które nie przynoszą bezpośrednich zysków (patrz roz. 4.6). Te cele są też częstą przyczyną, dla której firmy i instytucje dotują niezarządzane przez siebie otwarte projekty.

Mimo wymienionych w tym rozdziale słabych stron oraz często mało transparentnym sposobom osiągania zysków, projekty *open source* już teraz odgrywają istotną rolę w światowej gospodarce. Zgodnie z obecnymi badaniami statystycznymi oraz prognozami na przyszłość można oszacować, że[GSH2]:

- ogół usług i produktów związanych z otwartym oprogramowaniem szacunkowo wyniósł 32% rynku usług IT w roku 2010, co stanowiłoby do 4% ogólnej gospodarki europejskiej,
- otwarte oprogramowanie wspiera w dużej skali bezpośrednio 29% wewnętrznych projektów Unii Europejskiej oraz 49% w USA,
- *open source* pozwala oszczędzić światowej gospodarce szacunkowo 36% kosztów związanych z badaniem i rozwojem oprogramowania,
- wartość środków, zainwestowanych w rozwój *open source* w Europie szacuje się na 22 miliardy euro, co reprezentuje 20.5% inwestycji ulokowanych w rozwój oprogramowania na kontynencie; w USA jest to 36 miliardów euro i 20% wartości inwestycji w rozwój oprogramowania.

4.2. Podwójne licencjonowanie

Podwójne licencjonowanie jest praktyką polegającą na rozprowadzaniu oprogramowania na dwóch różnych zestawach zasad i warunków. Może to oznaczać dwie różne licencje lub dwa różne zestawy licencji. Oprogramowanie jest czasem rozprowadzane również w postaci więcej niż dwóch różnych licencji, dlatego też określenia typu *potrójne licencjonowanie* lub *wielolicencjonowanie* mogą być w tych przypadkach bardziej precyzyjne. Takie podejście daje odbiorcom możliwość wyboru zasad pozyskania prawa do użytkowania i dalszego rozpowszechniania danych produktów. W zależności od ustaleń twórców oprogramowania, do każdej opcji

licencyjnej z osobna może zostać dołączony obowiązek uiszczania opłat. Dwa czynniki, które zazwyczaj motywują do zastosowania modeli opartych na wielolicencjonowaniu to:

- kompatybilność z innymi licencjami – pomaga to rozwiązać pewne aspekty prawne związane z dystrybucją oraz integracją z innymi systemami informatycznymi (patrz rozdział 2.4),
- zastosowanie modeli biznesowych opartych na segmentacji rynku, które są tematem tego rozdziału.

Podwójne licencjonowanie jest częstą praktyką stosowaną w modelach biznesowych opartych na produktach *open source*. Bazują one głównie na segmentacji rynku na klientów chcących integrować otwarty produkt w ich zamkniętych projektach oraz na środowisko *open source*. Ci pierwsi, zwani często klientami komercyjnymi, muszą zaopatrzyć się we własnościową licencję oraz uiścić opłaty z nią związane. Przy tym zapewniane są często różnego rodzaju usługi powiązane. Grupa komercyjnych klientów służy w tym modelu do spieniężania pracy, którą wykonali autorzy oprogramowania. Natomiast grupy programistów rozwijających otwarte oprogramowanie mogą skorzystać z otwartej licencji typu *copyleft* (patrz rozdział 2.4). Zgodnie z ogólną kulturą środowiska *open source* (patrz roz. 3.1), pomagają oni w rozwoju projektu. Według wyników ankiet zaprezentowanych przez The FLOSS Metrics Consorcium[SME], podwójne licencjonowanie jest stosowane przez około 5% firm czerpiących zyski z *open source*.

Z racji tego, że w tym modelu autorzy oprogramowania posiadają z założenia prawa autorskie do wszystkich komponentów systemu, mogą oni budować na jego bazie równoległy produkt rozszerzany o komercyjne dodatki. Jest on wtedy sprzedawany na zasadach licencji własnościowych i często sugerowany klientom, dla których bazowy produkt przestał być wystarczający. Technika taka nosi nazwę up-sellingu (patrz rozdział 4.3). Sam model biznesowy jest często wybierany tylko ze względu na swoje walory marketingowe. Dzieje się to zazwyczaj pośród firm otwierających swoje produkty, jednakże przy założeniu, że większość programistów będzie nadal działać w tradycyjny, zamknięty sposób. Liczą oni głównie na zbudowanie w ten sposób licznej i oddanej społeczności, która będzie im zapewniać sprzężenie zwrotne dotyczące tematu ich pracy oraz promować ich produkt za

pośrednictwem tak zwanego marketingu szeptanego. Jednak poprzez wymuszanie wymogów licencji restrykcyjnych oraz reklamowanie swoich zamkniętych produktów, które się uchylają od tych wymogów, można spodziewać się pewnego poziomu utraty zaufania oraz wsparcia ze strony społeczności projektu[36] [37].

Wpływ na strukturę organizacyjną

Model biznesowy, opisany w niniejszym rozdziale, ma ogromny wpływ na strukturę organizacyjną, w związku z tym, iż wymusza on od autorów projektu zachowanie całkowitych praw autorskich w trakcie rozwoju oprogramowania. Powoduje to często dezorientację programistów, którzy po utworzeniu jakiegoś dodatku i udostępnieniu go na równie restrykcyjnej otwartej licencji nie mogą wymusić włączenia go do źródeł projektu. Dzieje się tak, ponieważ firma fundująca projekt nie może zawierać tego dodatku w ich produkcie sprzedawanym na zasadach licencji własnościowej. Autorzy projektu nie mogą zatem pozwolić sobie na rozpływanie się praw autorskich od źródeł projektu (patrz rozdział 2.4). Dlatego też często wymagane jest od programistów wspierających projekt, aby przekazali oni swoje prawa autorskie do udostępnionych prac, zanim trafią one do oficjalnych repozytoriów[38]. Konieczność ta często zmniejsza motywację do współpracy nad projektem, czyniąc go bardziej zależnym od pracowników zatrudnionych przez fundatora projektu. Jest to często świadomy wybór, ponieważ firmy mogą w ten sposób zachować większą kontrolę nad rozwojem oprogramowania i dlatego wybierają strukturę organizacyjną opartą na procesie wytwórczym wewnętrznym oraz sprzężeniu zwrotnym w społeczności (patrz rozdział 3.4).

Przykłady w biznesie

Za kombinacją podwójnego licencjonowania oraz wspomnianej wcześniej formą organizacji przemawia kilka spektakularnych historii biznesowych. Najbardziej znanymi z nich są przypadki firmy Trolltech z ich toolkitem (ang. zestawem narzędzi) Qt oraz MySQL AB z bazą danych MySQL[OLP]. Firmy te powstały na końcu lat

36 On the Netscape Public License - Richard Stallman, http://www.gnu.org/philosophy/netscape-npl.html
37 FSF's Opinion of the Apple Public Source License (APSL) 2.0 - http://www.gnu.org/philosophy/apsl.html
38 Asterisk Guidelines, The contributor license agreement - Digium Incorporated, http://www.asterisk.org/developers/bug-guidelines

dziewięćdziesiątych i po kilkukrotnym zwiększeniu swojej wartości zostały zakupione przez większych graczy za odpowiednio 100 milionów dolarów przez Nokię[39] i 1 miliard dolarów przez Sun Microsystems[40]. Z drugiej strony, wiele osób postrzega model oparty na podwójnym licencjonowaniu jako element, który powstrzymuje projekty od osiągnięcia wszystkich korzyści płynących z metodologii *open source*. Możliwe, że właśnie w związku z tym Nokia zaraz po przejęciu Trolltech rozluźniła restrykcje licencyjne na Qt dodając trzecią opcję w postaci licencji LGPL. Otworzyło to furtkę dla nieodpłatnego wykorzystywania tych narzędzi w zamkniętych projektach. *De facto* taka polityka zapewnia opłaty licencyjne tylko od firm, które chcą sprzedawać produkty wykorzystujące zmodyfikowaną wersję Qt bez udostępniania źródeł wprowadzonych modyfikacji. Zawęża to grono klientów, którzy będą skłonni zapłacić za wariant płatny, jednak spodziewane jest zwiększenie zaangażowania stron trzecich. Ponieważ Nokia czerpie swoje główne zyski z produkcji telefonów komórkowych, a Qt staje się jej nową mobilną platformą[41], jest prawdopodobne, że te zabiegi mają na celu poświęcić zyski z opłat licencyjnych na rzecz niższych kosztów rozwoju oprogramowania (patrz. roz. 4.6).

4.3. Up-selling i Cross-selling

Up-selling i cross-selling są technikami sprzedażowym polegającymi na oferowaniu klientowi, który już zamówił u nas jakiś produkt, innego produktu nawiązującego lub ulepszającego pierwotny zakup. Pierwsza technika wiąże się z promowaniem wersji rozszerzonych lub samych rozszerzeń. Druga natomiast koncentruje się bardziej na oferowaniu produktów luźno powiązanych, z tymi które już trafiły do klienta. Obydwie techniki są domeną firm, które posiadają bogate portfolio zarówno otwartych, jak i zamkniętych projektów. Oprogramowanie *open source* jest wtedy wykorzystywane do argumentowania sprzedaży produktów zamkniętych.

39 Nokia ostaa norjalaisen ohjelmistoyrityksen – 28.01.2008, http://www.hs.fi/talous/artikkeli/Nokia+ostaa+norjalaisen+ohjelmistoyrityksen/1135233615595

40 Sun acquires MySQL – artykuł na blogu MySQL AB, 16.01.2008, http://blogs.mysql.com/kaj/2008/01/16/sun-acquires-mysql/

41 Nokia Ports Qt platform to S60 platform - informacja prasowa HANDCELL PHONE, http://www.handcellphone.com/archives/nokia-ports-qt-platform-to-s60-platform

Technika up-sellingu z reguły wymusza pewną formę organizacji wzajemnie powiązanych projektów. Podstawą jest otwarty projekt wytwarzający bazowy produkt, z którym na początku zapoznają się potencjalni klienci. Mimo że zazwyczaj większość zasobów firmy pochłania wytwarzanie tego oprogramowania, bezpośrednie zyski są generowane przez powiązane zamknięte projekty. Te natomiast wytwarzają rozszerzoną wersję produktu bazowego albo dedykowane dla niego wtyczki (ang. *plugins*). Według wyników ankiet zaprezentowanych przez The FLOSS Metrics Consorcium[SME] takie podejście stanowi element modelu biznesowego dla około 12% firm czerpiących zyski z *open source*. Pierwsza metoda polegająca na sprzedaży rozszerzonej wersji produktu bazowego jest również często nazywana modelem *freemium*[42]. Wymagane jest w nim, aby firma posiadała prawa autorskie do całości bazowego systemu, jeżeli ten udostępniany jest na zasadach licencji restrykcyjnej. Dlatego też ta forma up-selling`u jest często wykorzystywana wraz z podwójnym licencjonowaniem, które też tego wymaga. Brak restrykcyjnej licencji i pełni praw autorskich zwiększa ryzyko powstania konkurencyjnego zamkniętego „forka” projektu. Natomiast zachowanie tych dwóch zasad odbija się tak samo na strukturze organizacyjnej, jak w przypadku podwójnego licencjonowania, omówionego w poprzednim rozdziale. Poza podobnymi wymaganiami można też zauważyć pewne podobieństwa w technikach sprzedaży stosowanych w omawianych tu modelach. Z tych względów niektóre firmy stosują połączenie up-selling'u oraz podwójnego licencjonowania, co nazywa się wówczas *podejściem fazowym*[43].

W przypadku powiązanych projektów, wytwarzających wtyczki, licencja produktu bazowego nie ma aż tak istotnego znaczenia pod względem biznesowym, o ile umożliwia powstanie zamkniętych dodatków. Często w takim przypadku wykorzystywana jest nierestrykcyjna licencja BSD, aby maksymalnie uprościć współpracę ze społecznością. Praca opłacanych członków projektu jest wówczas zwykle finansowana przez firmy zarabiające na zamkniętych komercyjnych wtyczkach.

42 Freemium–Free to Paid Conversion Rates - Don Dodge, http://dondodge.typepad.com/the_next_big_thing/2007/05/freemium_free_t.html

43 A Time to Reap, a Time to Sow: A Phased Approach for Open-Source Businesses - Matt Asay, http://news.cnet.com/8301-13505_3-9945870-16.html

Przykłady up-sellingu

Jednym z głośnych w ostatnich latach przykładów firmy, opierającej swoje zyski na up-sellingu produktu *open source*, jest SugarCRM. Jest to obecnie szeroko rozpoznawalny system CRM bazujący na stosie technologicznym LAMP, który narodził się w roku 2004 na SourceForge.net. SugarCRM błyskawicznie zyskiwał na popularności zarówno pośród użytkowników, jak i społeczności programistów. Pozwoliło to na pozyskanie firmie 46 milionów dolarów od trzech funduszy *venture capital*: Draper Fisher Jurvetson, Walden International oraz New Enterprise Associates[44]. Obecnie SugarCRM zatrudnia 150 ludzi i jest uznawany za jeden z najbardziej aktywnych projektów na SourceForge.

Kiedy firma startowała na początku roku 2004, zatrudniała około 10 pracowników, a jej zyski pochodziły głównie z zapewniania powiązanych usług (patrz roz. 4.4) w cenie 149 dolarów od użytkownika. Usługi te nadal stanowią ważne źródło dochodów, jednak firma przesunęła swój model nieco bardziej w stronę up-sellingu swoich komercyjnych produktów. Na otwartej wersji SugarCRM bazują dwa zamknięte produkty nazywane Professional i Enterprise. Szacuje się, że otwarty kod zajmuje 85% ich źródeł. Zgodnie z oświadczeniami autorów projektu, 50% czasu pracy ich programistów jest poświęconych na rozwój otwartego oprogramowania, a kod ofiarowany przez osoby trzecie nie jest nigdy zamykany w komercyjnych wariantach[45]. Obie płatne edycje są nastawione na odpowiednio większych odbiorców potrzebujących takich funkcjonalności, jak raportowanie, prognozowanie sprzedaży lub modyfikowalny *workflow* (ang. przepływ pracy), a zarazem na klientów nieobawiających się stosunkowo wysokich opłat licencyjnych. Wszystkie te dodatkowe możliwości zostały zaimplementowane przez pracowników firmy. Licencje na SugarCRM Professional i Enterprise kosztują odpowiednio 360 i 600 dolarów rocznie na jednego użytkownika. W roku 2006 zarząd firmy ogłosił, że płatne edycje zostały sprzedane już około 350 klientom, wśród których największy wykupił licencję na 400 użytkowników[46]. Poza sprzedażą licencji na wersje instalowane we własnym zakresie, oprogramowanie SugarCRM jest także dostępne na zasadzie Software as a Service, z czego korzysta około 35% klientów.

44 SugarCRM raises $20M more for open source CRM - VentureBeat, http://venturebeat.com/2008/02/07/sugarcrm-raises-20m-more-for-open-source-crm/
45 Commercial open source, a misnomer? - artykuł ZDNet, Dan Farber, 2005, http://blogs.zdnet.com/BTL/?p=1787
46 SugarCRM Software Review - recenzja ERP.asia, http://www.erp.asia/sugarcrm.asp

Twórcy CRM zapewniają także program partnerski dla innych firm, w zakresie *resellingu* (ponownej sprzedaży) oraz certyfikowania programów szkoleniowych i partnerskich rozwiązań software'owych. Udział w programie jest płatny, a wysokość wymaganych opłat jest zależna od zakresu współpracy. SugarCRM koordynuje oraz hostuje powstawanie wtyczek dla swojego produktu w ramach SugarForge. W ten sposób firma wprowadza także taktykę up-sellingu opartą o komercyjne wtyczki.

Mimo iż SugarCRM towarzyszy duży rozgłos, a samo oprogramowanie przeżywa spontaniczny rozwój, sukces tego przedsięwzięcia jest dyskusyjny. Ponieważ podstawą wszystkich generujących zyski produktów i usług jest produkt darmowy, firma musiała przyjąć zasadę „5% klientów generuje 90% zysków”. Przymus stosowania takiej strategii potwierdzają oświadczenia z roku 2006, zgodnie z którymi archiwum z otwartą wersją SugarCRM zostało ściągnięte przez około milion użytkowników, natomiast sama firma posiadała około 1200 płacących klientów[47]. Projekt również nie ustrzegł się zagrożenia, jakie niesie ze sobą *open source*. Grupa programistów, która była niezadowolona z faktu, że termin *open source* jest bardziej sloganem marketingowym SugarCRM niż metodyką pracy, postanowiła stworzyć „forka” projektu o nazwie vtiger[48] (patrz rozdział 3.6). Mimo tego, według oficjalnych ogłoszeń, firma osiągnęła zyskowność w roku 2007[49]. Jednak w roku 2009 po nagłym odejściu założyciela i CEO SugarCRM, pojawiły się pogłoski, że firma wydała już wszystkie środki pozyskane od podmiotów venture capital, a doniesienia o jej zyskowności były przedwczesne.

Innym przykładem zastosowania up-sellingu względem produktu *open source* jest ekosystem zawiązany wokół środowiska programistycznego Eclipse utworzonego przez IBM. W trakcie, gdy całe środowisko i wiele z jego plugin'ów jest dostępnych nieodpłatnie na zasadach otwartej licencji, sprzedawane jest także wiele płatnych wtyczek. Z ponad tysiąca rozszerzeń dostępnych na Eclipse Marketplace 279 z nich jest udostępnianych odpłatnie[50]. Kilka z tych produktów znajduje się na liście 10 pluginów najczęściej oznaczanych jako ulubione przez programistów, tak jak

47 Sugar CRM – recenzja, Online-CRM.com, http://www.online-crm.com/sugarcrm.htm
48 Talking about Open Source Vs Walking the Talk - oficjalny blog vtiger, 2007, http://www.vtiger.com/blogs/2007/01/19/talking-about-open-source-vs-walking-the-talk/
49 SugarcCRM revenue doubles - artykuł The VAR Guy, 2007, http://www.thevarguy.com/2007/08/27/sugarcrm-revenue-doubles/
50 Eclipse Marketplace – wyniki wyszukiwania dla płatnych wtyczek z 11.2009.

np. WindowBuilder Pro lub CodePro AnalytiX firmy Instantiations. Ceny wtyczek w portfolio tej firmy wahają się w przedziale od 39 do 2199 dolarów. Przyglądając się głębiej profilowi działalności firmy Instantiations można zauważyć, że wytwarzanie wtyczek również otwiera drogę ku sprzedaży usług powiązanych z oprogramowaniem Eclipse na zasadzie tzw. specjalistów produktu (patrz rozdział 4.4).

Bardziej rozbudowaną formą zarabiania na tym środowisku programistycznym jest tworzenie własnych specjalizowanych lub certyfikowanych środowisk IDE. Wykorzystywany jest do tego opisany wcześniej wariant up-selling'u zwany freemium. W tym kontekście oryginalny Eclipse stanowi nasz produkt bazowy, z którym zapoznają się użytkownicy. Natomiast sprzedajemy licencję na naszą dystrybucję użytkownikom, którzy wymagają szerszej funkcjonalności lub sprawniejszego audytu jakości. Według takiego modelu postępują twórcy:

- JBuilder 2008 – jest to próba uratowania koncepcji wytworzonych podczas rozwoju środowisk Borland. Po rosnących problemach firmy Borland, oddział wytwarzający narzędzia programistyczne został wydzielony w postaci przedsiębiorstwa CodeGear, a idea tzw. *visual developemnt model* została przekierowana z języków Delphi i C++ w stronę platformy Java. Polityka ta wraz z rozwojem JBuildera wydaje się przynosić efekty, a samo CodeGear zostało w roku 2008 kupione przez Embarcadero za 24,5 miliona dolarów[51],
- IBM WebSphere Studio Workbench – jest dystrybucją samych autorów Eclipse. Stanowi podstawowe środowisko programistyczne dla stosu technologicznego WebSphere,
- JBoss Developer Studio – dystrybucja przeznaczona dla stosu JBoss.

Dystrybucje te często są również związane z różnymi płatnymi usługami. Niektóre firmy związane z Eclipse bazują tylko na tym modelu biznesowym, jak na przykład EclipseSource.com specjalizująca się w wytwarzaniu dystrybucji pod potrzeby specyficznego klienta.

51 Embarcadero Completes Acquisition of CodeGear - SQL Server Magazine, 2008, http://www.sqlmag.com/Articles/ArticleID/99811/99811.html?Ad=1

Przykład cross-sellingu

Trudno jest opisać jednoznaczne przykłady cross-selling'u, który jako technika sprzedażowa odznacza się pewną subtelnością działania oraz luźnym powiązaniem ze sobą produktów i usług. Mimo to łatwo można dostrzec w szerokich portfoliach dużych koncernów wiele produktów, które mimo że nie są ze sobą bezpośrednio związane, to często są łączone podczas przygotowywania ofert dla klientów. Taką praktykę można zazwyczaj zaobserwować wśród tzw. *dostawców platform* (patrz roz. 4.4), którzy argumentują zakup produktów zamkniętych oraz płatnych usług produktami projektów *open source*. Jedną z takich firm jest bez wątpienia IBM. Określany także jako Enterprise Systems Provider, zapewnia klientom kompleksowe rozwiązania w postaci gotowych systemów informatycznych, stosów technologicznych, *middleware*, architektur sprzętowych oraz zapewnia usługi wspierające wszystkie te elementy. Technika cross-selling'u w tak szerokim zakresie jest o wiele bardziej skomplikowaną formą sprzedaży niż up-selling. Aby wspomóc w tym zadaniu swoich sprzedawców i partnerów, IBM zapewnia obszerny instruktaż o nazwie IBM Software Cross Sell Reference Guide[52]. W tym dokumencie można znaleźć opis 871 produktów oraz ponad 3400 relacji występujących między nimi. Do przykładów cross-sellingu, w kontekście koncernu, można zaliczyć wcześniej wspomniany Eclipse, w którym łatwo uzyskuje się integrację z wieloma produktami firmy IBM.

4.4. Zapewnianie powiązanych płatnych usług

Zapewnianie powiązanych usług jest jednym z najczęściej spotykanych modeli firm zaangażowanych w oprogramowanie *open source*. Nawet jeżeli nie stanowią one ich głównego źródła dochodu, to są z reguły czynnikiem komplementarnym w stosunku do pozostałej działalności. W świadczeniu tych usług bierze udział także wiele firm nie zaangażowanych bezpośrednio w prace projektowe. Mowa tu zwłaszcza o firmach consultingowych o ogólnym zakresie działania oraz adwokackich zajmujących się sprawami z dziedziny *open source*. Uczestnictwo tego typu podmiotów w projekcie często nie jest wymagane przez ich klientów. Jednak zgodnie z ankietami przeprowadzonymi przez The FLOSS Metrics Consorcium na

52 IBM Software Cross Sell Reference Guide - IBM, 2008, http://www.techdata.com/techsolutions/Softwareconnections/files/july08/IBM-Software%20Cross-sell%20Reference%20Guide_v1.pdf

grupie 451 firm czerpiących zyski z oprogramowania open source[SME], te dwa typy usług stanowią łącznie jedynie 2%. Szkolenia pracowników są świadczone zaledwie przez 1%. Tymczasem najpopularniejszą pozycją jest grupa usług świadczonych przez tzw. specjalistów produktu, która stanowi 29%.

Specjalista produktu jest mianem jakim określamy firmę, która rozwija produkt w całości dostępny na licencji open source oraz czerpie zyski ze świadczeń nakierowanych na niego usług, takich jak:

- subskrypcja certyfikowanej wersji produktu wraz z zapewnianiem łat krytycznych luk w przyszłości,
- konsulting oparty na produkcie,
- wsparcie lub całościowa realizacja wdrożeń,
- integracja z innymi systemami,
- tworzenie nowych elementów produktu na zlecenie,
- szkolenia i wsparcie techniczne.

Firmy będące specjalistami produktu promują fakt, że są autorami większości kodu danego oprogramowania i że stanowi to gwarancję najwyższej jakości wymienionych usług. Osiągnięcie sukcesu biznesowego, w tym modelu, wymaga kombinacji jakości danego produktu oraz wiedzy ekspertów zatrudnianych do jego rozwoju. Jednak należy pamiętać, że oryginalny autor projektu nie ma wyłączności na tego rodzaju działalność i inne podmioty mogą również argumentować swoje usługi jakością produktu. Stąd też jedynym czynnikiem wpływającym na konkurencyjność w danym segmencie jest jakość samych ekspertów. W przypadku agresywnych inwestycji, wykonywanych przez duże koncerny, mogą one odebrać pierwotnym autorom możliwość efektywnego spieniężenia pracy włożonej w rozwinięcie projektu. Z drugiej strony, autorzy ci mogą liczyć na znaczną przewagę marketingową oraz PR'ową w przypadku wspomnianych akcji.

Z powyższych względów firmy zarabiające na usługach tego typu znajdują się w paradoksalnej sytuacji. Z jednej strony chcą, aby wiele podmiotów uczestniczyło w procesie wytwarzania ich flagowego produktu. Z drugiej jednak,

obawiają się, iż ich wiedza stanie się podstawą do konkurowania z nimi na rynku. Sprowadza się to do dylematu na początku organizacji projektu:

- Zdecydować się na strukturę bardziej otwartą, zapewniającą produktowi szybszy potencjalny rozwój? (patrz rozdziały 3.2 i 3.3)
- Skoncentrować się na pracy wewnątrz własnego zespołu, ograniczając ryzyko powstania konkurencji?

Innym sposobem podejścia jest również konstrukcja programu partnerskiego. Firmy często sięgają po to rozwiązanie, z tego względu, iż pozwala im rozszerzyć zakres swojego działania poprzez certyfikację partnerów operujących w innych regionach świata.

Podobną grupą dostawców usług do specjalistów produktu są tak zwani dostawcy platformy. Mimo że według wcześniej wspominanych badań[SME] stanowią zaledwie 1% ankietowanych firm, są oni wyraźnie widoczni na rynku z powodu szerokiego zakresu swojej działalności. Dostawcy platformy świadczą zazwyczaj tę samą gamę usług, co specjaliści produktu, jednak zamiast koncentrować się jedynie na wsparciu dla jednego produktu, obsługują całą platformę programistyczną. W ten sposób mogą wspomagać wszystkie działania klienta w dziedzinie, z którą korespondują. Często jednak nie są oni główną siłą napędową rozwoju oprogramowania wchodzącego w skład danej platformy, a sam projekt jej dostarczania ma często strukturę organizacyjną dystrybucji (patrz roz. 3.8).

W grupie zarówno dostawców platformy, jak i specjalistów produktu, najważniejszą bazową usługę stanowi subskrypcja certyfikowanej wersji produktu lub platformy. Badania pośród dostawców rozwiązań open source z roku 2007[53] wskazują, że stanowi to ich główną siłę napędową generując 59% dochodów. Niemal połowa firm biorąca udział w badaniu oznajmiła, że ich biznes opiera się wyłącznie na tej usłudze.

53 IDC survey proves open-source software as viable business model - cnet news, artykuł, 2008, http://news.cnet.com/8301-13846_3-9995988-62.html

Przykład specjalistów produktu

Jednym z szeroko rozpoznawanych projektów, opierających swój model biznesowy na koncepcji specjalistów produktu, jest firma JBoss (obecnie dywizja Red Hat). Produktem firmy jest JBoss AS, otwarty serwer aplikacji Java EE oraz komponenty z nim powiązane. Początki samego projektu sięgają roku 1999, gdy stanowił on element badań nad możliwościami poszerzenia koncepcji oprogramowania typu middleware. Z biegiem czasu serwer JBoss dojrzewał i zyskiwał na popularności. Wkrótce też zespół z nim związany zaczął otrzymywać pytania o możliwość przeprowadzenia szkoleń oraz wykupienia wsparcia technicznego[54]. Marc Fleury, który był jedną z osób zaangażowanych w projekt, dostrzegł w tym pewną okazję. Postanowił zorganizować pierwszą prowizoryczną sesję treningową w garażu swoich teściów. Pomysł okazał się udany, gdyż pozwolił mu na osiągnięcie aż 60 000 $ czystego zysku. W ten sposób przedstawiały się początki założonej w 2001 roku JBoss Group, która stanowiła przykład organizacji *open source* zyskownej od pierwszego dnia istnienia. Czerpiąc swoje zyski niemal wyłącznie z zapewniania usług jako specjalista produktu, firma przeżywała dynamiczny rozwój, aby w końcu zostać kupioną przez korporację Red Hat za 420 milionów dolarów[55]. W tym czasie także JBoss AS zyskał znaczącą część rynku serwerów Java EE oraz wiele poważnych *use case'ów*, takich jak np. system rezerwacji Travelcity lub sieć monitorująca linie przesyłu energetycznego California ISO[56].

Marc Fleury miał znaczący wpływ na rozwój JBoss'a i mimo tego, że nie jest już prezesem w ramach dywizji Red Hat'a, jego poglądy odbijają się na obecny model biznesowy. Był on szczególnie znany z zaprzeczania jakoby poświęcenie nieopłacanych woluntariuszy było główną siła napędową komercyjnych projektów open source.

54 Redemption for JBoss's Boss, BusinessWeek, 19.10.2004, http://www.businessweek.com/technology/content/oct2004/tc20041019_2492_tc182.htm
55 Red Hat Completes Acquisition of Jboss - Red Hat. 05.06.2006, http://www.redhat.com/about/news/prarchive/2006/jboss
56 The Myth of Open-Source - BusinessWeek, 08.07.2005, http://www.businessweek.com/technology/content/jul2005/tc2005078_5465_tc121.htm

Podkreślał on także, iż firmy zarabiające na usługach w tej branży muszą liczyć się z zasadą „5% klientów generuje 90% zysku”. Cytat[57]:

Nikt nie będzie pracował za darmo. To jest mit open source. [Nowe open source'owe startup'y] próbują kupić sposób na dystrybucję. I możliwe, że odniosą sukces, ponieważ jeżeli oddasz coś za darmo, ludzie przybędą. To jedyna część modelu jaką wymieniamy: Daj za darmo i kasuj za usługi. Zasadniczym pytaniem jest czy masz wystarczająco dużo rozgłosu? Jeżeli wykorzystujesz ten model, będziesz spieniężać od 1 do 2% swojej bazy użytkowników, czyli będziesz potrzebował dosyć dużej bazy, aby zachować stabilność.

Zgodnie z tymi wytycznymi styl pracy w dywizji JBoss można określić jako *wewnętrzny proces wytwórczy, sprzężenie zwrotne w społeczności* (patrz rozdział 3.4). Także świadczone usługi w większości są nakierowane na wsparcie dużych klientów i dzielą się na trzy kategorie[58]:

- JBoss Enterprise Subscriptions and Support – najczęściej wybierana usługa w portfolio. Na czas określony w umowie zapewnia dostęp do certyfikowanych wersji źródeł serwera wraz z subskrypcją aktualizacji, łat bezpieczeństwa oraz backport'ów. Do tego dochodzi określona liczba kontaktów z obsługą techniczną, Ceny tej usługi są uzależnione w głównej mierze od liczby procesorów wykorzystanych we wdrożeniu.
- JBoss Enterprise Consulting Services – zakres usług umożliwiających zapewnienie bezpośredniego wsparcia ze strony programistów JBoss na wszystkich etapach projektu. Patrząc etapami mamy do wyboru: wynajem certyfikowanego architekta, organizację warsztatów dla programistów, zamówienie prac migracyjnych, implementacjach oraz optymalizacyjnych.
- JBoss Enterprise Training Services – możliwość przystąpienia do kursów oraz certyfikacji programistów.

57 Redemption for JBoss's Boss - BusinessWeek, 19.10.2004, http://www.businessweek.com/technology/content/oct2004/tc20041019_2492_tc182.htm
58 JBoss Enterprise Services – strona Jboss, http://www.jboss.com/services/

Jest to dość typowe portfolio firm prowadzących tego typu działalność i możemy zauważyć na rynku wiele podobnych lub uproszczonych wariantów wspomnianego wzorca.

Przykłady dostawców platformy

Firmy działające na zasadzie dostawców platformy są najczęściej utożsamiane z komercyjnymi dystrybucjami GNU/Linux (patrz. roz. 3.8). Dwiema najpopularniejszymi z nich są: korporacja Red Hat oraz działająca obecnie w ramach grupy Novell dystrybucja SUSE. W obu firmach dochody generuje głównie pełen zakres usług dla oprogramowania integrowanego w ramach ich systemów operacyjnych. Zestawienie przedstawia tabela 4.4.A.

Typ usługi	Red Hat	SUSE
Certyfikowane dystrybucje wraz z aktualizacjami oraz wsparciem technicznym	Red Hat Enterprise Linux Server/Desktop	SUSE Linux Enterprise Server/Desktop
Consulting	Assessments, Quickstarts, Implementations, Health Checks, Workshops	Linux Core Build Fast Track, Core Build Design and Certification, UNIX to Linux Migrations, Workload Migrations, SUSE Server Management Design & Implementation, SUSE Desktop Migrations
Certyfikacja	sprzętu, programistów, partnerów	wybranych aplikacji firm trzecich, sprzętu, programistów, partnerów
Szkolenia	*	*
Wdrożenia infrastruktury	*	*

Tab.4.4.A. Porównanie usług Red Hat oraz SUSE

Obie firmy wytwarzają własne otwarte narzędzia programistyczne i *middleware* dla swoich platform. Rozwiązania Red Hat w tym zakresie skupiają się na projekcie JBoss, natomiast Novell promuje w świecie linuxowym własną implementację platformy .NET o nazwie Mono. Wszystkie produkty Red Hat mają oddzielnie zakupywane usługi subskrypcyjne, które są agregowane w postaci pakietów zorientowanych tematycznie, jak np. Cloud Computing, Datacenter lub High Performance Computing and Grid. Jeśli zaś spojrzymy na firmę Novell, to należy zauważyć, że ona również opiera swój model biznesowy na cross-selling'u własnych zamkniętych produktów z otwartym oprogramowaniem wykorzystywanym w SUSE (patrz rozdział 4.3). Większość otwartych projektów utrzymywanych przez obie firmy skierowanych jest do płacących klientów. Struktura organizacyjna owych projektów opiera się na wewnętrznym procesie wytwórczym i sprzężeniu zwrotnym w społeczności. Wyjątkami w Red Hat i Novell są odpowiednio dystrybucje Fedora i openSuse, które są nastawione na użytkowników poszukujących darmowych rozwiązań. Obie dystrybucje „żyją" w symbiozie ze swoimi komercyjnymi odpowiednikami Red Hat Enterprise Linux i SUSE Enterprise Linux. Społecznościowe dystrybucje dostarczają z reguły nowszych wersji oprogramowania, które zostało uznane za nieodpowiednio przetestowane, aby trafić na system serwerowy. W ten sposób wykorzystują wyższy stopień zaangażowania użytkowników oprogramowania open source, aby przenieść na nich część kosztów związanych z zachowaniem jakości (patrz roz. 3.1). Model biznesowy podobny do wykorzystywanego przez obie opisane tutaj firmy, a zwłaszcza Novella, wykorzystywał Sun Microsystems[59]. Wszystkie trzy podmioty są często nazywane Enterprise Systems Providers i opierają swoje działania głównie na otwartych produktach.

4.5. Dotacje i inne formy uznaniowe

Dotacje są czasami ważnymi źródłami dochodów dla projektów *open source* i mogą stanowić ich główny przychód lub wspomagać inne źródła utrzymania.

59 Sun Microsystems and Novel: Step Brothers? - The VAR Guy, 2009, http://www.thevarguy.com/2009/01/30/is-sun-microsystems-the-new-novell/

Ze względu na skalę poszczególnych dotacji rozróżniamy:

- drobne dotacje od indywidualnych użytkowników,
- duże granty od firm oraz instytucji.

Drobne dotacje mogą grać istotną rolę w mniejszych projektach, w których mała grupa lub pojedynczy programista proszą o pewną materialną formę uznania dla ich pracy. Szczególnie tyczy się to projektów wytwarzających produkty dla pracowników IT, spośród których około 54% oznajmia, że przynajmniej raz uiściło dotację na tego typu projekt[60]. W niektórych przypadkach pieniądze pochodzące z tego źródła wystarczają na zorganizowanie corocznej konferencji i zakup sprzętu. Zazwyczaj jednak stanowią one jedynie dodatek do grantów zapewnianych przez większe przedsiębiorstwa.

Jako pewną formę dotacji uznaje się często również kupno różnego typu akcesoriów, jak np. kubki lub T-shirty z logiem projektu. Praktyka ta jest stosowana w niektórych dystrybucjach GNU/Linux, np. Ubuntu lub Mandriva. Definicję tej formy dotacji często poszerza się o praktykę sprzedaży estetycznie opakowanych i obrandowanych płyt CD, instrukcji oraz podręczników, które są równocześnie dostępne za darmo i legalnie w sieci. Ta forma jest utożsamiana głównie z dystrybucjami GNU/Linux i jest ona nawet przypisywana jako początkowe źródło dochodów firm, takich jak: Red Hat i SUSE[OLP].

Za największą siłą napędową projektów *open source*, które nie są finansowane przez firmy z modelami biznesowymi opartymi na ich produkcie, uznaje się granty instytucji oraz dużych graczy branży IT. Należy przy tym pamiętać, że tego typu datki ze strony korporacji często nie wynikają z altruistycznych potrzeb ich szefów, lecz są wynikiem realizacji innych celów biznesowych, opisywanych w rozdziale 4.6.

60 Packt Open Source Survey Results – PACKT Publishing, 2009, http://www.packtpub.com/article/packt-open-source-survey-results

Przykład programu fundującego

Jednym z najgłośniejszych ogólnych programów dotowania projektów *open source* ostatnich lat jest Google Summer of Code, nazywany w skrócie GSoC. Program jest kierowany do studentów, którzy chcą pozyskać sponsoring dla pracy wykonywanej w ramach jednego z wybranych przez Google otwartych projektów. Taką pracę należy wykonać w okresie wakacji, a każdy z zainteresowanych musi wysłać szczegółowy plan realizacji oraz kryteria weryfikujące jego wykonanie. Na każdego zaakceptowanego studenta poświęcane jest 5000 $, z czego 4500 $ trafia do niego, a 500 $ do przewodzącej w danym projekcie organizacji. Pierwsza edycja Google Summer of Code odbyła się 2005 roku, a odzew zainteresowanych zaskoczył samych organizatorów. Do programu zgłosiło się około 8 740 projektów, co spowodowało zwiększenie liczby miejsc dla studentów z 200 do 400[61]. Liczba udanych realizacji była wysoka (na średnim poziomie 80%). Choć zdarzały się projekty w ramach, których stopień udanych realizacji był znacznie niższy jak np. Ubuntu 64% oraz KDE 67%. Firma Google organizuje kolejne edycje każdego roku, aż po dzień dzisiejszy, a liczba miejsc dla studentów w programie sukcesywnie wzrasta. Statystyki kolejnych edycji przedstawione są w poniższej tabeli 4.5.A.

	2005	**2006**	**2007**	**2008**
Przyjęte organizacje OSS	40	100	130	175
Przyjęci studenci	400	600	900	1125
Poświęcone fundusze	2 mln $	3 mln $	4,5 mln $	5,6 mln $

Tab.4.5.A. Przebieg kolejnych edycji GSoC[62]

61 Google's Summer of Code concludes - Bruce Byfield, 2005, http://www.linux.com/articles/48232

62 Google Summer of Code 2009 Frequently Asked Questions – oficjalny FAQ dla edycji 2009, http://socghop.appspot.com/document/show/program/google/gsoc2009/faqs

Przykład sponsorowanych fundacji

W świecie *open source* istnieje wiele fundacji będących parasolem dla otwartych podprojektów spełniających ich cele statutowe (patrz roz. 3.7). Mimo że większość z nich utrzymuje się z jakichś form dotacji, można je podzielić ze względu na stosunek środków, które poświęcają na utrzymanie infrastruktury wspierającej pracę, do sumy wynagrodzeń programistów wypłacanych bezpośrednio przez fundację. Również stosunek dotacji od indywidualnych użytkowników do funduszy przekazanych od firm oraz instytucji stanowi ciekawą charakterystykę. Poniższa tabela 4.5.B przedstawia zestawienie oparte na rocznych raportach finansowych z roku 2008 od czterech najpopularniejszych tego typu fundacji związanych z *open source*.

	Free Software Foundation[63]	**Mozilla Foundation**[64]	**Apache Software Foundation**[65]	**WikiMedia Foundation**[66]
Cele statutowe	promocja i ochrona wolnego oprogramo-wania	wsparcie organizacyjne, prawne i finansowe dla projektów Mozilla	wspieranie projektów open source zarządzanych merytokra-tycznie	opieka nad otwartymi projektami opartych na idei Wiki
Roczny przychód	1 mln $	86,5 mln $	200 000 $	8,6 mln $
Dotacje firm i instytucji	>20%	96,60%	>80%	BD
Dotacje indywidualne	BD	0,05%	BD	>53%
Najwięksi darczyńcy	Sun, IBM, HP, Google	Google (ok. 80%), Yahoo, Amazon	Google, Yahoo, Microsoft	BD

63 Free Software Foundation – Charity Navigator, 2009, http://www.charitynavigator.org/index.cfm?bay=search.summary&orgid=8557
64 http://www.mozilla.org/foundation/documents/mf-2008-audited-financial-statement.pdf
65 Apache Public Records, http://www.apache.org/foundation/records/
66 http://upload.wikimedia.org/wikipedia/foundation/4/4f/FINAL_08_09From_KPMG.pdf

	Free Software Foundation	**Mozilla Foundation**	**Apache Software Foundation**	**WikiMedia Foundation**
Koszt administracji	9%	19%	5%	26%
Koszt promocji	10%	12%	<0,01%	7%
Koszt infrastruktury	BD	4%	49%	>60%
Koszt wynagrodzeń programistów	BD	62,00%	0%	0%

Tab.4.5.B. Porównanie budżetów fundacji

Jak widać na powyższym zestawieniu, udział donacji od tak zwanych drobnych darczyńców, jest o wiele mniejszy niż to sugerowało wiele obiegowych opinii. Należy także zauważyć, że Mozilla jest dosyć specyficzną fundacją na tle innych, jako że przeznacza swoje środki głównie na zatrudnionych programistów, a zarazem jest bardzo uzależniona od finansowania jednego podmiotu. Z drugiej strony, nawet po utracie 80% przychodów fundacja Mozilla nadal posiadałaby największy przychód z wszystkich fundacji w zestawieniu. Google realizuje poprzez te dotacje strategię tzw. *lidera strat (ang. loss-leader)*, która to zostanie omówiona w następnym rozdziale. Dosyć interesujące jest podobieństwo struktury kosztów dwóch fundacji charakteryzujących się bardziej merytokratycznymi formami zarządzania (patrz roz. 3.3): Apache i WikiMedia. Otóż obie koncentrują się głównie na wsparciu infrastruktury oraz przyjmują jako swoją zasadę brak finansowania z budżetu fundacji ludzi biorących udział w projekcie. Warto także nadmienić, że Google pełni istotną role w budżetach wszystkich wymienionych fundacji.

4.6. Cele biznesowe nie przynoszące bezpośredniego zysku

W tym rozdziale omówione zostaną cele biznesowe firm zaangażowanych w projekty *open source*, które nie przynoszą bezpośrednich zysków. Jednym z

najczęściej dostrzeganych tego typu celów jest obniżenie kosztów produkcji oprogramowania. Jak już zostało wspomniane w rozdziale 3.1, otwarcie projektu i sukces w obudowywaniu go społecznością, mogą w istotny sposób zmniejszyć koszty rozwoju danego produktu. W branży IT można znaleźć liczne przykłady, gdzie świadome podjęcie takiej taktyki przyniosło wymierne efekty:

- Tokeneer System – system bezpieczeństwa oparty na biometryce. Amerykańska Agencja Bezpieczeństwa umyślnie rozpoczęła to przedsięwzięcie jako otwarty projekt, na podstawie przekonania, że tego typu metodologie są najbardziej efektywne kosztowo przy wytwarzaniu systemów o wysokich wymaganiach bezpieczeństwa[67].
- Maemo – przedsięwzięcie sponsorowane przez Nokię, stanowiące projekt dystrybucji (patrz rozdział 3.8) stosu technologicznego zasilającego Nokia N810 Internet Tablet. Suma oprogramowania zawartego w tym stosie wynosi około 10 milionów linii kodu, z czego 85% pochodzi z oprogramowania *open source*, natomiast pozostałe 15% zostało napisane przez pracowników Nokii. Połowa ze wspomnianych 15% została otwarta dla społeczności. Wykorzystując model COCOMO szacuje się, że takie podejście zapewniło Noki oszczędności rzędu 228 milionów dolarów[SME].

Zazwyczaj decyzja o otwarciu danego składnika oprogramowania przez firmę komercyjną, powinna być zależna od odpowiedzi na dwa poniższe pytania[TEO]:

- Czy to oprogramowanie jest naszym aktywem, czy też pasywem?

  W pierwszym przypadku wymagana jest głębsza analiza, ponieważ, jak wynika z informacji zawartych w poprzednich podrozdziałach, modele biznesowe *open source* niosą ze sobą wiele zagrożeń. W drugim przypadku należy przejść do kolejnego pytania.
- Czy to oprogramowanie jest elementem odróżniającym nas od konkurencji w oczach klienta?

67 NSA: Open source provides extreme security at lower cost – wiadomość cnet news, 07.10.2008, http://news.cnet.com/8301-13505_3-10059767-16.html

Z oczywistych względów oprogramowanie, które daje klientom coś, czego nie znajdą u konkurencji, nie jest czymś, co można otworzyć pochopnie. Takim elementem może być nowy wydajniejszy sterownik kontroli trakcji u producenta samochodów lub rozbudowany system podpowiedzi kontekstowych w serwisie do szukania lotów lotniczych. W obu tych przypadkach korzyści z otwarcia produktów mogłyby być niewspółmierne do ryzyka utraty klientów na rzecz konkurencji, która zaaplikowałaby nasze rozwiązania. Jednak co z oprogramowaniem, takim jak np. narzędzia programistyczne użyte do konstrukcji tego sterownika lub komponentu szablonującego odpowiedzialnego za generowanie wyglądu wspomnianego serwisu? Udostępnienie tych komponentów nie musi oznaczać utraty pewnej przewagi biznesowej w oczach klientów. Są one ważne w procesie wytwórczym, jednak nie muszą być źródłem innowacyjności. W takim przypadku konkurencja może się okazać najlepszym współpracownikiem, a grupy podmiotów potrafiących współpracować nad tego typu projektami wraz z upływem czasu zyskają większą konkurencyjność w stosunku do pozostałych. Dobrym przykładem tego typu relacji są HP i IBM, które często współpracują nad projektami *open source* będącymi produkcjami narzędzi oraz *middleware*, a jednocześnie żarliwie konkurują ze sobą w wielu innych segmentach rynku IT[TEO]. Częstą obawą przed rozpoczęciem tego typu współpracy nad wspólnym oprogramowaniem jest fakt, że mogą z niego czerpać korzyści także konkurencyjne firmy, bez ponoszenia jakichkolwiek kosztów. Alternatywą dla otwartych projektów, które mogą zapobiec takiej sytuacji, jest założenie konsorcjum. Mimo, że wydaje się to atrakcyjną formą współpracy, historia branży IT wskazuje znacznie więcej spektakularnych porażek takich przedsięwzięć niż sukcesów. Można by nadmienić konsorcja, jak chociażby Taligent i Monterey, które miały wykonać zamiennik dla, odpowiednio, systemu Unix i Common Desktop Enviorment. Wszystkie te projekty zostały *de facto* zastąpione przez projekty *open source*[TEO].

Innym istotnym celem biznesowym, jaki z dużą dozą prawdopodobieństwa powinno nam zapewnić otwarcie projektu, jest obniżenie kosztów marketingu oraz znaczne poprawienie relacji publicznych. Dzieje się tak z powodu szeregu czynników takich jak:

- Przesunięcie postrzegania projektu z komercyjnego produktu w stronę dobra publicznego. Zostało to wspomniane w rozdziale 3.1.

- Ogólne obniżenie kosztów pozyskania klienta, o których wspomniano w rozdziale 4.1.
- Darmowy rozgłos w stosunkowo aktywnych środkach przekazu, zapewnianych przez serwisy związane z hostingami projektów, jak np. SourceForge i Freashmeat, a także portali społecznościowych typu Slashdot.org.
- Duża skuteczność tzw. *marketingu szeptanego*, świadczonego spontanicznie przez społeczność zbudowaną wokół samego projektu[COS].

Darmowy marketing tego typu (często nazywany partyzanckim) może pozwolić przesunąć więcej środków na badania i rozwój. Przeciętny współczynnik kosztów marketingu do kosztów prac rozwojowych w tradycyjnych firmach wynosi około 2,3. Oznacza to, że na każdą złotówkę wydaną na rozwój przypadają 2 złote i 30 groszy wydane na działania marketingowe. Często współczynnik ten jest bardzo wysoki w porównaniu z poszczególnymi firmami open source[68]. Dla przykładu w segmencie systemów CRM, dla firmy Salesforce współczynnik ten wynosi 6. Natomiast dla open source'owych produktów CRM szacuje się go na poziomie 0,6. Mówiąc prościej, oznacza to, że na każdą złotówkę wydaną na rozwój przypada jedynie 60 groszy wydanych na marketing[69].

W ostatnich latach wspomina się często strategię utożsamianą bardziej z zamkniętymi projektami aniżeli *open source* o angielskiej nazwie *loss-leader*[COS]. W polskim tłumaczeniu można nazwać to określeniem: lider strat. Strategia ta ma na celu ekstensywne finansowanie produktów nie przynoszących zysków, aby: stworzyć nowy rynek, zmienić zasady gry na obecnym rynku lub podkopać dochody konkurencji w jej najbardziej dochodowym sektorze. Jednym z najbardziej widocznych przykładów takich działań są starania Google prowadzone w kierunku zmiany doświadczeń użytkowników z przeglądarkami internetowymi. Ponieważ firma opiera swoją działalność głównie na aplikacjach webowych jest w jej interesie, aby promować poszanowanie dla standardów W3C oraz rozszerzanie możliwości przeglądarek. Nie ma lepszego sposobu, jak robienie tego poprzez wzmaganie konkurencji w tym sektorze. Jako realizację tej strategii Google od początku

68 A New Bread of P&L: The Open Source Business Financial Model - Larry Augustin.
69 The Next Wave of Open Source: Applications - prezentacja z konferencji GOSCON 2005, Larry Augustin.

intensywnie finansuje fundację Mozilla i ich flagowy produkt Firefox. Zgodnie z oświadczeniem Fundacji Mozilla, z niemal 67 milionów dolarów dochodu na rok 2006, około 66% tej sumy stanowiło dotację od giganta wyszukiwarek internetowych[70]. Jednak to dopiero początek realizacji strategi *loss-leader* dla Google. W roku 2008 korporacja wypuściła na rynek własną przeglądarkę Chrome na bardzo liberalnej licencji BSD ogłaszając, że przy jej pomocy zamierzają wspomóc innowacyjność rozwoju sieci poprzez dodanie szybkiej implementacji języka JavaScript[71]. W tym samym roku została przedłużona umowa wsparcia dla fundacji Mozilla na kolejne 3 lata[72]. Wysiłki giganta nie poszły na marne i obie przeglądarki posiadają łącznie 37% rynku sukcesywnie podmywając pozycję lidera, którą od dłuższego czasu zajmuje Internet Explorer[73] z wynikiem 55% w roku 2010. Marginalizacja przeglądarki Microsoftu wydaje się być jednym z najważniejszych celów biznesowych Google, który z biegiem czasu zaczyna konkurować na niemal wszystkich operowanych przez siebie segmentach rynku z korporacją z Redmond.

Za kolejny duży przykład realizacji strategii *loss-leader* można uznać projekt Eclipse firmy IBM. Źródłem tego projektu jest zamknięte IDE o nazwie Visual Age, rozwijane i wykorzystywane głównie do realizacji wewnętrznych projektów. Źródła tego środowiska, których wartość wyceniano na 40 milionów dolarów[74], zostały uwolnione w roku 2001 wraz z założeniem Eclipse Foundation, kierującą jego rozwojem. Jak wynika z oficjalnych źródeł IBM otwarcie na rynku miało na celu ujednolicenie rynku środowisk programistycznych, a sama fundacja miał stać się *Apachem narzędzi programistycznych*. Poprzez wzorowanie na fundacji Apache, firma miała nadzieje maksymalnie przyspieszyć rozwój Eclipse poprzez zachęcanie innych firm do współpracy w merytokratycznych strukturach (patrz. roz. 3.3). Założenia okazały się trafne: środowisko programistyczne w roku 2005 zajęło 65%

70 Google funds hold Firefox fate - Cade Metz, artykuł The Register, 2007, http://www.theregister.co.uk/2007/10/25/mozilla_releases_2006_financial_statement/
71 A fresh take on the browser – oficjalny blog Google, 09.01.2008, http://googleblog.blogspot.com/2008/09/fresh-take-on-browser.html
72 Mozilla Extends Lucrative Deal With Google For 3 Years - artykuł TechCrunch, 28.08.2008, http://www.techcrunch.com/2008/08/28/mozilla-extends-lucrative-deal-with-google-for-3-years/
73 StatCounter Global Stats 2010 - http://gs.statcounter.com/#browser-ww-daily-20100101-20100105-bar
74 IBM makes $40 million open-source offer - Staff Writer, CNET News, 2001, http://news.cnet.com/IBM-makes-40-million-open-source-offer/2100-1001_3-275388.html

rynku IDE związanych z platformą Java[75], a także zyskało wielu użytkowników związanych z innymi językami programowania, takimi jak C++ lub PHP. Jak już wspomniane zostało w rozdziale 4.3, dzięki swojej modularnej i otwartej architekturze, Eclipse zbudował również pokaźny ekosystem firm uczestniczących w projekcie. Pozwoliło to również firmie IBM znacznie obniżyć koszty związane z rozwojem swojego podstawowego narzędzia.

75 Eclipse gains market share in 2005 - blog Ian Skerrett, 2006, oparte na badaniach BZ Media, http://ianskerrett.wordpress.com/2006/03/11/eclipse-gains-market-share-in-2005/

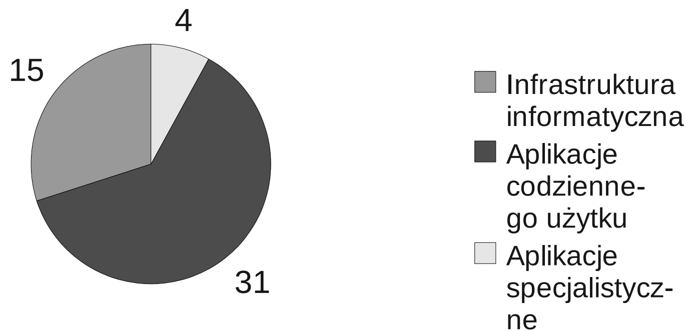
Wpływ typu oprogramowania

Rozdział 5

W ramach tej pracy podzielono na kategorie najczęściej spotykane struktury organizacyjne projektów *open source* oraz modele biznesowe firm biorących w nich udział. Jednak nadal trudno odpowiedzieć na pytanie: jaka jest skuteczność owych technik w kontekście typu wytwarzanego przez nie oprogramowania? Bez wątpienia otwarte metodologie najbardziej zakorzeniły się w projektach wytwarzających narzędzia programistyczne oraz ogólnie szeroko pojmowaną infrastrukturę IT. Świadczy o tym chociażby popularność produktów, które od początku były rozwijane jako *open source*, jak np. Apache Web Server, MySQL, Linux, JBoss. Projekty te okazały się źródłem sukcesu wielu firm z branży IT, jak chociażby Red Hat, który niedawno wszedł do prestiżowego indeksu giełdowego S&P 500[76]. Większość historii biznesowych przytoczonych w niniejszej pracy opiera się również na podmiotach działających w tej branży. W projektach wytwarzających infrastrukturę IT można znaleźć przypadki wykorzystania wszystkich wymienionych tutaj struktur organizacyjnych. Także firmy oraz instytucje organizujące owe projekty mają do swojej dyspozycji całą gamę wymienionych tutaj praktyk biznesowych. Fakt ten nie powinien dziwić, jako że z tego właśnie sektora IT wywodzą się pionierzy idei otwartego i wolnego oprogramowania (patrz. 2.1).

Jednak nie tylko otwarte oprogramowanie stanowiące infrastrukturę IT zdobywa popularność. Jeżeli rozdzielić 50 najczęściej ściąganych projektów na SourceForge.net pod względem typu oprogramowania, jakie wytwarzają, otrzymamy następujący rozkład (rys.5.A).

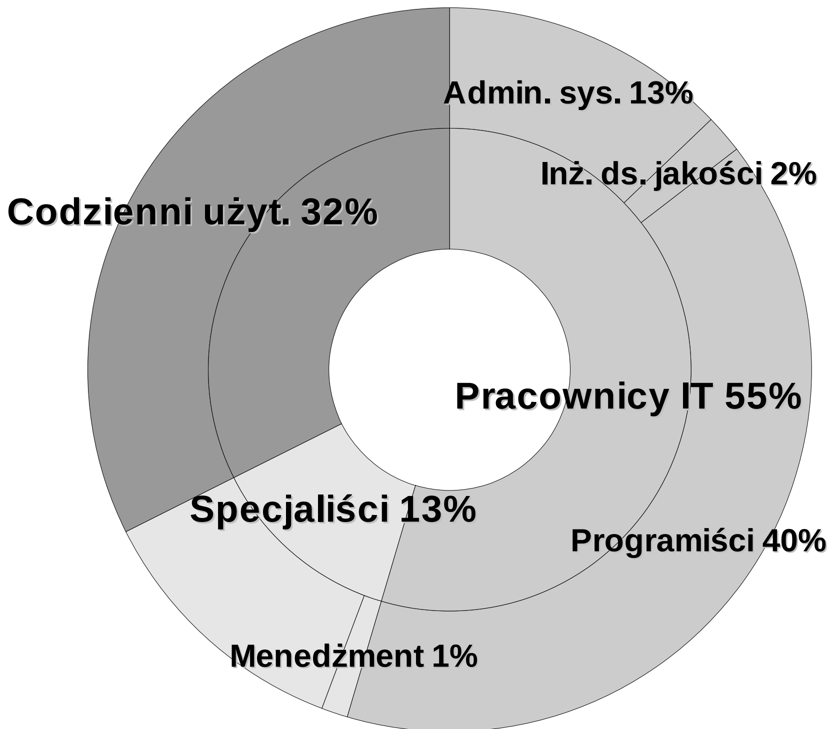
76 S&P 500 to drop CIT Group, add Red Hat – Reuters, informacja prasowa, 2009, http://www.reuters.com/article/idUSWEN084920090720



Rys.5.A. 50 najpopularniejszych projektów

Jak widzimy 62% z nich stanowią aplikacje codziennego użytku służące do, między innymi, współdzielenia plików przez sieci P2P, kompresji danych, obsługi multimediów i komunikacji. Narzędzia programistyczne stanowią tutaj 30%, co należy także uznać za wysoki wynik, gdy zestawiamy je z ogółem oprogramowania. Bardzo niski poziom (8% popularności) zaliczają natomiast specjalistyczne aplikacje dla biznesu i nauki.

Pogłębiając analizę danych z SourceForge należy sprawdzić, jak wygląda aktywność projektów wytwarzających poszczególne typy oprogramowania. Biorąc 28 800 najaktywniejszych z około 215 000 umieszczonych tam projektów i rozbijając je względem grupy docelowej, otrzymamy poniższe zestawienie (rys. 5.B).

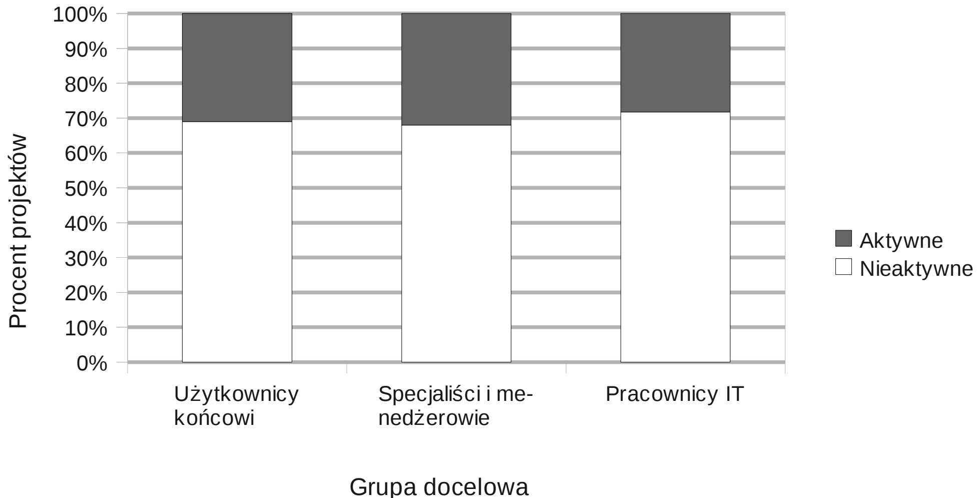


Rys.5.B. 28 tysięcy najaktywniejszych projektów

Jak można zauważyć, proporcje znacznie się odwracają, gdy spojrzymy na projekty z tej perspektywy. Miano najaktywniejszych przypada producentom oprogramowania nakierowanego na pracowników IT. Stosunkowo niski procent software'u celującego w potrzeby przeciętnych użytkowników końcowych potwierdza fakt, że popularność w świecie *open source* nie przekłada się bezpośrednio na aktywność prac w projekcie. Należy pamiętać, że wypadkową tej aktywności jest połączenie dwóch czynników: zaangażowania społeczności wokół projektu oraz/lub liczby programistów opłacanych bezpośrednio przez organizatorów projektu.

Przedstawiony rozkład aktywności mógłby sugerować, że projekty nakierowane na użytkowników końcowych cechują się wyższym ryzykiem niepowodzenia w stosunku do reszty. Jednak bardziej wiarygodne rozstrzygnięcie tej

kwestii można uzyskać poprzez analizę stosunku aktywnych do nieaktywnych projektów w danej grupie (rys. 5.C). Tak jak na końcu rozdziału 3.1, za próg nieaktywności uznaje się wynik poniżej 80% według współczynnika aktywności SourceForge.

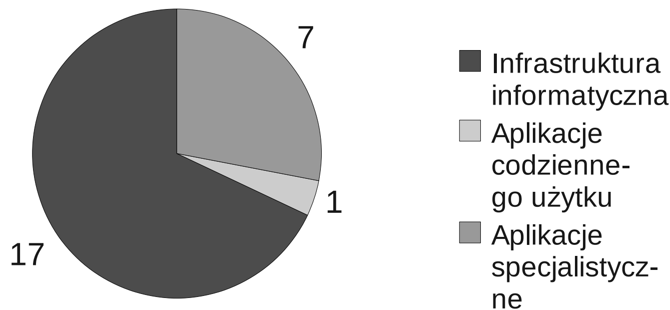


Rys.5.C. Stosunek projektów aktywnych do nieaktywnych

Z otrzymanych wyników widać, że ryzyko zamarcia projektu jest podobne. Na podstawie zestawień na rysunku 5.B można jedynie wnioskować, że statystycznie społeczności wytwarzające specjalistyczny software oraz (tak zwane) infrastruktury IT, cechują się wyższą aktywnością.

Kolejnym aspektem, który różnicuje typy projektów *open source* jest ryzyko nieosiągania zysków przez firmy, które chcą oprzeć na otwartym oprogramowaniu zdecydowaną część swojego modelu biznesowego. Jak omówiono w rozdziale 4 większość z nich wymaga inwestycji w dany projekt, a modele te mają różny stopień zastosowalności do różnych typów wytwarzanego oprogramowania. Z raportu The VAR Guy's[77] wynika, że 25 najbardziej obiecujących biznesowo firm *open source* produkuje:

77 The Open Source 50 - The VAR Guy's, raport, 2009, http://www.thevarguy.com/the-open-source-50/the-open-source-50-listed-a-to-z/

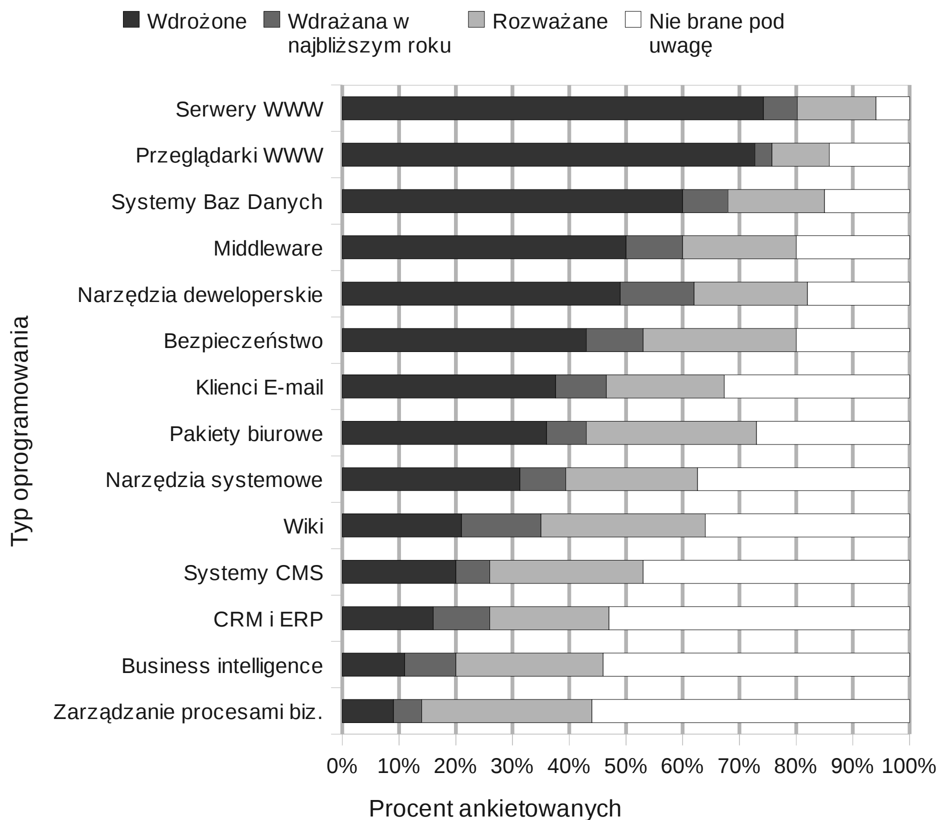


Rys.5.D. 25 najbardziej obiecujących firm

Potwierdza to tezę, iż firmy zaangażowane w otwarte narzędzia programistyczne mają największe szanse na sukces na rynku. Dobrze prezentują się również aplikacje specjalistyczne, jednak należy zaznaczyć, że 6 z nich jest ukierunkowanych na odbiorców biznesowych. Zaskakuje natomiast fakt, że mimo swojej popularności aplikacje codziennego użytku zdają się nie stwarzać zbyt wielu możliwości biznesowych. Jedynym reprezentantem tego typu oprogramowania, pojawiającym się w rankingu, jest projekt Zimbra, który jest także produktem powiązanym w pewnym stopniu ze sferą biznesową. Właściciele Zimbry, początkowo Yahoo! teraz VMware, zbudowali model biznesowy na bazie tego produktu oparty o up-selling oraz usługi powiązane (patrz rozdziały 4.3 i 4.4).

Jak wynika z powyższych informacji, firmy czerpiące zyski z oprogramowania *open source* w dużej mierze celują w odbiorów biznesowych oraz firmy z sektora IT. Ten fakt potwierdzałby raport CIO Insight z roku 2007[78], w którym ankietowane były przedsiębiorstwa z sektora małych i średnich przedsiębiorstw. Wynika z niego, że tylko 3% z nich nie zamierza wydać jakichkolwiek funduszy na wdrożenie lub utrzymanie oprogramowania open source. Natomiast przeciętna kwota, jaką te firmy zamierzają wydać, to ponad 500'000 $. W raporcie znajdziemy również zestawienie popularności danych typów oprogramowania w tym sektorze.

78 CIO Insight May 2007 - http://www.cioinsight-digital.com/cio/200705/?pg=85



Rys.5.E. Popularność w biznesie

Z powyższego zestawienia płynie potwierdzenie dla popularności otwartej infrastruktury IT, która zajmuje 4 z 5 najwyższych kategorii. Popularność przeglądarek WWW można utożsamić *de facto* z programem FireFox oraz wykluczyć go z kategorii zarabiających dzięki popularności w tym sektorze. Bierze się to z faktów, które zostały już omówione przy analizie budżetu **fundacji Mozilla w** rozdziale 4.5 oraz na końcu rozdziału 4.6. Zaskakująca jest natomiast niska popularność specjalistycznego oprogramowania dla biznesu, które znajduje się na samym końcu zestawienia. Zgodnie z rozdziałem 4 większość omawianych tam modeli biznesowych opiera się na zasadzie „5% klientów generuje 90% zysków". Zasada ta w stosunku do oprogramowania *open source* zazwyczaj przejawia się w 1% płacących użytkowników, którzy staną się najbardziej dochodowymi klientami. Jednak porównując zestawienia z rysunków 5.E i 5.D można dojść do wniosku, że

otwarte oprogramowanie biznesowe cechuje się tutaj znacznie lepszym współczynnikiem. Z raportu CIO Insight warto także nadmienić, że 36% firm uważa, iż nie rozważałoby wykorzystania otwartego oprogramowania bez wsparcia ze strony dużych firm, jak np. IBM, Red Hat, itp. Przemawiałoby to, jako istotny argument, za stosowaniem modeli biznesowych opartych na powiązanych usługach (patrz roz. 4.4).

Wróćmy jednak do oprogramowania dla użytkownika końcowego. Aby zweryfikować domniemanie o tym, że mimo swojej popularności tego typu software ma ograniczone możliwości w zastosowaniu modeli biznesowych open source, należy zestawić ze sobą 5 najpopularniejszych produktów.

	Firefox	**OpenOffice**	**7zip**	**BitTorrent**	**Video LAN**
Liczba pobrań	1 265 mln	98 mln	78 mln	~ 60 mln	373 mln
Główna firma fundująca	Google	Sun Microsystems	brak	BitTorrent	Anevia (firma autorów projektu)
Najaktywniejsi programiści	pracownicy Mozilla	pracownicy Oracle	Igor Pavlov	pracownicy BitTorrent	grupa programistów wywodzących się z École Centrale Paris
Modele biznesowe gł. fundatora	loss-leader, cross-selling	loss-leader, cross-selling	dotacje	cross-selling, powiązane usługi	cross-selling, powiązane usługi

Tab.5.A. Najpopularniejsze aplikacje dla użytkownika końcowego

Dwa najpopularniejsze produkty: Firefox i OpenOffice, stanowią pewną ikonę *open source* na desktopach. Jednak, jak widać z powyższego zestawienia, obydwa projekty są silnie uzależnione od swoich korporacyjnych patronów oraz nie przynoszą bezpośrednich zysków. Obie firmy realizują tutaj strategię *loss-leader* (patrz rozdział 4.6), aby odpowiednio wpłynąć na segment rynku przeglądarek oraz pakietów biurowych. Zarówno Firefox, jak i OpenOffice są przedmiotami **c**ross-selling'u (patrz

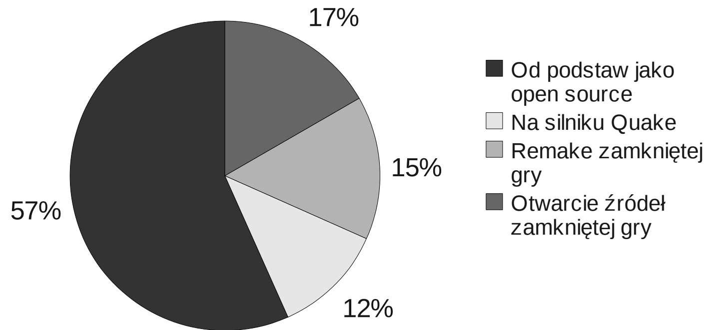
rozdział 4.3). Google poprzez dotowanie większości budżetu Mozilla gwarantuje sobie status domyślnej przeglądarki. Oracle natomiast argumentuje sprzedaż swoich tak zwanych *enterprise systems* (ang. systemów korporacyjnych) dostępnością dobrze integrowanego pakietu biurowego, rozwijanego pod patronatem firmy.

Ciekawym przykładem jest firma BitTorrent, która jest twórcą standardu sieci P2P o tej samej nazwie. Są oni także autorami pierwszego otwartego klienta dla owej sieci. Przedsiębiorstwo czerpie swoje główne zyski z cross-selling'u serwerów dla tej sieci oraz certyfikacji specjalizowanych pod nie urządzeń. Natomiast 7zip oraz Video LAN są najbardziej społecznościowymi przedsięwzięciami pośród tego zestawienia. Główną siłą napędową pierwszego jest jego autor Igor Pavlov, który jest życzliwym dyktatorem projektu (patrz rozdział 3.2). Drugie przedsięwzięcie, Video LAN, rozpoczynało swoją działalność jako projekt studentów z École Centrale Paris. Gdy ich główny produkt VLC media player zyskał popularność, otworzyli oni firmę Anevia, która **cross-sell'uje** z nim serwery multimedialne oraz usługi z zakresu strumieniowania mediów.

Z powyższych porównań można wnioskować, że otwarte oprogramowanie dla użytkownika końcowego samo w sobie nie stanowi podatnego gruntu pod budowę modeli biznesowych. Jest ono natomiast ważnym narzędziem przy argumentacji oraz promocji innych produktów. W przytoczonych historiach biznesowych dominuje cross-selling produktów i usług kierowanych do odbiorcy korporacyjnego oraz zespołów programistycznych. Istotnym wyjątkiem jest tu Google, który czerpie większość zysków ze sprzedaży reklam internetowych.

Jeden z typów oprogramowania, który nie został wyszczególniony dotychczas w obecnym rozdziale, jest elektroniczna rozrywka oraz gry video. Jest to branża, w której widać najmniejszą adaptację otwartych metodologii. Trudno także znaleźć historie firm, którym udało się zbudować efektywny model biznesowy wokół gry *open source*. Jedną z przyczyn takiego stanu rzeczy może być fakt, że branża ta głównie opiera się na sprzedaży instalowanych pakietów oprogramowania. Modele opisane w rozdziale 4 nie mają szerszego zastosowania w świecie *open source*. Na liście gier tworzonych przez otwarte społeczności, które wikipedyści uznali za wystarczająco wartościowe, aby wymienić w swojej encyklopedii, znajduje się

jedynie 68 pozycji[79]. Pod względem źródła ich powstania można je podzielić na następujące kategorie.

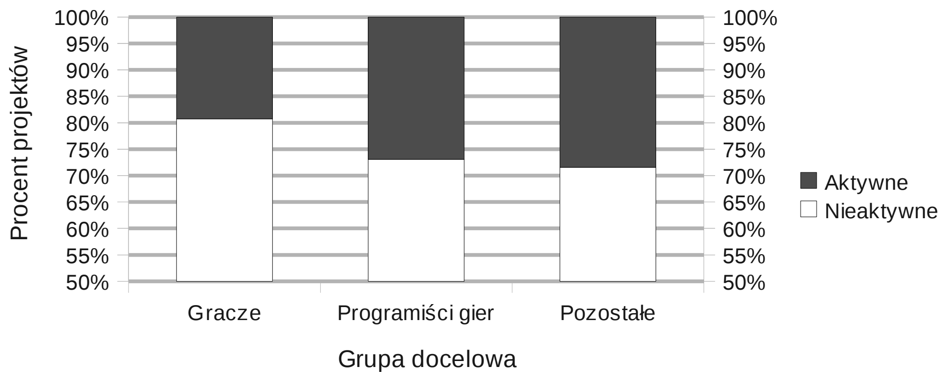


Rys.5.E. Najpopularniejsze gry a źródło ich powstania

Mimo tak małej liczby pozycji widać, że niemal połowa z otwartych gier jest pochodną zamkniętych produktów. Również jeżeli przyjrzeć się najpopularniejszym, które powstawały od początku jako *open souce*, jak np. Cube 2: Sauerbratem, FreeCiv, UFO: Alien Invasion lub The Battle for Wesnoth, można zauważyć, że odstają one pod względem technicznym od swoich zamkniętych odpowiedników o jakieś 4 - 8 lat. Należy jednocześnie zaznaczyć, że istotny wpływ na otwarte gry ma gama silników Quake, których kolejne wersje są uwalniane przez id Software, na 5 lat po dacie oficjalnej premiery. Bazują na nim między innymi trzy najpopularniejsze gry *open source* z gatunku FPS: Tremulous, Alien Arena oraz Nexuzin.

Należy jednak sprawdzić, w jaki sposób projekty związane z grami i rozrywką radzą sobie z tworzeniem społeczności wokół siebie. W bazie danych SourceForge znajduje się ponad 2100 pozycji związanych z tą kategorią, z czego 55% jest kierowanych do końcowych użytkowników, a 29% do programistów. Badając oprogramowanie pod względem aktywności (opierając się na współczynniku 80%) w kontekście tych dwóch kategorii oraz porównując je z ogółem oprogramowania na tym hostingu, otrzymane zostanie następujące zestawienie (rysunek 5.F).

79 List of open-source video games – artykuł na angielskiej Wikipedii, wersja z 10.01.2010, http://en.wikipedia.org/w/index.php?title=List_of_open-source_video_games&oldid=338771420



Rys.5.F. Grupa docelowa a aktywność projektu

Analizując otrzymane wyniki można zauważyć, że projekty związane z produkcją końcowych produktów dla graczy mają o niemal 10% większe ryzyko zamarcia projektu w stosunku do oprogramowania dla pozostałych dwóch grup. Ryzyko to natomiast nie jest znacząco powiększone w stosunku do narzędzi programistycznych stosowanych do wytwarzania gier. Zwiększone ryzyko w projektach samych gier prawdopodobnie jest wynikiem dużej rozbieżności pomiędzy metodykami *open source*, a tymi stosowanymi w branży. Większość tytułów jest wykonywanych w tzw. metodyce kaskadowej (ang. *waterfall*) i wypuszczanych na rynek dopiero, gdy osiągną swój ostateczny kształt. Natomiast jak zostało napisane w rozdziale 3.1, projekty *open source* są nakierowane bardziej w stronę metodyk zręcznych (ang. *agile*), gdzie sam produkt powinien dojrzewać wraz z jego społecznością użytkowników.

Podsumowanie

Rozdział 6

Jak wynika z informacji zawartych w niniejszej książce, świat open source przez ponad 20 lat swojego istnienia zdołał wypracować wiele specyficznych dla siebie form organizacji oraz finansowania. Wiele udanych połączeń tych dwóch aspektów przyczyniło się do utworzenia dynamicznie rozwijających się projektów, w których otwarte metodyki współpracy przybrały globalną skalę. Zapewniło to także krociowe zyski dla firm budujących swoją rentowność na ich bazie. Jednak, jak widać, praktyki stosowane w otwartych projektach nie stanowią panaceum na problemy związane z szeroko pojętym wytwarzaniem oprogramowania. Ryzyko niepowodzenia przedsięwzięć tego typu jest porównywalne do tego spotykanego w zamkniętych wewnątrzfirmowych projektach. Również, pomimo faktu, że istnieją rynki, w których otwarte produkty śmiało konkurują z ich pierwotnymi liderami, efektywność *open source* nie jest niezależna od typu wytwarzanego oprogramowania. Istnieją segmenty, w których zastosowanie wymienionych w tej pracy praktyk jest ograniczone lub niemożliwe, zarówno w sferze struktur organizacyjnych, jak i modeli biznesowych.

Jak już wspomniano w 5 rozdziale, metodyki oraz modele biznesowe *open source* najlepiej sprawują się w projektach wytwarzających narzędzia programistyczne i szeroko pojętą infrastrukturę IT. W przypadku aplikacji codziennego użytku widać, że są one w stanie utrzymać zaangażowanie społeczności w ich rozwój, jednak trudno jest dla nich zastosować modele biznesowe przynoszące bezpośrednie zyski. Oprogramowanie tego typu jest zależne od celów biznesowych powiązanych z nimi firm oraz motywacji nieopłacanych wolontariuszy. Wyjątkiem są tutaj aplikacje biznesowe, takie jak systemy ERP i CRM, które potrafiły zidentyfikować pewien ograniczony zakres efektywnych modeli biznesowych. Natomiast praktyki *open source* są bardzo nieefektywne dla branży gier komputerowych oraz branży rozrywki elektronicznej. Nie odnotowano tutaj żadnych większych sukcesów firm, których modele biznesowe miałyby bazować całkowicie na otwartym oprogramowaniu. Także największe projekty rozwijające otwarte gry nie są wstanie nawiązać realnej konkurencji z ich zamkniętymi odpowiednikami.

Literatura

[AGP] About the GNU Project - Richard Stallman, artykuł, http://www.gnu.org/gnu/the-gnu-project.html

[CatB] The Cathedral and the Bazaar - Eric S. Raymond, esej, 2000, http://www.catb.org/%7Eesr/writings/cathedral-bazaar/cathedral-bazaar/

[COS] The Commercial Open Source Business Model - Dirk Riehle, SAP Research, SAP Labs LLC, artykuł, 29.07.2009, http://dirkriehle.com/publications/2009/the-commercial-open-source-business-model/

[EPM] Effective Project Management: Traditional, Adaptive, Extreme, Third Edition - Robert K. Wysocki, Rudd McGary, książka, 2005.

[FSD] The Free Software Definition - The GNU Project, wersja 1.80, http://www.gnu.org/philosophy/free-sw.html

[GSH1] Free/Libre/Open Source Software Worldwide impact study: FLOSSWorld. FLOSSWorld - Gosh, prezentacja projektu, 2005, http://www.flossproject.org/papers/20051217/flossworld-intro3.pdf

[GSH2] Economic impact of FLOSS on innovation and competitiveness of the EU ICT sector - Gosh, 2006, http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf

[MMM] The Mythical Man-Month - Fred Brooks, esej, 1995.

[OLP] Open Life: The Philosophy of Open Source - Henrik Ingo, książka, 2006, http://www.openlife.cc/

[OSD] The Open Source Definition - The Open Source Initiative, artykuł, http://www.opensource.org/docs/osd

[POS] Producing Open Source Software - Karl Fogel, książka, 2009, http://producingoss.com/

[SME] Guide for SMEs - Carlo Daffara, The FLOSSMetrics Consortium. http://flossmetrics.org/sections/deliverables/docs/deliverables/WP8/D8.1.4-SMEsGuide.pdf

[TES] The economics of sharing: Open source and Beyond - Josh Lerner, Jean Tirole, NBER Working Paper No. 10956, 2004, http://www.nber.org/papers/w10956

[TEO] The Emerging Economic Paradigm of Open Source - Bruce Perens, George Washington University, artykuł, 2005, http://perens.com/works/articles/Economic.html

[WhLG] Why you shouldn't use the Lesser GPL for your next library - Free Software Foundation, artykuł, 2007, http://www.gnu.org/philosophy/why-not-lgpl.html

[DA] Badania ankietowe przeprowadzone przez autora książki w roku 2010. Ich opis znajduje się w Dodatku A

W przypisach dolnych, na poszczególnych stronach książki, znajduje się również ponad 70 odwołań innych źródeł, jak np. artykuły prasowe, informacje z serwisów poszczególnych projektów, raporty różnych organizacji, itp.

Dodatek A – badania ankietowe

Na poczet materiału potrzebnego do opracowania tej książki zostały wykonane dwa badania ankietowe:

- „Poglądy pracowników IT na oprogramowanie *open source*” (ang. „The view of the IT staff on open source software”);
- „Projekty *open source*” (ang. „Open source projects”).

Obydwie ankiety zostały opracowane w angielskiej wersji językowej i udostępnione na łamach serwisu www.openeo.biz. Silnikiem obsługującym nadsyłane odpowiedzi było otwarte oprogramowanie LimeSurvey. Pierwsza z ankiet została skierowana do pracowników branży IT z dbałością o promowanie jej na jak najbardziej neutralnych środkach przekazu. Poprzez neutralne, rozumie się media nie związane wyłącznie ze światem zamkniętego lub otwartego oprogramowania. Były to głównie tematyczne fora internetowe, takie jak np. cnet.com, InfoWorld lub Programmers Heaven oraz wiadomości wysłane na około 3 000 adresów email pozyskanych z blogów IT. Dzięki tej akcji uzyskano ponad 200 ankietowanych. Druga ankieta była natomiast kierowana do administratorów projektów *open source*. Środkami promocji ankiety były fora internetowe oraz listy mailingowe kilkudziesięciu serwisów zapewniających infrastrukturę dla otwartych projektów. Między innymi były to serwisy: SourceForge, Lunchpad, Savannah, Google Code, Apache, OpenSuse Build, Java.net, RubyForge, LuaForge. Informacja została również przekazana oficjalnymi kanałami niektórych z nich, jak na przykład:

- Java.net - http://java.net/story/adam-walczak-openeobiz-open-source-project-survey
- O'Reilly GMT - http://www.oreillygmt.co.uk/2009/12/openeobiz-survey-about-open-source.html
- SourceForge - http://twitter.com/sourceforge (dostęp: 15 grudnia 2009).

Mimo dużego rozgłosu uzyskano jedynie 43 w pełni wypełnione ankiety. Wśród respondentów znalazły się jednak czołowe postacie takich projektów, jak: FreeBSD, GNU m4, RapidSVN lub Apache OJB.

Skorowidz

7zip..122, 123
Affero GPL..35
Alien Arena..124
Anevia..122, 123
Apache............................25, 35, 49, 57, 58, 59, 60, 61, 62, 78, 79, 107, 108, 112, 116
Apache ActiveMQ..38, 57
Apache Ant..57
Apache Commons..57
Apache Derby..60
Apache Geronimo..79
Apache Software Foundation..57, 62, 107
AT&T..17
Battle for Wesnoth..124
Benevolent Dictator..51, 52, 53
Benevolent Dictator For Life..53
Berkeley Software Distribution..17
Bill Gates..21
BitTorrent..122, 123
Blender..54
Borland..97
Bruce Perens..21, 24
BSD..17, 18, 25, 30, 79, 80, 94, 112
Chrome..112
CodeGear..97
CodePlex..22
Common Desktop Enviorment..110
Computer Systems Research Group..17
copyleft..18, 30, 91
Couldspace..60
cross-selling..57, 64, 88, 93, 98, 104, 122, 123
Cube 2: Sauerbratem..124
David A. Wheeler..33
Debian..20, 29, 76, 77, 79, 80, 81, 82, 83, 84, 85
Eclipse..21, 62, 96, 97, 98, 112, 113
École Centrale Paris..122, 123
EGCS..75
Ekiga..38
Embarcadero..97
Eric S. Raymond..21, 24, 52, 132
EULA..12, 13, 16, 25, 31
Firefox..76, 112, 121, 122
fork..46, 47, 53, 62, 66, 73, 74, 75, 76, 80, 94, 96
Freashmeat..111

free software.............12, 22, 23, 24, 25, 26, 107, 132
FreeCiv.............124
freemium.............94, 97
freeware.............23, 24
FSF.............18, 24
GCC.............75
Gentoo.............76
Glassfish.............79
Gnome.............75, 78
GNU Project.............17, 18, 19, 20, 75, 78, 79, 80, 132
GoneME.............75
Google.............106, 107, 108, 111, 112, 122, 123
GPL.............18, 19, 25, 30, 31, 33, 35
Guido van Rossum.............53, 54
Harmony.............57
HP.............107, 110
Ian Mardock.............80
IBM.............13, 21, 60, 62, 96, 97, 98, 107, 110, 112, 113, 122
id Software, na.............124
Igor Pavlov.............122, 123
in-house development, community feedback.............62
Internet Explorer.............112
Internet Standards Process.............65, 66
Java.............21, 57, 65, 68, 69, 73, 79, 97, 101, 113
Java Community Process.............65, 68
Java,.............65
JavaScript.............112
JBoss.............26, 97, 101, 102, 104, 116
JBuilder.............97
LAMP.............79, 95
LGPL.............30, 93, 133
lider strat.............108, 111
Linus.............26
Linus Torvalds.............20, 23
Linux.............20, 26, 40, 54, 76, 79, 80, 103, 104, 105, 116
log4j.............57
loss-leader.............56, 108, 111, 112, 122
Maemo.............109
marketing szeptany.............92
Massachusetts Institute of Technology.............17
merytokracja.............42, 48, 53, 55, 56, 66, 81, 107, 112
Microsoft.............21, 107, 112
MIT.............18, 30, 33
Monterey.............110

Mozilla Foundation..76, 107, 112, 121, 122
Mozilla Suite..76
MySQL...21, 25, 40, 64, 79, 92, 116
MySQL AB...21
Nexuzin..124
Nokia...93, 109
Novell..21, 104
NUI Group...78
Ohloh...56
Open Source Initiative..21, 24, 25, 29
OpenOffice...13, 122
OpenOffice,..122
Oracle..21, 68, 122
Patrick Volkerding..54
PHP..79, 113
podejściem fazowym...94
podwójne licencjonowanie.......................................12, 25, 31, 47, 64, 90, 91, 93
Portable Apps...80
Projekt parasolowy...57, 77
Python...53, 54
Qt...13, 21, 64, 92, 93
Quake..124
Red Hat...20, 21, 76, 79, 101, 103, 104, 105, 116, 122
Richard Stallman...17, 18, 19, 23, 24, 25, 27, 132
Salesforce...111
Slackware...54, 76
Slashdot...111
Softlanding Linux System...80
Solaris..79
SourceForge...31, 49, 50, 95, 111, 116, 117, 119, 124
SugarCRM...95, 96
SugarCRM,..96
Sun Microsystems...13, 21, 49, 68, 93, 104, 122
SUSE..20, 76, 79, 103, 104, 105
Taligent..110
Tokeneer System..109
Tomcat...49, 57
Ton Roosendaal,...54
Tremulous..124
Trolltech...21, 92, 93
Ubuntu...76, 77, 79, 80, 106
UFO: Alien Invasion..124
Unix..17, 74, 79, 103, 110
up-selling...............................31, 57, 64, 76, 79, 88, 91, 93, 94, 95, 96, 97, 98, 120

up-selling`u....94
Video LAN....122, 123
Visual Age....112
VLC....123
VMware....120
vtiger....96
WAMP....79
WebSphere....97
wewnętrzny proces wytwórczy oraz sprzężenie zwrotne w społeczności....47, 48
WikiMedia....107, 108
X Window System....18, 19
X.Org....75
X11....75
Xerces....57
Yahoo!....120
Zimbra....120
życzliwi dyktatorzy....47, 51, 52, 66

Notatki

Notatki

Notatki