Sebastian D. Kotuła
Maria Curie-Sklodowska University in Lublin, Poland
sdkotula@poczta.umcs.lublin.pl

Rules for implementing open source solutions

Abstract. For more than thirty years, there has been free software on the market, and for about twenty years there also have been open source software, and even also under a common name free and open source software. Terminological diversity, ignorance of the definition of the said categories of software, common ideas associated with them, as well as poor understanding of the free and/or open source software on the market may negatively affect the consumer's decisions regarding the implementation of these IT solutions in their own businesses.

Therefore, it was decided to indicate and discuss the selection criteria that should be followed by the user, who decided to implement open source solutions in any activity in almost every sphere where computer software is used. The focus is only on open source software, which provides greater freedom of use also in commercial activities. The indicated selection criteria were established on the basis of scientific and journalistic publications as well as personal experience while working with this type of software.

As a result, a universal set of criteria was presented, including the following: functionality, usability, realization/interface, availability/distribution method, documentation, licensing practice, source code, ease of personalization, ease of implementation, price/cost, reliability, opportunities, market penetration, community/support, pedigree, software development model, error/problem list, target, cross-platform, interoperability, modularity.

The user interested in using them can freely choose among them those, which in his opinion are the most crucial in a given consignment or can be guided by all of them, and even expanded, adding further, if necessary.

Finally, the given set of criteria was used when evaluating one of the distribution of Ubuntu, which is DigitLab (digitization laboratory).

Keywords: open source, implementation of software, the rules of choice of software, universal selection criteria.

Zasady wdrażania rozwiązań open source

Abstrakt. Od ponad trzydziestu lat na rynku funkcjonują programy wolne, a od około dwudziestu także otwarte, zaś od kilkunastu pojawiają się łącznie wolne i otwarte. Różnorodność terminologiczna w tym zakresie, nieznajomość definicji rzeczonych kategorii programów, potoczne wyobrażenia z nimi związane, jak również słabe rozeznanie w rynku oprogramowania wolnego i/lub otwartego wśród konsumentów może wpływać negatywnie na decyzje dotyczące wdrażania tych rozwiązań informatycznych we własnej działalności.

W związku z tym postanowiono wskazać i omówić kryteria wyboru, jakimi powinien kierować się użytkownik, podejmując decyzję o implementacji rozwiązań open source

w dowolnej działalności w każdej niemalże sferze, gdzie wykorzystuje się programy komputerowe. Skoncentrowano się jedynie na oprogramowaniu open source, które zapewnia większą swobodę wykorzystania także w działalności komercyjnej. Wskazane kryteria wyboru ustalono na podstawie publikacji naukowych i publicystycznych, jak również własnego doświadczenia podczas pracy z tym rodzajem oprogramowania.

W rezultacie zaprezentowano uniwersalny zestaw kryteriów, w którym znalazły się następujące: funkcjonalność, użyteczność, wykonanie/interfejs, dostępność/sposób dystrybucji, dokumentacja, praktyka licencyjna, kod źródłowy, łatwość personalizacji, łatwość implementacji, cena/koszt, niezawodność, możliwości, rozeznanie rynkowe (ang. *market penetration*), społeczność/wsparcie, rodowód, model rozwoju oprogramowania, lista błędów/problemów, target, wieloplatformowość, interoperacyjność, modularność.

Użytkownik zainteresowany ich wykorzystaniem może dowolnie wybierać spośród nich te, które jego zdaniem w danej sytuacji są najbardziej kluczowe lub też może kierować się nimi wszystkimi, a nawet rozbudowywać, dodając kolejne, jeśli zajdzie taka potrzeba.

Na koniec opisany zestaw kryteriów zastosowano przy ocenie dystrybucji Ubuntu, jaką jest DigitLab, czyli laboratorium digitalizacyjne.

Słowa kluczowe: open source, wdrażanie programowania, zasady wyboru oprogramowania, uniwersalne kryteria wyboru.

Wstęp

Minęło już ponad trzydzieści lat, odkąd Richard Matthew Stallman założył Fundację Wolnego Oprogramowania (ang. *Free Software Foundation*), rozpoczął pracę nad wolnym systemem operacyjnym GNU oraz wprowadził do publicznego dyskursu termin *wolne oprogramowanie* (ang. *free software*, skrót FS). W jego rozumieniu jest to taki typ programów, który charakteryzuje się czterema wolnościami, a mianowicie wolnością do: uruchamiania programu w dowolnym celu; analizowania, jak program działa i dostosowywania go do swoich potrzeb; rozpowszechniania kopii oraz udoskonalania programu i publicznego rozpowszechniania własnych ulepszeń (Kotuła 2014, s. 24).

Za inicjatywą Stallmana poszli kolejni informatycy. W konsekwencji w drugiej połowie lat 90. XX wieku kilku z nich, jak Eric Steven Raymond, Tim O'Reilly, Bruce Perens, ukuło termin *otwarte oprogramowanie* (ang. *open source*, skrót OS i/lub OSS) oraz utworzyło niekomercyjną organizację Open Source Initiative. Zasadniczym celem jej działania stało się nadzorowanie rynku programów otwartych, a więc zarządzanie definicją *open source* (ang. *open source definition*) oraz stosownymi licencjami warunkującymi funkcjonowanie tych programów. W rzeczonej definicji znajduje się dziesięć warunków, które każdy program komputerowy musi spełnić, aby być programem open source, jak również warunki te muszą znaleźć się w wytycznych licencyjnych każdej licencji open source. Wśród tych dziesięciu punktów są następujące: każdy program open source jest swobodnie redystrybuowany (może być sprzedawany lub rozdawany); jest dostarczany z kodem źródłowym, który może być modyfikowany, a w konsekwencji na jego podstawie mogą być tworzone programy pochodne (nowe wersje, modyfikacje, ulepszenia itp.); może stanowić integralną i niezależną autorsko całość, wyraźnie odróżnioną od kolejnych wersji, modyfikacji i wprowadzonych poprawek;

może być swobodnie wykorzystywany przez kogokolwiek i w jakimkolwiek celu; nie może być relicencjonowany; oraz wszystkie jego komponenty są również open source'owe (program jako całość oraz jako zbiór komponentów składających się na ten program jako całość, jest programem open source); można rozpowszechniać z programami komercyjnymi i *vice versa*; dostarczany jest z kodem źródłowym, który można utrwalać na różnych nośnikach (Kotuła 2014, s. 13-86).

Na rynku funkcjonują zatem niezależnie od siebie programy zarówno wolne, jak i otwarte. Sprawę dodatkowo komplikuje fakt, iż przez wielu terminy te stosowane są wymiennie i/lub łącznie w formie *wolne i otwarte oprogramowanie* (akronim WiOO lub angielski FOSS od słów *Free and Open Source Software*, także FLOSS od słów *Free/Libre and Open Source Software*). Poza tym znaczna część programów faktycznie stanowi łącznie zarówno wolne, jak i otwarte rozwiązania.

Tabela 1. Porównanie programów wolnych i otwartych

Cecha	Program wolny (FS)	Program otwarty (OS)	Program wolny i otwarty (FOSS)
Dostępny za darmo	**tak**	**tak/nie**	**tak**
Dostępny odpłatnie	**nie**	**tak/nie**	**nie**
Możliwość wykorzystania w projektach komercyjnych	**nie**	**tak**	**nie**
Swoboda rozpowszechniania	**tak**	**tak**	**tak**
Dostęp do kodu źródłowego	**tak**	**tak**	**tak**
Możliwość modyfikowania	**tak**	**tak**	**tak**

Źródło: Opracowanie własne.

Widać wyraźnie, że zasadnicza różnica między wskazanymi grupami programów leży w możliwości lub nie ich sprzedaży. OS można udostępniać odpłatnie i wykorzystywać w projektach komercyjnych, podczas gdy FS muszą w każdy możliwy sposób pozostać wolne, czyli także wolne od wszelkich opłat. Natomiast cechy FOSS stanowią *de facto* kopię cech FS.

Choć na rynku informatycznym programy zarówno wolne, jak i otwarte funkcjonują już znaczy czas, to jednak niska obecnie społeczna świadomość w tym zakresie powoduje, iż konsumenci zasadniczo kojarzą je jedynie z systemem operacyjnym Linux oraz pakietem biurowym Open Office (niekiedy Libre Office). Jako dowód, malejącego zainteresowania tematyką open source, mogą posłużyć dane uzyskane z serwisu Google Trends. Wynika z nich jasno, iż od 2005 roku częstość zapytań dotyczących „open source" systematycznie spada[1]. Niska świadomość wynika zapewne z nieznajomości eksplikacji/definicji terminu *open source* oraz także z braku znajomości większej liczby konkretnych przykładów tego typu programów. Z kolei, słabe rozpoznanie programów nastawia użytkowników raczej negatywnie do ewentualnego wdrażania w jakiejkolwiek działalności rozwiązań otwartych.

[1] *Google trends*, 2018, https://trends.google.pl/trends/explore?date=all&q=%22open%20source%22.

Cel pracy

Celem niniejszego artykułu jest wskazanie i omówienie kryteriów wyboru, jakimi powinno się kierować, podejmując decyzję o implementacji rozwiązań open source w dowolnej działalności. Przy czym skoncentrowano się jedynie na oprogramowaniu.

Zdecydowano skupić się tylko na OS z powodu zapewniania większej swobody twórcom programu, niż ma to miejsce w przypadku programów wolnych. Te ostatnie muszą być rozpowszechniane bez opłat, podczas gdy open source'owe mogą stać się częścią przedsięwzięć komercyjnych, co zależy jedynie od twórcy programu.

Zaprezentowany zestaw kryteriów ma być w założeniu na tyle uniwersalny, aby pozwolić na dopasowanie go do konkretnej działalności (komercyjnej, niekomercyjnej) na każdym niemalże polu, na którym wykorzystuje się programy komputerowe. Wskazane kryteria wyboru ustalono na podstawie publikacji naukowych i publicystycznych, jak również własnego doświadczenia podczas pracy z tym rodzajem oprogramowania.

Kryteria wyboru

Każdy w zasadzie program komputerowy OS, FS czy własnościowy można wybierać, kierując się czteroetapowym procesem. W nowszych pracach określa się go mianem metody QSOS (ang. *Qualification and Selection of Opensource Software*)[2], którą można deskrybować jako: **zdefiniowanie** potrzeb, **ocenę** programów według określonych kryteriów, **kwalifikację** programów na podstawie powyższych kryteriów, ostateczny **wybór** konkretnego rozwiązania (Semeteys 2008).

Kluczowym jest precyzyjne określenie do czego program ma służyć, a tym samym, jakie funkcje ma spełniać? Na tym etapie należy zatem ustalić kryteria, którymi będzie kierował się użytkownik podczas wybierania oprogramowania. Konsekwencją zastosowania określonych kryteriów będzie wskazanie programu i/lub grupy programów, które należy zestawić i porównać ze sobą w celu dokonania wyboru konkretnego rozwiązania. Użytkownicy o mniej sprecyzowanych na wstępie potrzebach, mogą zacząć od przeglądu rzeczonych kryteriów w celu wyboru tych, które odpowiadają ich oczekiwaniom.

Zastosowane kryteria będą oczywiście zależały od zdefiniowanych potrzeb. Katalog dostępnych kryteriów jest szeroki, uniwersalny i w zasadzie nieograniczony. Można go dowolnie modyfikować, np. ograniczać, wybierając kryteria najbardziej pożądane, ale także istnieje możliwość dodania nowych wytycznych dla zindywidualizowanych potrzeb, choć wykorzystanie zaprezentowanych poniżej zdaje się wyczerpywać większość możliwości.

KRYTERIUM	WYJAŚNIENIE
Funkcjonalność	Czy funkcje, jakie program udostępnia odpowiadają tym, których poszukuje użytkownik (*Choosing…* 2011)? Funkcje powinny być wyliczone na stronie domowej projektu (Cruz, Wieland, Ziegler 2006, s. 115). Wśród funkcji wylicza się także techniczne aspekty, czyli stabilność, bezpieczeństwo, łatwość odzyskiwania danych (Benlian 2011, s. 549), ale także ochronę danych użytkownika.

[2] *QSOS*, 2018, http://www.qsos.org/Method.html.

Użyteczność	Łatwość i prostota użytkowania (dokumentacji, projektu) (Benlian 2011, s. 549), ale także intuicyjność.
Wykonanie/interfejs	Czy program jest trudny/ciężki w pracy (Surman 2004). Czy interfejs ułatwia pracę z narzędziem? Czy program nie wpływa ujemnie na pracę innych programów/systemu oraz czy nie spowalnia pracy całego urządzenia?
Dostępność/sposób dystrybucji	Czy program posiada własną stronę WWW, czy jest dostarczany z określonej bazy, repozytorium, kuźni itp. (*Choosing…* 2011)? Strona domowa ułatwia prezentację narzędzia, sprzyja budowaniu pozytywnego wizerunku programu, jak również zespołu, który go tworzy oraz świadczy o przykładaniu większej wagi dla projektu. Poza tym, każdy kanał dystrybucji zapewnia wgląd w inne statystyki, np. pobrań, instalacji, itp. (Balter 2014).
Dokumentacja	Jakość dokumentacji (Ahmad, Lapante 2015, s. 1103), a zatem czy jest to plik tekstowy z podstawowymi informacjami, czy też rozbudowana instrukcja obsługi z wszelkimi ważkimi informacjami, co oznaczałoby, że twórcom zależy na prawidłowym rozpowszechnianiu swojego produktu (Balter 2014)? Czy dokumentacja nie jest zbyt skomplikowana, co może utrudnić posiłkowanie się nią?
Praktyka licencyjna	Czy program jest poprawnie licencjonowany i jaka jest dokładnie treść licencji (Balter 2014)?
Kod źródłowy	Czy można i ewentualnie w jakim zakresie wykorzystywać kod danego programu (Niiranen 2011)? Ponadto, jak wygląda kod? Dostępny kod źródłowy powinien był przejrzysty, dobrze zaprezentowany i przemyślany, bowiem wtedy można mieć pewność, że za jego wytworzeniem stoi profesjonalny programista(ści). Tylko taki kod może oznaczać, że służył on skutecznie określonej grupie osób, które na nim polegały. Ponadto, prawidłowo opracowany i przygotowany kod będzie mógł być dalej wykorzystywany przez kolejnych, czyli będzie go można łatwo ulepszać, poprawiać i modyfikować (Wskazane kryterium w oparciu o Reichert 2017).
Łatwość personalizacji	Łatwość dostosowywania do własnych potrzeb (ang. *ease of customization*), a więc aktualizowania, rozszerzania, adaptowania (Benlian 2011, s. 549).
Łatwość implementacji	Czy czas jaki upłynie, aby uzyskać pożądany rezultat, jest z punktu widzenia biznesowego opłacalny, czy też oczekiwany rezultat będzie możliwy do uzyskania dopiero po długim czasie (ang. *time to value*) (Benlian 2011, s. 549)?
Cena/koszt	Jak bardzo czasochłonna i kosztochłonna jest implementacja narzędzia? Jaki będzie koszt utrzymania? Czy wdrożone narzędzie będzie łatwe w utrzymaniu, tzn. czy potrzebne będzie stałe

	nadzorowanie jego działania i w konsekwencji wprowadzanie tzw. *patchy* (pol. *łatek*) oraz czy dostępne są powszechnie odpowiednie narzędzia informatyczne, ułatwiające wskazane zadania (Tuma 2005, s. 8)?
Niezawodność	Czy program należycie wykonuje zadania, do których został stworzony (Wheeler 2011)?
Możliwości	Potwierdzenie, że program jest w stanie skutecznie wykonywać zadania, do których został stworzony (Wheeler 2011).
Rozeznanie rynkowe (ang. *market penetration*)	Czy dany produkt jest wystarczająco popularny w specyficznym obszarze, na potrzeby którego został opracowany. Inaczej rzecz ujmując, czy inne, podobne podmioty, a więc skupione na tych samych działaniach, co podmiot planujący wdrożenie danego rozwiązania, również z tego rozwiązania korzystają (Ahmad, Lapante 2015, s. 1103)? Jak wiele podmiotów używa programu i kim oni są (Balter 2014)? Jaka jest liczba pobrań i instalacji programu? Czy produkt został zaimplementowany w projektach, które się udały (ang. *infectiousness of prior adopters*) (Na podstawie artykułu Singh, Phelps 2010, s. 16)? Jaką program ma reputację (Metcalfe 2004)?
Społeczność/wsparcie	Czy program posiada wsparcie społeczności, w tym także komercyjne (Metcalfe 2004)? Wsparcie może być techniczne, stałe 24 godziny na dobę, czy też w formie szkoleń (Benlian 2011, s. 549).
Rodowód (Ahmad, Lapante 2015, s. 1103)	Kim jest twórca(y) projektu oraz jakie ma(ją) doświadczenie w tworzeniu podobnych projektów (Balter 2014)? Czy podmiot odpowiedzialny za dany projekt pozostaje pod jakimikolwiek wpływami, które mogą skutkować dalszymi decyzjami w zakresie rozwoju lub nie danego projektu (Cruz, Wieland, Ziegler 2006, s. 114-115)? Czy podmiot, który wprowadził na rynek dany program OS posiada dobrze udokumentowaną historię rozwoju i utrzymywania tego typu projektów? Czy wspiera rozpoczęte przez siebie projekty OS w ramach własnego serwera? Czy też, rezygnując z dalszych prac nad projektami, umieszcza je w przestrzeni ogólnie dostępnych baz (Reichert 2017)?
Model rozwoju programu	Każdy program powinien mieć jasno przedstawiony sposób rozwoju projektu w przyszłości, a zatem czy twórcy wskazują, jak *in spe* program będzie rozwijany, kiedy pojawią się nowe wydania, uaktualnienia, poprawki itp. (*Choosing…* 2011). Ponadto, czy program rozwijał się równomiernie, np. poprawki były wprowadzane regularnie (częstotliwość uaktualnień), czy też miał okresy częstszych aktualizacji oraz okresy przestoju (ang. *project velocity)* (Ahmad, Lapante 2015, s. 1103)? Kiedy została wydana ostatnia stabilna wersja programu (Metcalfe 2004)?

Lista błędów/problemów	Aktualna lista błędów programu jest opracowywana wraz z rozwojem programu. Błędy pozostałe do naprawienia powinny być jasno wskazane, jak również powinny być wskazane te błędy, które zostały już usunięte (Cruz, Wieland, Ziegler 2006, s. 115-116). Program, który taką listę zawiera jest dobry, bowiem oznacza to, iż w pracę nad nim zaangażowanych jest wiele osób, które pracują nad likwidacją rzeczonych problemów, wyłapują nowe, projekty bez takich list wcale nie oznaczają, że program pozbawiony jest wad, ile raczej, że niewiele osób nim się zajmuje (Balter 2014). Lista problemów jest jednym z gwarantów stałego rozwoju programu (Metcalfe 2004).
Target	Dla kogo przeznaczony jest program (Niiranen 2011)? Czy użytkownicy, którzy będą mieli pracować z danym oprogramowaniem mają wystarczające doświadczenie w podobnych przedsięwzięciach (kryterium na podstawie Turnbull 2017)? Jeśli zaś ich umiejętności nie są wystarczające, to, czy istnieje możliwość odbycia odpowiedniego szkolenia, choćby w formie online, podczas którego nabyliby odpowiednich kompetencji? Jeżeli zaś praca z programem wymaga zaawansowanych umiejętności, których nabycie nie będzie możliwe w krótkim czasie dla wdrażającego, jak również dla jego pracowników, to należy sprawdzić, czy na rynku dostępne są podmioty, które będą w stanie przyjąć zlecenie realizacji wdrożenia danego oprogramowania w zadowalającej cenie (informacje na podstawie Tuma 2005, s. 8.)?
Wieloplatformowość	Program powinien być przeznaczony do pracy pod wszystkie aktualnie działające systemy operacyjne, a przynajmniej pod te, pod którymi ma być używany (Cruz, Wieland, Ziegler 2006, s. 111).
Interoperacyjność	Oprogramowanie powinno umożliwiać współpracę z innymi programami, m.in. poprzez stosowanie otwartych standardów (Metcalfe 2004). Czy dane oprogramowanie wspiera popularne standardy, np. otwarte, czy też wymusza pracę w nowym, wyjątkowym standardzie zamkniętym (pol. *blokada dostawcy*, ang. *vendor lock-in*)? Narzędzia spod znaku open source zasadniczo wspierają otwarte i/lub powszechnie używane standardy, jednakże może się zdarzyć, iż dany produkt informatyczny open source będzie wikłał użytkownika w pracę w środowisku o ograniczonej otwartości. Najprostszym przykładem jest format pliku. Dany program OS może pracować na plikach o indywidualnym rozszerzeniu i/lub też takich, które potrafi otworzyć jeszcze dany program własnościowy (informacje w oparciu Tuma 2005, s. 8). Najbardziej pożądane jest, aby nie były to nowe formaty plików, lecz te, które są w powszechnym obiegu.

Modularność	Program powinien posiadać budowę modułową, czyli dać się podzielić na mniejsze, odrębne fragmenty, z których każdy pełni określone funkcje, co w konsekwencji ułatwia ewentualne modyfikacje (Vand den Berg 2005, s. 17).

Powyższy repertuar kryteriów jest uniwersalny. Nie wszystkie muszą zawsze być wzięte pod rozwagę. Użytkownik może zaczerpnąć tylko te, które będą kluczowe w konkretnej konsytuacji.

Na koniec, po zastosowaniu ustalonych kryteriów, należy wybrać na ich podstawie kilka programów, spełniających wskazane warunki i porównać je ze sobą. Zestawienie narzędzi spełniających wymagania użytkownika pomoże w podjęciu ostatecznej decyzji, co do wykorzystania któregoś z nich.

Użycie kryteriów na przykładzie DigitLab

W niniejszej części artykułu zaprezentowane zostanie studium przypadku – jak wskazane kryteria można zastosować do systemu operacyjnego DigitLab (laboratorium digitalizacyjnego) opartego o dystrybucję Linuxa, jaką jest Ubuntu w wersji 12.04.

Funkcjonalność	System operacyjny Ubuntu 12.04 wzbogacony o darmowe i ogólnodostępne narzędzia i oprogramowanie używane w procesie digitalizacji zbiorów, np. bibliotecznych, archiwalnych itp. Jednocześnie istnieje strona przeznaczona dla Ubuntu (https://www.ubuntu.com/, https://ubuntu.pl/), jak również strona przeznaczona *stricte* dla projektu DigitLab, gdzie opisane zostały funkcje i przeznaczenie, jak również zadania, które może wykonywać (http://digitlab.psnc.pl/). Przydatne informacje zawarto również na kolejnej stronie (http://dl.psnc.pl/activities/projekty/access-it-plus/digitlab/) oraz w krótkim przewodniku (https://jbc.bj.uj.edu.pl/dlibra/publication/229576/edition/218077/content?ref=desc).
Użyteczność	System jest prosty i przyjazny w obsłudze, choć wymaga przyzwyczajenia się do pracy w środowisku Ubuntu.
Wykonanie/ interfejs	Jest prosty w pracy i nie spowalnia działania komputera. Ponadto można go uruchamiać na starszych urządzeniach. Interfejs jest przejrzysty.
Dostępność/ sposób dystrybucji	Projekt posiada własną stronę http://digitlab.psnc.pl/.
Dokumentacja	Zasadniczo dotyczy systemu Ubuntu, jak i narzędzi i programów, w które Ubuntu zostało wyposażone. Materiały dotyczące DigitLab umieszczono na stronie: https://confluence.man.poznan.pl/community/display/DIG/DigitLab+Wiki.
Praktyka licencyjna	System operacyjny oraz dodatkowe programy znajdują się na różnych licencjach open source'owych.

Kod źródłowy	Dostępny jest kod źródłowy systemu Ubuntu oraz dodatkowych programów i narzędzi wzbogacających Ubuntu.
Łatwość personalizacji	Jak każdy system Linux można go dowolnie rozbudowywać, instalując kolejne rozszerzenia, dodatki itp., natomiast zmiany wprowadzane w kodzie źródłowym wymagają znajomości programowania, a więc są zależne od umiejętności użytkownika.
Łatwość implementacji	Wdrożenie jest natychmiastowe (podobnie, jak Ubuntu i DigitLab można uruchomić bez instalacji). Dla podmiotu, rozpoczynającego działania w zakresie digitalizowania zbiorów i posiadającego niewielkie środki, produkt jest opłacalny z punktu widzenia biznesowego.
Cena/koszt	Omawiana dystrybucja jest darmowa.
Niezawodność	Ubuntu jest sprawnie działającym systemem operacyjnym, zaś poprawność działania narzędzi, o które została wzbogacona dystrybucja DigitLab, zależeć będzie od preferencji użytkownika w konkretnych, zdefiniowanych obszarach problemowych.
Możliwości	DigitLab należycie wykonuje zadania, do których został zaprojektowany.
Rozeznanie rynkowe	Nie jest możliwe do oszacowania wykorzystanie DigitLab. System został nagrodzony w konkursie akademickim na projekt open source 2013, zajmując trzecie miejsce. Por. A. Musialska, *DigitLab nagrodzony!* http://www.man.poznan.pl/online/pl/artykuly/2098/DigitLab_nagrodzony.html.
Społeczność/ wsparcie	Zarówno Ubuntu, DigitLab, jak i dodatkowe narzędzia posiadają wsparcie twórców i społeczności.
Rodowód	Twórcy DigitLab mają duże doświadczenie w realizacji projektów przeznaczonych na potrzeby środowiska bibliotecznego.
Model rozwoju programu	Dotyczy zasadniczo Ubuntu. Brak informacji na temat rozwoju *stricte* DigitLab. Projekt DigitLab realizowany był w latach 2012-2013 (Por. *DigitLab. Oprogramowanie przydatne w digitalizacji dla instytucji kultury,* http://digitlab.psnc.pl/, wtedy też powstała jego obecna wersja.
Lista błędów/ problemów	Dotyczy zasadniczo Ubuntu.
Target	DigitLab przeznaczony jest dla osób digitalizujących zbiory, np. biblioteczne. Nie jest skomplikowany w użyciu i nie wymaga specjalistycznej wiedzy i umiejętności.
Wielo-platformowość	Nie dotyczy, bowiem DigitLab sam jest systemem z rodziny Linux.
Interoperacyjność	Wspiera otwarte standardy.
Modularność	Jak każdy system z grupy Ubuntu, umożliwia instalowanie kolejnych programów, usuwanie niepotrzebnych itp.

Zakończenie

Oprogramowanie OS wymaga zastosowania określonej metodologii, umożliwiającej ocenę, a w konsekwencji wybór konkretnych narzędzi, które spełnią ustalone na wstępie wymagania (Semeteys 2008). Z jednej strony wynika to ze specyfiki tego obszaru IT, a z drugiej strony z ilości dostępnych narzędzi OS. Zaprezentowany w artykule zestaw kryteriów należy traktować jako uniwersalny i dostosowywać w kontekście zindywidualizowanych potrzeb. Innymi słowy, w zależności od preferencji, należy wybrać te kryteria, które będą w danej sytuacji najbardziej pożądane.

Wskazane kryteria wyboru warto zawsze odnosić i porównywać z rozwiązaniami własnościowymi i zamkniętymi. Dla przykładu, może się bowiem okazać, iż o wiele bardziej przydatny program open source nie zapewnia jednak odpowiedniej gwarancji, na której może zależeć użytkownikowi i dlatego lepszym wyborem będzie ostatecznie produkt własnościowy, który wymaganą gwarancję niezawodności oferuje (Tuma 2005, s. 8).

Bibliografia

1. Ahmad N., Lapante P.A., 2015, *Chapter 53. A systematic approach to evaluation open source software*, [w:] *Open source technology. Concepts, methodologies, tools and applications*, Hershey, s. 1091-1109.
2. Balter B., 2014, *How to identify a stron open source project*, https://ben.balter.com/2014/06/02/how-to-identify-a-strong-open-source-project/ [wszystkie adresy WWW użyte w niniejszym artykule były aktywne w marcu 2018].
3. Benlian A., 2011, *Is traditional, open-source, or on-demand first choice? Developing an AHP-based framework for the comparison of different software models in office suites selection*, „European Journal of Information Systems", vol. 20, iss. 5, s. 542-559.
4. *Choosing the right open-source software for your project*, 2011, https://www.software.ac.uk/choosing-right-open-source-software-your-project.
5. Cruz D., Wieland T., Ziegler A., 2006, *Evaluation criteria for free/open source software products based on project analysis*, „Software Process Improvement and Practice", vol. 11, iss. 2, s. 107-122.
6. Kotuła S.D., 2014, *Wstęp do open source*, Warszawa.
7. Metcalfe R., 2004, *Top tips for selecting open source software*, http://oss-watch.ac.uk/resources/tips.
8. Niiranen O., 2011, *What are the most important criteria to select open source license?*, https://www.quora.com/What-are-the-most-important-criteria-to-select-an-open-source-license.
9. Reichert J., 2017, *Open source. To use or not to use (and how to choose)*, „Forbes", July 20, https://www.forbes.com/sites/forbestechcouncil/2017/07/20/open-source-to-use-or-not-to-use-and-how-to-choose/3/.
10. Semeteys R., 2008, *Method for qualification and selection open source software*, http://www.timreview.ca/article/146.
11. Singh P.V., Phelps C., 2010, *Determinants of open source software license choice. A social influence perspective*, s. 16, http://repository.cmu.edu/cgi/viewcontent.cgi?article=2479&context=tepper.

12. Surman M., 2004, *Choosing open source. A decision – making guide for civil socjety organizations*, https://marksurman.commons.ca/publications/choosing-open-source-a-decision-making-guide-for-civil-society-organizations/full-text/.
13. Tuma D., 2005, *Open Source Software. Opportunities and challenges*, „The Journal of Defense Software Engineering", January, s. 6-10, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.431.3202&rep=rep1&type=pdf.
14. Turnbull D., 2017, *When to choose open source search (and when you shouldn't*), http://opensourceconnections.com/blog/2017/04/29/choosing-proprietary-vs-open-source-search/.
15. Van den Berg K., 2005, *Finding open options. An open source software evaluation model with a case study on Curse Management Systems*, Tilburg, s. 17, http://www.karinvandenberg.nl/Thesis.pdf.
16. Wheeler D.A., 2011, *How to evaluate open source software/free software (OSS/FS) programs*, https://www.dwheeler.com/oss_fs_eval.html#wrapup.