FACULTY OF LAW

MUNI Law Working Paper Series
David Kosař & Radim Polčák, Co-Editors in Chief

MUNI Law Working Paper 2015.05

Martin Loučka

SPECIFIKA VÝVOJE POČÍTAČOVÝCH PROGRAMŮ A JEJICH AUTORSKOPRÁVNÍ KONSEKVENCE

November 2015/ Listopad 2015

Masaryk university * Law Faculty * Brno * Czech Republic
MUNI Law Working Paper Series can be found at:
http://workingpapers.law.muni.cz/content/en/issues/

MUNI Law Working Paper Series
ISSN 2336-4947 (print)
ISSN 2336-4785 (online)
Copy Editor: Martin Bobák

Masarykova univerzita,
Právnická fakulta
Veveří 70, Brno 611 70
Česká republika

Abstract

This paper deals with copyright aspects of computer programs development and addresses the issue of distribution consequences of various development strategies and options. The purpose of the paper is to analyze main „easements", so to say, that are used by programmers and may have impact on licensing and distribution options of the final computer program. Recognized and studied are namely IDEs (integrated development environment), function libraries and compilers, as these are considered to be the key phenomenon of software development. The paper provides clear outputs related to each one of aforementioned, and uses consistent, code-oriented approach when making arguments as this fits best in the context of contemporary paradigm of copyright protection of the computer programs.

Keywords
Copyright, Computer Programs, Software Development, Licensing, IDE, Software Library, Compiler

Suggested Citation:
Loučka, Martin 'Specifics of Computer Programs Development and Their Consequences in Copyright' (2015), MUNI Law Working Paper No. 2015.05, available at: http://www.law.muni.cz/dokumenty/32596.

Abstrakt

Tento příspěvek se zabývá autorskoprávními aspekty vývoje počítačových programů a řeší otázky rozličných přístupů a možností při programování a jejich dopad do roviny distribuce. Účelem příspěvku je analyzovat hlavní metody zjednodušení práce používané programátory, které mohou ovlivnit možnosti při následném licencování a distribuci výsledného programu. Autor rozeznává a provádí studii IDE (integrované vývojové prostředí), softwarových knihoven a kompilátorů, jakožto klíčových fenoménů spojených s vývojem software. Příspěvek podává dílčí závěry ke každému z vyjmenovaných, přičemž jako jednotící aspekt byl zvolen přístup k právní argumentaci orientovaný na kód, což rezonuje se současným paradigmatem autorskoprávní ochrany počítačových programů.

Klíčová slova
Autorské právo, počítačové programy, vývoj software, licencování, IDE, softwarová knihovna, kompilátor

Doporučená citace:
Loučka, Martin 'Specifika vývoje počítačových programů a jejich autorskoprávní konsekvence' (2015), MUNI Law Working Paper No. 2015.05, dostupné z: http://www.law.muni.cz/dokumenty/32596.

Author Contact Details

Martin Loučka
Law Faculty, Masaryk University
Brno, Czech Republic

Email: martin.loucka@mail.muni.cz

Kontakt na autora

Martin Loučka
Právnická fakulta Masarykovy univerzity
Brno, Česká republika

Email: martin.loucka@mail.muni.cz

SPECIFIKA VÝVOJE POČÍTAČOVÝCH PROGRAMŮ A JEJICH AUTORSKOPRÁVNÍ KONSEKVENCE

Martin Loučka

Úvodem

Počítačové programy jsou bezpochyby fenoménem moderní doby. Právě jejich prostřednictvím je možné na uživatelské úrovni ovládat stroje, které podle nich pracují. Není zaměřením této práce popisovat historii vývoje počítačových programů, jakkoli je zajímavá, ani kontemplovat nad jejich podstatou a mezemi jejich ochrany. Pro účely této práce budeme tedy počítačovým programem nazývat kód, kterému svědčí ochrana jako počítačovému programu ve smyslu autorského zákona[1], respektive EUSD[2]. Bude se tedy primárně jednat o kód určený stroji k tomu, aby jej vykonal. Legální definici nám evropská úprava, ani český autorský zákon neposkytují, což sice znamená zvýšenou míru právní nejistoty v otázce šířky spektra kódu požívající autorskoprávní ochrany, nicméně v konečném důsledku to z pohledu autora může přispět k tomu, že rozsah předmětů ochrany bude lépe reflektovat technickou realitu. Pro podrobné vymezení pojmu počítačového programu lze plně odkázat na jiné práce.[3] Snad až na drobné výjimky se budu tedy snažit tomuto jinak nesmírně zajímavému tématu vyhnout.

Aby kód mohl požívat autorskoprávní ochrany jako počítačový program, musí být splněny podmínky ochrany a kvalifikace pod předmět ochrany dle autorského zákona (takovým předmětem ochrany bude v případě počítačového programu autorské dílo[4]). To mimo jiné

[1] Zákon č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění účinném ke dni 16. března 2015.
[2] Směrnice Evropského parlamentu a Rady 2009/24/ES ze dne 23. dubna 2009, o právní ochraně počítačových programů.
[3] Například ŠAVELKA, Jaromír. *Autorskoprávní ochrana funkcionality softwaru* [online]. 2013 [cit. 2015-03-15]. Rigorózní práce. Masarykova univerzita, Právnická fakulta. Vedoucí práce Filip Křepelka. Dostupné z: <http://is.muni.cz/th/134449/pravf_r/>.
[4] Autorské dílo není jediným předmětem ochrany dle autorského zákona, dalšími mohou být například umělecké výkony nebo netvůrčí databáze.

znamená, že musí být *původní v tom smyslu, že je autorovým vlastním duševním výtvorem*[5]. Z dikce zákona vyplývá bezpodmínečná nutnost přítomnosti lidského elementu v procesu tvorby počítačového programu, neboť jinak by dílo nemohlo být autorovým *vlastním*[6] *duševním*[7] výtvorem. Autorem může být pouze fyzická osoba, resp. terminologií nového občanského zákoníku – člověk.

Tato práce se bude zabývat zejména otázkami přítomnosti naposledy zmiňovaného elementu v souvislosti s programy generovanými prostřednictvím jiných programů, a dále otázkami souvisejícími s používáním rozličných předpřipravených řešení a systémů pro usnadnění práce programátorů.

1 Jak vznikají počítačové programy

Stranou nyní ponechme problematiku motivace k započetí vývoje počítačového programu. Někteří programátoři tak činí ze soukromých pohnutků. Jiní očekávají za svou činnost nějaké protiplnění. Hlavní hybnou složkou však bezpochyby bude v naprosté většině ekonomický záměr, pod který můžeme podřadit jak „standardní" komerčně vznikající software, tak i model F/OSS – tedy Free and Open Source Software, který je sice distribuován zdarma, ale přináší vývojářům alespoň sekundární ekonomický prospěch ve formě nadstandardních služeb či alespoň získání velice kvalitního marketingového nástroje. Není náhodou, že Google či Facebook, jedny z nejúspěšnějších společností současnosti, poskytuje velké množství služeb nikoli výměnou za peníze a vynakládají nemalé investice právě do Open Source Software[8].

V této práci se budeme věnovat převážně jiné fázi vytváření počítačového programu, a to samotnému programování. V této činnosti totiž spočívá mnoho problematických aspektů z hlediska dopadu na následnou autorskoprávní kvalifikaci celého programu[9] a zásadně je tak ovlivněno celé právní postavení výsledného produktu. Takovýto dopad je dán zejména tím, že programátoři, ať již pracující samostatně či ve společnosti, směřují k maximální efektivitě svého snažení. Mnohé části kódu, které jsou známé tím, že se neustále opakují, tak nepíší vždy znovu, ale využívají předpřipravených řešení. Stejně tak využívají *programovacích jazyků*, jejich nadstaveb (tzv. *Framework*), různých *šablon kódu*, a v neposlední řadě ne vždy píší přímo

[5] § 2, odst. 2 autorského zákona.
[6] Tedy nikoli zprostředkovaným.
[7] Tedy vzniklý procesem myšlení.
[8] Google dokonce vede specializovaný Open Source Blog (k dispozici na adrese http://google-opensource.blogspot.cz/), a nelze nezmínit jeho open source projekty typu Android, Angular, Chromium či TensorFlow.
[9] Například zda se jedná o původní či odvozené dílo, popřípadě zda na produkt dopadá ochrana jako na počítačový program, nebo například i jako na databázi.

spustitelný kód[10] – v mnoha případech je naopak nutné napsaný zdrojový kód učinit spustitelným až následným procesem kompilace, při kterém jiný program, zvaný *kompilátor*, samostatně provede některé operace se vstupním kódem a na jehož výstupu je teprve spustitelný počítačový program. To vše se děje za pomoci vývojového prostředí pro programátory, tzv. *IDE*[11].

Všechny tyto prvky mívají jedno společné – jsou autorskými díly někoho jiného, než je autor finálního programu. Je tedy na místě věnovat jim podrobnější zkoumání, abychom zjistili, jak programátory – kromě toho, že jim extrémně usnadňují práci – limitují.

V dalších kapitolách se pokusíme analyzovat právní aspekty jednotlivých prvků, které vstupují do hry při vývoji počítačového programu, a které jsou vyjmenovány výše, a budeme zkoumat jejich vazbu na vznikající program jako takový.

2 Programovací jazyk a Framework

V souvislosti s vývojem počítačových programů je na prvním místě vhodné uvést pojem programovacího jazyka[12]. Jeho prostřednictvím totiž může programátor předávat příkazy stroji (zpravidla počítači), který je poté provádí. V krátkosti si tento pojem přibližme.

Programovací jazyk je ve své podstatě abstraktní konstrukt, který definuje jakýsi jednosměrný komunikační protokol mezi člověkem a strojem a napomáhá vyjádření programátorových idejí tak, aby byly jednoznačné a strojem proveditelné. Stěžejním pojmem této práce bude *syntax*.

Syntax je soubor definic, který nám umožňuje tuto jednoznačnost výrazů zachovat. Určuje, jakým stylem kódovat námi předávanou informaci a přiřazuje jednotlivým výrazům jejich obsah. Je důležité poznamenat, že programovací jazyk ovšem nemůže být efektivním způsobem přenosu *sdělení*, neboť je určen k přenosu *příkazů*. Není tedy možné jeho prostřednictvím kódovat jiné informace než, ty které následně vyústí v soubor příkazů předneseným procesoru adresáta-stroje. Jedinou výjimkou snad může být objektově orientované programování; rozbor tohoto ovšem není předmětem práce.

V každém případě zásadní je však poznatek, že syntax a tedy jazyk jako celek je pouze „prázdnou schránkou“, pouze reprezentací *metody*, pomocí které je možné převádět myšlenky

[10] Na tomto místě máme přímo spustitelným kódem na mysli programy psané v interpretovaných jazycích typu JavaScript, které neprochází procesem kompilace a šíří se a pracují v podobě lidsky čitelného zápisu (samozřejmě s výhradou vizuálního znepřehlednění kódu metodou „obfuskace“).

[11] Integrated Development Environment. Jako příklad lze uvést rozšířená prostředí Netbeans (https://netbeans.org/) či Eclipse (http://eclipse.org/). Sluší se poznamenat, že samotné IDE nemusí nutně vykonávat všechny zmíněné činnosti (např. kompilaci), ale je vývojáři při vývoji využíváno jako jednotící prvek s přidanou hodnotou, která bude rozebrána dále v této práci.

[12] Úvod do problematiky programovacích jazyků, jejich třídění, základní vlastnosti a paradigmata programování např. AABY, Anthony A. *Introduction to Programming Languages* [online]. 2004 [citováno 2015-03-16]. Dostupné z: http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/intro.htm.

do podoby smysluplného a zpracovatelného zdrojového kódu. Programovací jazyk jako metoda zápisu je vyloučen z ochrany na základě ustanovení § 2, odst. 6 autorského zákona. S ohledem na znění EUSD je nutné poukázat na to, že ochrana dle dané směrnice se nevztahuje na „*myšlenky a zásady, na kterých je založen kterýkoli z prvků počítačového programu, včetně myšlenek a zásad, na kterých je založeno jeho rozhraní*“[13]. Zároveň je třeba konstatovat, že programovací jazyk není vyjádřením počítačového programu, neboť postrádá podstatné prvky, které takové vyjádření musí nést, neboť není sám o sobě schopný takový program rozmnožit[14].

Není tedy chráněn jako autorské dílo, a to ani obecně podle autorského zákona, ani v rámci speciální úpravy EUSD. Ke stejnému závěru dochází i Soudní dvůr Evropské unie[15]. Ke stejnému závěru je možné dojít i v případě formátu dokumentů, který je také pouze metodou ukládání informací do paměti počítače[16].

Související složkou každého programovacího jazyka je ovšem jeho dokumentace, tedy jakýsi návod k použití, který obsahuje jednak informace ohledně použitých syntaktických pravidel, jednak soupis názvů příkazů a funkcí, kterými je daný jazyk ve svém základním provedení vybaven. Tato dokumentace bude zpravidla chráněna jako literární dílo podle autorského zákona, případně jako dílo souborné či jako databáze, v závislosti na její poskytované podobě. Před programováním, popřípadě během něj si programátor využívá k tomu, aby se s jazykem naučil pracovat a aby mohl využít veškerý potenciál, který jazyk pro jeho potřeby nabízí. Nejedná se ale o užití autorského díla z pohledu autorského práva. Text dokumentace se nepromítá do výsledného programu a slouží opravdu jen jako návodná informace pro vývojáře, který poté hledá řešení problému a píše zdrojový kód programu.

Jak jsme ukázali výše, programovací jazyk není autorskoprávně chráněn. To znamená, že samotné jeho použití při vývoji počítačového programu nemůže klást žádná autorskoprávní omezení týkající se možností nakládání s výsledným programem či limitující autorovy kompetence při jeho distribuci. V tomto prvním ideálním případě, pokud by autor pouze s použitím poznámkového bloku, nebo dokonce tužky a papíru[17] napsal zdrojový kód počítačového programu v některém programovacím jazyce, dosáhl by ve vztahu k takovému kódu nejširší možné škály práv a možností z pohledu autorského práva. Nutno podotknout,

---

[13] Čl. 1, odst. 2 směrnice Evropského parlamentu a Rady 2009/24/ES ze dne 23. dubna 2009, o právní ochraně počítačových programů.

[14] Rozsudek SDEU ze dne 22. prosince 2010. Bezpečnostní softwarová asociace. Věc C-393/09, Sb. rozh. s. I-13971, bod 35.

[15] Srov. Rozsudek Soudního dvora EU. SAS Institute Inc. proti World Programming Ltd. Věc C-406/10. Dostupné online z: http://curia.europa.eu/juris/document/document.jsf?docid=122362&doclang=CS.

[16] VENÂNCIO, Pedro Dias, Ph.D. Electronic File Formats – free to use under Copyright Law. In *3rd International Research Forum Göttingen*. 2014.

[17] Pro zajímavost lze podotknout, že touto metodou je řešena většina písemných zkoušek při studiu informačních technologií na FIT ČVUT.

že takovéto případy budou spíše v rovině studia či teorie. Během vývoje počítačových programů totiž zpravidla dochází ke „stavění“ software nad již hotovým řešením.

Jedním z takových řešení může být tzv. framework. Riehle uvádí framework následujícími slovy:

> *A framework is a model of a particular domain or an important aspect thereof. A framework may model any domain, be it a technical domain like distribution or garbage collection, or an application domain like banking or insurance. A framework provides a reusable design and reusable implementations to clients.*[18]

Lépe představitelnou definici uvádí server DocForge, když popisuje framework:

> *A software framework is a set of source code or libraries which provide functionality common to a whole class of applications. While one library will usually provide one specific piece of functionality, frameworks will offer a broader range which are all often used by one type of application. Rather than rewriting commonly used logic, a programmer can leverage a framework which provides often used functionality, limiting the time required to build an application and reducing the possibility of introducing new bugs.*[19]

Framework si můžeme představit jako předpřipravené abstraktní řešení, které programátorovi usnadňuje práci. S trochou nadsázky lze říct, že se jedná o mezistupeň mezi programem a programovacím jazykem. Práce s ním se totiž v mnoha ohledech neliší od práce s programovacím jazykem – stále píšeme kód, stále pracujeme s jeho dokumentací a stále hledáme řešení k jednotlivým problémům vedoucím k sestavení výsledného programu. Je to dáno tím, že každý framework musí být na nějakém programovacím jazyce založený[20]. Jedná se o abstrakční vrstvu, která v sobě obsahuje „předvyřešené“ některé běžné problémy, popřípadě vylepšuje daný programovací jazyk tím, že poskytuje jednoduché a intuitivní řešení pro překonání některých jeho nedostatků. Hlavními tématy bývá bezpečnost, ošetření uživatelského vstupu a práce s proměnnými a některými systémovými nástroji. Framework je implementací jazyka, ve kterém (a pro který) byl vytvořen, nicméně nezřídka se práce s ním do značné míry odlišuje od běžné práce s jazykem. Na rozdíl od programovacího jazyka, který stačí „jen“ navrhnout a vydefinovat jeho tzv. abecedu výrazů a syntax, framework musí již disponovat reálným kódem, který samozřejmě musel někdo napsat. Ačkoli framework tedy není počítačový program v běžném slova smyslu, neboť „sám o sobě“ často vůbec není spustitelnou entitou,

---

[18] RIEHLE, Dirk. *Framework Design: A Role Modeling Approach* [online]. 2000 [citováno 2015-03-16]. Disertační práce. Zürich, Switzerland, ETH Zürich. Dostupné z: http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf. Strana 54. Je zajímavé, že v celé přes dvě stě stran dlouhé práci se ani jednou nevyskytuje slovo „law“ nebo „legal“.

[19] DOCFORGE. *Framework* [online]. 2013 [citováno 2015-03-16]. Dostupné z: http://docforge.com/wiki/Framework. Server DocForge je recenzovaná znalostní báze pro vývojáře software.

[20] Například lze uvést Nette Framework od Davida Grudla, jakožto velice oblíbený a propracovaný český příspěvek do světa frameworků, který je postavený na programovacím jazyce PHP. Adresa projektu: http://nette.org/cs. Jiným příkladem je robustní kompilovaný framework od Microsoftu, Framework .NET, dostupný včetně podrobností z: http://www.microsoft.com/cs-cz/download/details.aspx?id=17851.

jedná se o robustní soubor kódu, nikoli jen o definovanou metodu, standard, jak takový kód zapisovat.

Dostáváme se k problematice odvozeného díla. Podrobněji se jí budeme věnovat v kapitole 4, zde je důležité zmínit základní východiska. Počítačový program je chráněný jako dílo literární. EUSD sice stanoví v článku 1, odst. 2, že se chrání vyjádření počítačového programu v jakékoli formě, nicméně SDEU výkladem upřesnil, že se ochrana nevztahuje na vyjádření uživatelského rozhraní počítačového programu[21]. V rozsudku dále uvedl, že „*předmětem ochrany přiznané touto směrnicí je tudíž jakákoliv forma vyjádření počítačového programu, která umožňuje program rozmnožit v různých počítačových kódech, jako jsou zdrojový a strojový kód.*[22]“ Kritérium pak dále rozvedl v tom směru, že „*jakákoli forma vyjádření počítačového programu musí být chráněna od okamžiku, kdy by její rozmnožení způsobilo rozmnožení samotného počítačového programu, a umožnilo by tak počítači, aby plnil svou funkci.*“[23]

Otázkou je, jaké jiné vyjádření počítačového programu připadá v úvahu, na které nebude možné analogicky aplikovat závěr SDEU o vyloučení ochrany a které nebude kódem. Dle názoru autora takové vyjádření programu není. V praxi je tedy nutné dané ustanovení směrnice číst tak, že *„ochrana podle této směrnice se vztahuje na vyjádření* ***kódu*** *počítačového programu v jakékoli formě.“* Kódem se rozumí buď kód zdrojový, strojový, nebo objektový. Z toho plyne, že se pro účely institutu odvozeného díla bude posuzovat zejména podobnost zdrojového kódu programu. Pro samotné zkoumání následně poslouží tzv. Abstraction-Filtration-Comparison test[24]. Při použití frameworku je nepochybné, že takový test skončí pozitivní odpovědí na otázku, zda je výsledný program odvozeným dílem od daného frameworku – respektive od *kódu* daného frameworku. Tento kód bude totiž v každém případě inkorporován ve výsledném programu a bude v mnoha případech dokonce tvořit jeho většinovou část.

Tuto část tedy můžeme uzavřít v tom smyslu, že samotné užití programovacího jazyka nezakládá žádná omezení autora výsledného programu ve vztahu k možnostem distribuce a licencování. Do opačné situace se dostáváme ohledně použití programovacích frameworků, jakožto nadstaveb nad programovacími jazyky. Zde je nutné konstatovat, že autorská práva autora výsledného programu budou poznamenána omezeními vyplývajícími z faktu, že výsledný program je derivátem frameworku. V tomto směru je možné očekávat uplatnění specifických licenčních omezení, mj. se může objevovat *copyleftová doložka*, nutící autora výsledného programu

[21] Rozsudek Soudního dvora EU ze dne 22. 12. 2010. Bezpečnostní softwarová asociace v. Ministerstvo kultury. Věc C-393/09. Dostupné online z: http://curia.europa.eu/juris/document/document.jsf?docid=122362&doclang=CS.
[22] Bod 35 rozsudku.
[23] Bod 38 rozsudku.
[24] ZBRÁNEK, Lukáš. *Právní aspekty zpětné analýzy počítačových programů* [online]. 2012 [cit. 2015-03-16]. Diplomová práce. Masarykova univerzita, Právnická fakulta. Vedoucí práce Matěj Myška. Dostupné z: http://is.muni.cz/th/98818/pravf_m/, str. 10.

uvolňovat zdrojové kódy a šířit program pod specificky určenou licencí.[25] Pro upřesnění je nutné podotknout, že tento závěr o frameworku je založen na předpokladu, že daný framework bude do výsledného produktu inkorporován staticky a nebude distribuován například pouze „kód, který při správném spojení s frameworkem bude představovat implementaci nějakého programu". Naopak, pokud bude k dispozici část zdrojového kódu *určená pro daný framework* (ale samostatně tedy neschopná běhu), na tuto část kódu se licenční omezení frameworku nevztahují.

S programovacím jazykem, popřípadě nad tím postaveným frameworkem následně programátor pracuje zpravidla v integrovaném vývojovém prostředí (IDE). O tom a o konsekvencích jeho použití pojednává následující kapitola.

3 Integrované vývojové prostředí (IDE) a šablony kódu

IDE si můžeme představit jako specializovaný program, který umožňuje zefektivnit a do jisté míry zpříjemnit práci vývojářům. Počítačové programy jsou zřídkakdy tvořeny jedním souborem se zdrojovým kódem. Mnohem častěji mají mnoho součástí, které se vzájemně doplňují a navazují na sebe. Prvním problémem tedy pro programátora je udržet si přehled o všech článcích svého programu a správně s nimi pracovat. Mezi standardní funkce IDE proto většinou patří správa projektu a do něj zařazených souborů. Dále často podporuje pokročilé funkce usnadňující práci s kódem jako takovým, jako je dynamické „našeptávání", resp. dokončování rozepsaných částí kódu či poskytování *ad hoc* vhledu do dokumentace používaného programovacího jazyka. Můžeme říct, že základní IDE je ve své podstatě prázdná schránka, která jen poskytuje podpůrné funkce jejímu uživateli. Sama o sobě ovšem neprodukuje žádný kód.

Některá IDE jdou v tomto ohledu o něco dále. Například Visual Studio[26] při práci s některými objektově orientovanými jazyky[27], jako je C#, umožňuje prostřednictvím grafického rozhraní vytvořit vzhled a definovat některé parametry programu bez psaní samotného kódu vývojářem. Jednotlivé prvky programu, jako jsou tlačítka či textová pole je tak možné metodou drag & drop umístit na simulované okno budoucího programu. Takovéto funkci budeme říkat Application designer.

---

[25] Takováto situace by nastala například při použití frameworku licencovaného pod open source veřejnou licencí GNU GPL.

[26] Prostředí od společnosti Microsoft. https://www.visualstudio.com.

[27] Tedy jazyky implementující objektově orientovaný přístup k programování. Související výklad např. PECINOVSKÝ, Rudolf. *OOP – Learn Object Oriented Thinking and Programming* [online]. 2013 [citováno 16. března 2015]. Dostupné z: http://files.bruckner.cz/be2a5b2104bf393da7092a4200903cc0/PecinovskyOOP.pdf, str. 10.

Ohledně první skupiny vlastností IDE není třeba se z hlediska užívání výsledného programu příliš znepokojovat. Nehrozí totiž reálný přesah podkladového autorského díla, které by bylo součástí IDE, do výsledného díla programátora. IDE v této konfiguraci plní pouze nástroje pro usnadnění orientace a zrychlení některých operací; veškerý intelektuální vklad ovšem pochází od autora, jakožto původce výsledného počítačového programu. Pouhé strojové předvídání konců slov založené na lexikální podobnosti právě psaného výrazu s předdefinovaným slovníkem nemění nic na faktu, že programátor musel provést analýzu problému a implementovat jeho řešení. IDE mu pomáhá, jen co se týče pohodlnosti provedení jím již vymyšleného postupu. Ohledně nápovědy spočívající v zobrazování relevantní dokumentace k danému programovacímu jazyku lze uvést, že na ochranu nebo autorskoprávní kvalitu počítačového programu nemůže mít vliv taková vnější okolnost, která programátorovi poskytne informaci ohledně technického, resp. v tomto případě syntaktického aspektu jím prováděného řešení. Nabízí se ještě srovnání s institutem spoluautorství díla ve smyslu § 8 autorského zákona. *Prima facie* absurdní možnost, že by našeptávač kódu, respektive jeho autor, který do něj zavedl funkcionalitu a informace, mohl být spoluautorem díla vyvrací odstavec 2 zmiňovaného paragrafu, když stanoví, že *„[s]poluautorem není ten, kdo ke vzniku díla přispěl pouze poskytnutím pomoci nebo rady technické, administrativní nebo odborné povahy nebo poskytnutím dokumentačního nebo technického materiálu (…)“*. I kdyby na základě takovéto zobrazené informace programátor změnil svůj plán a implementoval řešení jinak, nebo dokonce i jiné, stále je to výhradně on, kdo provádí intelektuální činnost, a zejména – je to stále on, kdo vytváří a formuluje výsledný kód; bez jeho rozhodnutí, či jinak řečeno vkladu, se kód nezmění.

Mnohem komplikovanější je situace ohledně druhé nastíněné funkcionality některých IDE, tedy tzv. Application designera. Stále můžeme vycházet ze stejných předpokladů. Počítačový program je chráněn jako dílo literární. IDE vývojáři velice silně ulehčuje práci tím, že mechanicky provádí rutinní činnosti za něj. Je nutné na tomto místě osvětlit další zásadní informaci: I aplikace, která byla vytvořena prostřednictvím funkce Application designer disponuje zdrojovým kódem. Dokonce ten je to jediné, co se nakonec ukládá do paměti a spouští. Přehledný vývojový nástroj, ve kterém můžeme myší vytvářet uživatelské prostředí, je pouze iluze interpretovaná naším IDE. Application designer reálně pracuje na principu extenzivního doplňování – či v tomto případě dokonce autonomního vkládání – zdrojového kódu na odpovídající místa zdrojových souborů aplikace: v extrémním případě nemusí vývojář napsat ani jediný řádek kódu, a výsledkem jeho (nutno podotknout, že naprosto neliterární) činností je spustitelná aplikace. Zářným příkladem tohoto postupu může být například program zvaný Game Maker[28], který umožňuje i neprogramátorovi vytvořit hratelnou aplikaci s nepřeberným

[28] Program společnosti YoYo Games. https://www.yoyogames.com/studio.

množstvím variant herního scénáře. Game Maker je v tomto případě IDE *sui generis*. Základním rozdílem oproti předcházejícím úvahám je tedy to, že vývojář zde již není tím, kdo vytváří a formuluje výsledný kód; stále ovšem platí, že bez jeho rozhodnutí se kód samostatně nezmění. Zdrojový kód, který výsledný program obsahuje po jeho navržení prostřednictvím Application designera, samozřejmě nepsal stroj sám. Tento kód musel napsat jiný člověk – a to konkrétně jeden z vývojářů samotného IDE obsahujícího Application designera. Takzvaná šablona kódu, která je obsažena „uvnitř" IDE, je tedy na základě vstupu „našeho" vývojáře doplněna o jeho nastavení (například popisky tlačítek či atributy použitého písma) a vložena do zdrojových souborů výsledného programu. Stojí za povšimnutí, že v rámci tohoto procesu nejsou doplňovány jisté minoritní prvky do kódu programátora, ale je tomu právě naopak – jisté, z hlediska literárního vyjádření minoritní, prvky jsou doplňovány do předvypracované šablony. Sluší se ovšem na tomto místě dodat, že ne každý takovýto vklad IDE do výsledného kódu bude konstituovat natolik podstatné rozšíření kódu, aby byl s to ovlivnit autorskoprávní kvalifikaci výsledného programu jako odvozeného díla – v mnoha případech bude naopak činnost Application designera spadat po právní stránce stále pod režim „našeptávání" popsaný výše. Zejména tomu tak bude v případech, kdy bude ochrana vloženého *snippetu* kódu vyloučena, například na základě *merger doctrine*[29], resp. *scenes a faire*. Tak tomu bude u velké části kódu sloužícího pro umisťování komponent do grafického rozhraní aplikace nebo při vkládání obecných jazykových konstrukcí[30].

V případě, kdy se ovšem takové vyloučení neuplatní, lze tedy uzavřít, že autorem výsledného počítačového programu je sice „náš" programátor, jakožto uživatel Application designera, ovšem jeho práva k programu jsou pouze derivativní – jsou odvozena od práv poskytovatele licence k šabloně kódu, která byla použita v rámci Application designera, a výsledný program je odvozené dílo od takovéto šablony. Platí tu tedy jistá analogie s případem programového frameworku. Je tedy nezbytné při následné distribuci výsledného počítačového programu dbát na zachování souladnosti s původní licencí k použité šabloně a v tomto ohledu tedy může být omezeno i nakládání s výsledným programem.

Tento závěr lze podpořit i příkladem z praxe. Softwarové společnosti na podobné záležitosti zřejmě již myslely, a proto například v EULA[31] již zmiňovaného Microsoft Visual Studia nalezneme ujednání, které se této problematice věnuje[32]. Jedná se o článek 3 citované EULA, nadepsaný *Šiřitelný kód*. Výslovně počítá s tím, že Software (tedy Visual Studio) obsahuje

[29] TETER, Timothy S. Merger and the Machines: An Analysis of the Pro-compatibility Trend in Computer Software Copyright Cases. *Stanford Law Review* 45.4 (1993). s. 1061–1098.
[30] Zrychlené vložení „if-else" či „while" bloků.
[31] End-User License Agreement
[32] MICROSOFT. *Microsoft Software License Terms for Microsoft Visual Studio Community 2013* [online]. 2013 [citováno 16. 3. 15]. Dostupné z: https://www.visualstudio.com/en-us/dn877550.aspx.

kód, který je uživatel (programátor) oprávněn dále šířit, jako jsou například *ukázky kódu, šablony a styly*. Je zajímavé, že šíření je zde výslovně vztáhnuto i na šíření formou distribučního modelu SaaS[33]. Oprávnění šířit tento kód ovšem není neomezené a podléhá splnění několika podmínek. Kromě částečně redundantních negativních závazků neužívat ochranné známky Microsoftu a neupravovat poskytnuté oznámení o autorských právech připadají v úvahu zajímavější podmínky, které mohou dosáhnout dokonce až na následného nabyvatele licence k „našemu" výslednému programu. Jedná se o povinnost před šířením *Šiřitelných kódů* přidat k nim „významnou primární funkcionalitu"[34], požadovat po distributorech a koncových uživatelích výsledného programu, aby byli vázáni takovou licencí, která chrání *Šiřitelný kód* minimálně stejnou měrou, jako tato EULA, a konečně podmínka *nelicencovat výsledný program pod Vyloučenou licencí*. Vyloučenou licencí se přitom rozumí jakákoli licence, která (i) požaduje otevření software ve formě zdrojového kódu nebo (ii) uděluje ostatním právo *Šiřitelný kód* upravit. Bude se tedy typicky jednat o tzv. open-source licence, jako je například GNU GPL či jiné GNU licence[35] nebo Mozilla Public License[36].

Použití IDE tedy může podstatným způsobem limitovat programátora v následné volbě distribučního modelu pro svůj software a je nutné tyto otázky zvážit před započetím vývoje.

4 Kompilace a kompilátor

Posledními pojmy, kterými se budeme zabývat, jsou *kompilace* a *kompilátor*. Jsou to příbuzné pojmy nejen lexikálně, ale i obsahově, když kompilátor můžeme definovat jako specializovaný počítačový program, který provádí kompilaci. Pro ucelenost výkladu a možnost navázání dalších úvah je v tuto chvíli nutné učinit stručný technický exkurs do problematiky kompilace počítačových programů. Nespokojíme se s tautologií naznačenou výše a vezmeme si na pomoc definici. Jednoduchou, ale užitečnou definici nám poskytuje server PCMag:

> *Software that translates a program written in a high-level programming language (C/C++, COBOL, etc.) into machine language. A compiler usually generates assembly language first and then translates the assembly language into machine language. A utility known as a "linker" then combines all required machine language modules into an executable program that can run in the computer.*[37]

[33] Detailněji k otázkám a problematickým bodům licencování SaaS například TOMÍŠEK, Jan. Licence při poskytování software jako služby. In *Revue pro právo a technologie: odborný recenzovaný časopis pro technologické obory práva a právní vědy* [online]. Brno: Masarykova univerzita, 2014, roč. 5, č. 10, s. 47-69 [cit. 2015-03-16]. Dostupné z: https://journals.muni.cz/revue/issue/viewIssue/245/pdf_8.

[34] Orig. „Significant primary functionality"; překlad autor.

[35] Seznam a aktuální znění k dispozici z: https://www.gnu.org/licenses/licenses.html

[36] Mozilla Public License, dostupná z: https://www.mozilla.org/MPL/

[37] THE COMPUTER LANGUAGE COMPANY, INC. *Definition of: Compiler* [online]. 2015 [citováno 2015-03-16]. Dostupné z: http://www.pcmag.com/encyclopedia/term/40105/compiler.

Kompilace je tedy proces, který je u některých jazyků nezbytný k vytvoření spustitelného souboru. Citovaný zdroj obsahuje i přehlednou ilustraci funkcionality kompilátoru, na které je patrná postupná transformace původního zdrojového kódu až do strojového kódu. Zároveň je zřejmé, že kompilace je prováděna jen u programovacích jazyků fungujících na principu překladu zdrojového kódu do jiné podoby. První možností jsou jazyky překládané do bytekódu, což je zkompilovaný zdrojový kód, který ale neplní stejnou funkci jako plnohodnotný strojový kód. Nemůže být spuštěn sám o sobě, ale potřebuje ke svému provedení program zvaný *interpret*, který jej na cílovém stroji načte a provede. Druhou možností jsou „klasické" kompilované programovací jazyky, u kterých je výsledkem kompilace strojový kód, který je sám o sobě spustitelný na cílovém stroji. Zde je nutné na okraj podotknout, že kompilace je vždy prováděna pro nějakou konkrétní konfiguraci cílového stroje. Neexistuje tedy kód, který by mohl být se stejným výsledkem spuštěn na všech programovatelných strojích. Vše je otázkou specifikace procesorové jednotky. To však není předmětem této práce a další výklad není nezbytný pro naše zkoumání.

Zjednodušeně řečeno kompilace funguje tak, že kompilátor prochází zdrojový kód a na základě svých vlastních instrukcí jej postupně překládá do podoby, která je na cílovém počítači považována za spustitelnou. To s sebou nese velice důležitý aspekt činnosti kompilátoru: pro zajištění spustitelnosti výsledného zkompilovaného kódu je nutné ke zdrojovému kódu přidat další části, na které zdrojový kód ve své původní podobě odkazuje. Kdyby tyto části ve výsledném kódu chyběly, spuštění programu by se nemohlo podařit, protože by strojový kód odkazoval na místa, která by nebyla definována. Nabízí se poměrně přiléhavá analogie z právního prostředí: ve smlouvě se vyskytuje odkaz na obecné obchodní podmínky smluvních stran. Pokud chceme smlouvu přečíst a zároveň pochopit, abychom zjistili přesný obsah závazku mezi stranami, je nutné prostudovat i obchodní podmínky. Mohli bychom tedy spojit obchodní podmínky i smlouvu do jednoho dokumentu, protože jsou k sobě právně navázány. Následně bychom si při čtení vytvářeli obsah závazku s ohledem na celek a nemuseli bychom nutně reflektovat například strukturu smlouvy nebo řazení jednotlivých článků. Naším cílem by bylo detailně zjistit, jaké jsou práva a povinnosti stran a co je předmětem plnění. Ve své podstatě se jedná o určitou formu analýzy. Kompilátor pracuje na podobné bázi – odstraňuje z kódu balastní informace a zapracovává všechny relevantní soubory do jednoho výsledného strojového kódu, přičemž respektuje definici použitého programovacího jazyka (tedy výrazům použitým v překládaném kódu připisuje významy definované ve specifikaci programovacího jazyka). Implementace programovacího jazyka kompilátorem spočívá v tom, že kompilátor v sobě zahrnuje jazyková pravidla a algoritmus jeho překladu do nižších vrstev jazyka, které se blíží výslednému strojovému kódu, a zpravidla i standardní knihovny. Z toho mimo jiné vyplývá,

že každý kompilátor je určený jen pro některé programovací jazyky. Stejně jako neexistuje univerzální strojový kód, spustitelný na všech strojích, neexistuje ani univerzální kompilátor, použitelný pro všechny jazyky.

4.1 Právní aspekty překladu a linkování

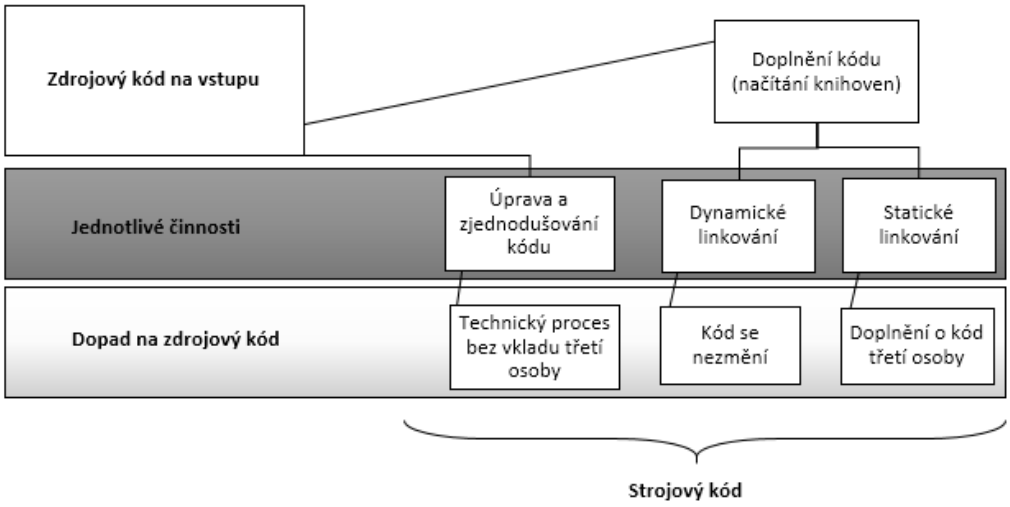
Kompilátor tedy provádí dva druhy činností: jednak překládá zdrojový kód do nižších vrstev, jednak se stará o to, aby byly výslednému strojovému kódu přístupné všechny dodatečné součásti, tzv. knihovny.

První činnost spočívá v pouhém technickém přeformulování stávajícího kódu a probíhá zcela automatizovaně jeho převodem z výrazů definovaných v dokumentaci programovacího jazyka do strojových příkazů definovaných v dokumentaci cílového stroje, pro nějž kompilace probíhá. V této souvislosti je důležité si uvědomit, že během tohoto procesu neprobíhá žádný kvalitativní vklad do díla a nepoužívají se ani žádné kódy třetích osob. Kompilátor je z tohoto pohledu pouze nástrojem, který kód předem daným způsobem zpracovává.

Druhá činnost, tedy zajišťování přístupnosti knihovních funkcí, je o poznání zajímavější. Může být totiž provedena dvěma různými způsoby, a který způsob se použije, je volbou programátora. Obecně se spojování různých souborů nazývá linkování[38]. To může být buď statické, nebo dynamické. Statické spočívá v tom, že se zdrojové kódy linkovaných souborů (tedy nejčastěji knihoven), respektive jejich relevantní části, v průběhu kompilace vloží do hlavního souboru se zdrojovým kódem, který vytvořil programátor. Výsledný kód, nad kterým je následně provedena výše zmiňovaná první činnost spočívající v jeho zjednodušování a přeformulování, je tedy obsáhlejší a zahrnuje některé kódy, které sám programátor nepsal – pouze jejich užití zapříčinil tím, že volal odpovídající knihovní funkce a použil statické linkování. Dynamické linkování dosahuje kýženého výsledku jinak. Můžeme si jej představit jako blanketní odkaz – kompilátor na příkaz programátora do výsledného strojového kódu vloží informaci, že některé knihovní funkce se mají hledat ve specifických souborech[39]. Ty přitom vůbec nemusí být součástí distribuce výsledného počítačového programu.

[38] Tamtéž.
[39] Soubory typu DLL, Dynamic-link Library.

Následující diagram ilustruje proces kompilace z pohledu práva:



Obrázek 1: Diagram činností kompilátoru

Diagram znázorňuje právně relevantní činnosti spojené s kompilací počítačového programu. Vzhledem k výše uvedenému tedy docházíme k následujícím závěrům:

1. Pouhé technické zjednodušení existujícího kódu není podstatným vkladem do díla, a proto nemá vliv na autorskoprávní status zkompilovaného programu. Situace je zčásti podobná použití IDE rozebranému výše. Pouhé technické zázemí poskytnuté jiným programem nemá za následek vznik odvozeného díla od původního zdrojového kódu programátora, neboť tento zdrojový kód nijak nemění.
2. Linkování souborů a knihoven třetích osob, které je prováděno dynamickou metodou také nemá za následek změnu zdrojového kódu, která by byla vyvolána inkorporací linkovaného souboru do původního zdrojového kódu. Zde opět není na místě dovozovat autorskoprávní dopady.
3. Naopak statické linkování má za následek reálné rozšíření původního zdrojového kódu o další části, které jsou vtaženy do kompilovaného kódu a následně zařazeny do výstupního spustitelného souboru. Situace je tedy obdobná jako u použití frameworku – kódy vytvořené třetí osobou jsou použity při vývoji „našeho“ počítačového programu takovým způsobem, že jsou zařazeny do výsledného programu a distribuovány společně s ním, jakožto jeho součást.

V případě použití kompilovaného programovacího jazyka, nebo interpretovaného programovacího jazyka založeného na technologii překladu do bytekódu je tedy nutné se při metodě statického linkování cizích souborů vypořádat s případnými licenčními omezeními vážícími se k linkovaným souborům.

Lawrence Rosen ve svém – byť postarším – článku, publikovaném na webu Linux Journal[40] argumentuje proti této tezi, když tvrdí, že by se neměl reflektovat rozdíl v technickém provedení linkování, pokud není z okolností zřejmé, že tvůrce programu chápal tento rozdíl a možný dopad na problematiku odvozených děl. Dovolím si s tímto názorem nesouhlasit. Jak bylo demonstrováno výše, rozdíl mezi těmito použitými technikami je evidentní a mimořádně důležitý. V případě statického linkování dochází fyzicky ke zhotovování rozmnoženiny linkovaného souboru nebo jeho kvalitativně podstatných částí a tedy nový program, byť svou funkčností může být zcela odlišný, je odvozeným dílem od linkovaného souboru. Neobstojí ani argument, že takovéto právní rozlišování je zbytečnou zátěží pro právní systém a že právo nestíhá na technologický vývoj reagovat. Samozřejmě, že ze své podstaty rigidní právní úprava bude po nějakou dobu nedotčená a nebude se měnit s každou novou technologií, aby nám poskytla odpovídající kazuistickou úpravu. V tomto případě nám postačí, že právní úprava poskytuje koncept toho, jak vznikají odvozená a díla a jaká práva se v takovém případě váží. Výkladem můžeme poměrně dospět k jednoznačnému výsledku.

Dalším argumentem, který můžeme uvést ve prospěch zde vyloženého přístupu, je, že knihovny použité při vývoji počítačových programů jako předměty linkování, jsou samy o sobě bez jakýchkoli pochybností autorskými díly. Implementují některé základní funkce programovacího jazyka, ale činí tak autorsky originálně. Můžeme si dokonce vybrat, ze které knihovny danou funkci využijeme a můžeme se setkat s různými výsledky co do možností a výkonu použitého řešení. Není důvod odpírat tvůrcům knihoven možnost dále kontrolovat život svých autorských děl, stejně jako není tato možnost odpírána tvůrcům frameworků. V případě, že by daná knihovna opravdu nesplňovala znaky autorského díla, pochopitelně by se omezení daná v její licenci neaplikovala na výsledný program v duchu zásady *nemo plus iuris*.

Navíc, i pokud bychom připustili, že softwarová knihovna není natolik důležitým autorským dílem, aby zasluhovala stejné míry ochrany, jako ostatní počítačové programy, a tím bychom limitovali možnost omezit licenci pro jejího uživatele-programátora, stále bude existovat možnost paralelně sjednat mezi autorem omezeně chráněné knihovny a programátorem mimolicenční smluvní závazek. V rámci toho by bylo možné licenční omezení suplovat. Zbytečně by se ovšem zhoršila pozice autora knihovny z hlediska dokazování a nároků z odpovědnosti za porušení smlouvy.

Posledním argumentem je příklad z praxe. Free Software Foundation, společnost zajišťující projekt GNU, která vydává a udržuje soubor veřejných softwarových licencí rodiny GNU GPL,

[40] ROSEN, Lawrence. *Derivative Works* [online]. 2003 [citováno 2015-03-16]. Dostupné z: http://www.linuxjournal.com/article/6366.

mimo jiné vydala i licenci GNU LGPL – tedy Lesser GPL; licenci, která je o něco permisívnější, než standardní GPL. LGPL se původně jmenovala Library General Public License, což značilo úmysl poskytnout ji vývojářům knihoven pro jejich licencování programátorům a koncovým uživatelům. LGPL se pro licencování knihoven poměrně hojně používá. Free Software Foundation se aktuálně snaží její použití omezovat a apeluje na vývojáře, aby pro knihovny do budoucna používali standardní GPL z důvodu přílišné permitivity LGPL ve vztahu k zařazování knihoven do proprietárního (uzavřeného) software.[41] Postoj FSF a používání copyleftových licencí pro samotné knihovny jen dokazuje, že není možné odmítnout vliv licence knihovny na výsledný produkt.

Závěrem

V této práci jsme provedli analýzu několika základních prvků, které jsou běžně využívány při vývoji počítačových programů. Jednalo se o *programovací jazyk*, *Framework*, *IDE*, *šablony kódu* a *kompilátor*. Na základě srovnání s platnou legislativou a vybranou smluvní dokumentací jsme vyhodnotili vliv těchto prvků na právní postavení výsledného počítačového programu, respektive na případnou existenci limitace jeho tvůrce z hlediska nakládání s takovým programem.

Dospěli jsme k obecnému závěru, že k takovéto limitaci dochází vždy tam, kdy je v rámci vývoje počítačového programu využito nástroje, který podstatným způsobem ovlivňuje znění zdrojového kódu výsledného počítačového programu nebo pro něj poskytuje jiné zdrojové soubory. Zejména se jedná o případ stavby počítačového programu na existujícím základě využitím hotového frameworku nad daným programovacím jazykem, dále o případy využití připravených šablon kódu, a to i v rámci designování aplikace pomocí grafického rozhraní (tzv. Application designer) a konečně o využití kompilovaného jazyka s využitím statického linkování knihoven a jiných souborů do výsledného strojového kódu programu.

V dalším zkoumání bude nutné provést aplikaci získaných závěrů na praktický příklad a dále provést jejich další extenzivní testování za účelem ověření jejich způsobilosti korektně regulovat rozličné právní případy.

[41] FREE SOFTWARE FOUNDATION. *Why you shouldn't use the Lesser GPL for your next library* [online]. 2014 [citováno 2015-03-16]. Dostupné z: https://www.gnu.org/licenses/why-not-lgpl.html.

Seznam zdrojů

[1] AABY, Anthony A. *Introduction to Programming Languages* [online]. 2004 [citováno 2015-03-16]. Dostupné z: http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/intro.htm.

[2] DOCFORGE. *Framework* [online]. 2013 [citováno 2015-03-16]. Dostupné z: http://docforge.com/wiki/Framework.

[3] FREE SOFTWARE FOUNDATION. *Licenses* [online]. 2015 [citováno 2015-03-16]. Dostupné z: https://www.gnu.org/licenses/licenses.html.

[4] FREE SOFTWARE FOUNDATION. *Why you shouldn't use the Lesser GPL for your next library* [online]. 2014 [citováno 2015-03-16]. Dostupné z: https://www.gnu.org/licenses/why-not-lgpl.html.

[5] MICROSOFT. *Microsoft Software License Terms for Microsoft Visual Studio Community 2013* [online]. 2013 [citováno 16. 3. 15]. Dostupné z: https://www.visualstudio.com/en-us/dn877550.aspx.

[6] MOZILLA. *Mozilla Public License version 2.0* [online]. 2012 [citováno 2015-03-16]. Dostupné z: https://www.mozilla.org/MPL/2.0/.

[7] PECINOVSKÝ, Rudolf. *OOP – Learn Object Oriented Thinking and Programming* [online]. 2013 [citováno 16. března 2015]. Dostupné z: http://files.bruckner.cz/be2a5b2104bf393da7092a4200903cc0/PecinovskyOOP.pdf.

[8] RIEHLE, Dirk. *Framework Design: A Role Modeling Approach* [online]. 2000 [citováno 2015-03-16]. Disertační práce. Zürich, Switzerland, ETH Zürich. Dostupné z: http://dirkriehle.com/computer-science/research/dissertation/diss-a4.pdf.

[9] ROSEN, Lawrence. *Derivative Works* [online]. 2003 [citováno 2015-03-16]. Dostupné z: http://www.linuxjournal.com/article/6366.

[10] Směrnice Evropského parlamentu a Rady 2009/24/ES ze dne 23. dubna 2009, o právní ochraně počítačových programů.

[11] ŠAVELKA, Jaromír. Autorskoprávní ochrana funkcionality softwaru [online]. 2013 [cit. 2015-03-15]. Rigorózní práce. Masarykova univerzita, Právnická fakulta. Vedoucí práce Filip Křepelka. Dostupné z: http://is.muni.cz/th/134449/pravf_r/>.

[12] TETER, Timothy S. Merger and the Machines: An Analysis of the Pro-compatibility Trend in Computer Software Copyright Cases. *Stanford Law Review* 45.4 (1993). s. 1061–1098

[13] THE COMPUTER LANGUAGE COMPANY, INC. *Definition of: Compiler* [online]. 2015 [citováno 2015-03-16]. Dostupné z: http://www.pcmag.com/encyclopedia/term/40105/compiler.

[14] TOMÍŠEK, Jan. Licence při poskytování software jako služby. In *Revue pro právo a technologie: odborný recenzovaný časopis pro technologické obory práva a právní vědy* [online]. Brno: Masarykova univerzita, 2014, roč. 5, č. 10, s. 47-69 [cit. 2015-03-16]. Dostupné z: https://journals.muni.cz/revue/issue/viewIssue/245/pdf_8.

[15] VENÂNCIO, Pedro Dias, Ph.D. Electronic File Formats – free to use under Copyright Law. In *3rd International Research Forum Göttingen*. 2014.

[16] Zákon č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění účinném ke dni 16. března 2015.

[17] ZBRÁNEK, Lukáš. *Právní aspekty zpětné analýzy počítačových programů* [online]. 2012 [cit. 2015-03-16]. Diplomová práce. Masarykova univerzita, Právnická fakulta. Vedoucí práce M. Myška. Dostupné z: http://is.muni.cz/th/98818/pravf_m.

* * *