
Теоретични импликации на софтуерната революция

Издание 1.0.1

Калоян Доганов

13 февруари 2001 г.

Интернет страница: <http://revolution.sourceforge.net>
Електронна поща: kaloian@europe.com, kaloian@users.sourceforge.net

Copyright © 2001 Kaloian Doganov. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Section being Bibliography, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

СЪДЪРЖАНИЕ

1	Въведение	1
2	Софтуерът	4
2.1	Да започнем от предмета	4
2.2	Една история на натрупването	9
2.2.1	Нула и единица	9
2.2.2	Подпрограми и библиотеки	12
2.2.3	Набиране на височина	15
2.2.4	Производство на отчуждение	18
3	Производството на свободния софтуер. История и теория.	23
3.1	История на GNU. Ричард Столман	24
3.1.1	Паралелни светове	25
3.2	Катедралата и базарът	27
3.2.1	Парадоксални отношения	32
3.2.2	Поуката от Mozilla	35
3.3	Общество в криза	37
4	Заклучение	39
A	GNU Free Documentation License	43
A.1	Applicability and Definitions	43
A.2	Verbatim Copying	44
A.3	Copying in Quantity	44
A.4	Modifications	45
A.5	Combining Documents	46
A.6	Collections of Documents	46
A.7	Aggregation With Independent Works	47
A.8	Translation	47
A.9	Termination	47
A.10	Future Revisions of This License	47

Въведение

На пръв поглед, едва ли има нещо по-чуждо от философията и софтуера. Двете сфери изглеждат толкова отдалечени една от друга, че съпоставянето им или мисленето за тях като съотносими, изглежда проблематично. Затова преди да се захванем с каквото и да било друго, трябва първо да отговорим на въпроса: „Защо тези две сфери кореспондират една с друга?“ Трябва да покажем, че такава връзка е възможна. И не само възможна, но и необходима.

В областта на софтуера се случва нещо вълнуващо. То се случва бързо, като водовъртеж, който подема много участници и ги завихря в нови отношения. То едновременно привлича множество ентузиастични и бива заклеймявано, печели защитници и обвинители, поражда надежди и страхове. Като движение, то се ръководи от сладководни идеолози, след които маршируват зашеметяващи по размера си щастливи тълпи последователи. Огромен брой университетски преподаватели, студенти, и изобщо професионалисти, са се захванали с едно „немислимо“ начинание.

Именно това нещо е свободният софтуер. Съществуването му като действителна идея и действително движение поставя под въпрос основанията на традиционните механизми на работа в отрасъла. А колцина от днешните му привърженици само преди няколко години са предполагали, че това е възможно? Този и други подобни въпроси разтърсват компютърната индустрия и водят до създаването на нови теории върху същността на програмирането.

Но свободният софтуер не е просто някакъв нов вид програмистка техника. Като практика за създаване на софтуер, той вече е изследван, тъй като първите му теоретици са преди всичко програмисти. Но същите тези блестящи теоретици изпадат в неудобно положение, когато се напусне сферата на техническите подробности и идейните предпоставки. Не могат да бъдат извлечени самите последствия от тези предпоставки и от съществуването на свободния софтуер като такъв. А още по-малко е възможно идеята да се схване исторически. След като прочете „Катедралата и базарът“¹, човек започва да се чуди как изобщо свободният софтуер не е залял света по-рано.

Има нещо за софтуера, и по-специално за свободния софтуер, което самият той не може да доизкаже докрай за себе си. Именно затова има необходимост от помощта на една философска рефлексия.

През същото това време, положението във философията е близко до обратното. В академичните аудитории неизменно се задава и разглежда теоретичният въпрос: „Що е философия?“, чиито отговор несъмнено е философска задача, в случая разгърната в образованието по история на философията. В неофициални разговори извън аудиториите, горният въпрос намира съответен израз в множество форми: „За какво ни е философията?“, „Има ли нужда от философия?“, „За какъв дявол се забихме да учим философия?“, че дори и „Откъде идва самочувствието на занимаващия се с философия?“² Тези въпроси тревожат всеки студент. По-късно тревогата допълнително се усилва при установяването на факта, че в днешния вестник не може да се открие обява за работа: „Спешно търсим философ.“³

¹Една блестяща студия, която задава публично валидната рефлексия върху свободния софтуер ([14]).

²Така беше озаглавена една дискусия на студенстката група „Проектория“.

³Тук терминът „философ“ е употребен в тривиалното си значение, т.е. човек, завършил висше образование

От една страна, това се дължи на самата философия като занимание с абстракции. Как можем да очакваме от тези абстракции да получат реален, практически израз? Как можем наистина да очакваме някаква полза? Но това е наивната страна на въпроса, защото от друга страна, никой не проблематизира ползата от, да речем, математиката, която също оперира с абстракции от високо ниво. Впрочем авторитетът като качество е обществен продукт, а не нещо, което можем да открием, взирайки се в авторитетния. Идеята, че някой авторитет може гордо да се разхожда в изолацията на самотен остров, е една „лишена от фантазии робинзониада“. Авторитетът лежи извън авторитетния. Въпросът „Що е философия?“ е необходим и важен, но не може да бъде използван за нашите цели, защото така философията ще остане в своите собствени граници. Ще спечели територии, които и без това притежава.

Без да престава да бъде силно рефлексивна, философската практика трябва да демонстрира своята ценност в разглеждането на *друг*, различен от нея, предмет. Освен всичко друго, със собствената си актуалност една работа по философия трябва да покаже:

1. Има необходимост от изучаването на философия.
2. Това изучаване непременно трябва да преминава през и да се опира на философската традиция.

Първата задача се постига, като философски се разглежда един съвременен предмет, може би дори най-съвременния. Ако изследването на предмета с философски инструментариум е плодотворно, това ще покаже, че философията е наука и то необходима наука. Следователно това трябва да бъде едно ново, съвременно изследване.

Втората задача се постига, като анализът на предмета се опира на философски текстове, което се явява опит за изследване на самата плодотворност на тези текстове. Но защо е избран именно Маркс, а не, да речем, Киркегор или Хайдегер? Наистина, този предмет може да бъде анализиран и от други изходни позиции, с друг инструментариум. (Тук не се повдига претенцията, че се изчерпва единствения възможен поглед.) Маркс е предпочетен най-вече заради своеобразната адекватност на разглеждания предмет, представляващ една специфична сфера на дейност. За да запазим тази адекватност, на нас ни е необходима една философия на практиката. Но творчеството на Карл Маркс не ни е дадено. Използването му ще бъде едновременно изследване на предмета и реконструиране, наново интерпретиране на тези текстове. Настоящата дипломна работа няма претенцията, нито има за цел да постигне Маркс „такъв, какъвто е“. Напротив, тя се опитва да приложи методологията на Маркс, осветявайки един нов, съвременен предмет. Впрочем, изборът на Маркс да критикува своето съвремие, не е самоцел, а е необходим резултат от неговите философски позиции. Основанието на настоящата работа е, че днес не можем да консервираме тези позиции, оставяйки при Маркс. Тъкмо за да останем верни на Маркс, трябва да разгледаме явления от съвременната действителност. През последните години от живота си, Енгелс пише: „Но целият светоглед на Маркс е не доктрина, а метод. Той дава не готови догми, а опорни точки за по-нататъшно изследване и метода за това изследване.“ ([4] с.113)

Ще се опитаме да покажем, че в тези текстове, въпреки съществуващите, натрупали се през годините разнообразни и всеобхватни интерпретации, все още лежи голям евристичен потенциал. Следователно, това изследване ще се яви като изследване върху една философска традиция.

Настоящата работа има за *цел* да осъществи един експеримент за приложение на Марксовия методологически комплекс като направи един конкретен разрез на софтуера като предмет. На първи план изложението е подредено исторически – ще се тръгне от генезиса на предмета, където логиката и историята се намират в своеобразно отношение. Тук именно Маркс е необходим, защото предметът не се разглежда само като изолирана вещь със своя собствена логика, а като „сетивно-свръхсетивна вещь“ зад която стои цял един свят със своята богата тоталност от множество определения и отношения. Освен свободния софтуер по философия.

като предмет, тук ще бъде разгледана една не само широко разпространена, но което е важно за нас, придобила *валидност* рефлексия върху софтуера като специфична реалност, а именно – „Катедралата и базарът“ на Ерик Реймънд. Тази студия вече е задава принципите, чрез които се мисли свободния софтуер. Тази критическа нагласа е необходима, защото така би могло да се проследи отношението между предмет, легитимната рефлексия върху него, и нейните собствени основания.

Тази работа няма действително разказвателен характер. Историята и биографията тук не се разглеждат като низ от курioзни случки и случайности, а като необходима зависимост в един контекст.

Софтуерът

2.1. Да започнем от предмета

Предметът, който имаме да разглеждаме тук, е твърде особен. За да не бъде нашето разглеждане поставено във въздуха, трябва най-напред да се докоснем до тази особеност. Може би наистина не е нужно да си Цезар, за да разбереш Цезар, но със сигурност най-лесният начин да го направиш е като станеш Цезар за малко. Най-лесният начин да разберете какво представлява програмирането, е като за малко застанете в позицията на програмист. Това не е никак трудно, въпреки широко разпространеното мнение, и тук аз ви каня да го направите. За да е възможно това, ще ни е необходимо само малко въображение.

Представете си един правоъгълен лист хартия с размер 100x100. Листът е мислено разграфен с познатата ни Декартова координатна система (долният ляв ъгъл е с координати (0,0), а горният десен е с координати (99,99)). Над този лист стои просто устройство, което разполага само с една механична ръка, държаща молив. Отстрани на нашия работен лист стоят наредени четири номерирани молива с различен цвят – черен (номер 0), червен (номер 1), зелен (номер 2) и син (номер 3). По наше желание ръката може да сменя молива, който държи. След малко ще видим как точно става това.

Механичната ръка може да се движи свободно по листа, като има пълен достъп до всичките му точки (в границите 100x100). Ръката има два режима на движение по листа:

Със спуснат молив – тогава моливът оставя следа по листа. Цветът на следата зависи от цвета на молива, който ръката държи в момента. Черният молив би оставил черна следа.

С вдигнат молив – тогава моливът не се докосва до листа и не оставя следа.

Тъй като нашето устройство е напълно лишено от инициатива, ние можем да го управляваме, като последователно му подаваме прости команди. То ни предоставя ограничен набор (краен брой) команди, които автоматично преобразува в реални движения на ръката си. Командите са общо 5 на брой и са номерирани от 10 до 14:

- 10 – Премества ръката до точка с определени координати (абсолютни координати).
- 11 – Премества ръката до точка с определени координати, спрямо текущата позиция (релативни координати).
- 12 – Вдига молива нагоре, така че да не оставя следа при движение.
- 13 – Спуска молива надолу (опирайки в листа), така че да оставя видима следа при движение по листа.
- 14 – Сменя молива с друг.

И тъй, устройството стои пред нас и ние можем да експериментираме като го караме да рисува по хартиения лист. В началото листът е празен и изглежда както е показано на фигура 2.1 (с пунктир са показани границите на листа).

Ако сега подадем команда:



Фиг. 2.1. Празен лист, готов за експерименти

14 0

Това ще накара ръката да вземе черния молив. Ако се чудите защо, ето какъв е ходът на мисълта. Първото число е 14, което означава команда номер 14. Това е командата за смяна на молива. Следващото число е номера на молива, който искаме да вземем – в случая, това е черния молив, за който по-горе казахме, че е с номер 0. Ако леко променим командата така:

14 3

Ще накараме ръката на устройството да грабне зеления молив (с номер 3).

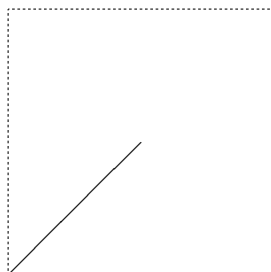
За да позиционираме молива в определна точка на листа, например точката с координати (0,0), трябва да подадем командите:

10 0 0 13

Ето как се тълкува тя. Първото число, 10, е номерът на командата за преместване на ръката до определени координати. Следващите две нули представляват конкретните координати ($x = 0$, $y = 0$). След като машината получи координатите, тя премества ръката си до определената точка. Следващата команда е с номер 13, която кара машината да спусне молива до повърхността на листа, при което моливът оставя следа – една точка. Ако в този момент, когато моливът е спуснат, ние решим да кажем на машината да го премести до друга точка, на листа ще се нарисова права линия, свързваща двете точки – първоначалната и крайната точка на движението на молива. Ето така:

10 50 50

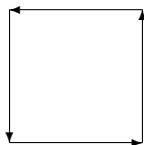
Така моливът отива на точка с координати (50,50), които попадат точно в средата на листа. Получената линия изглежда както е показано на фигура 2.2.



Фиг. 2.2. Нарисувахме първата си линия

И тъй, моливът е в средата на листа. Ако искаме да го предвижим до друга точка, без да чертаем линия, трябва преди това да вдигнем молива нагоре:

Така безопасно преместваме молива над точката с координати (60,60). Нека тук пробваме нещо по-сложно, като например нарисуваме квадрат. С отделни последователни движения на молива трябва да начертаем всяка от страните на квадрата. За размер на страната на квадрата ще изберем стойността 30. Тоест трябва да начертаем четири линии с този размер, свързани по подходящ начин. На фиг. 2.3 е показан един възможен начин.



Фиг. 2.3. План на движенията на молива, описващи квадрат

За наше улеснение, машината ни предоставя възможността да работим с релативни координати. Какво означава това? Вече казахме, че листът е мислено разграфен и представлява една координатна система с точка (0,0), съвпадаща с долния ляв ъгъл на листа. Нека сега си представим, че съществува още една въображаема координатна система, чиято отправна точка (0,0) винаги съвпада с върха на молива. Координатната система, взимаща ъгъла на листа за отправна точка, наричаме абсолютна. А координатната система, взимаща молива за отправна точка, наричаме релативна. По този начин, ако въведем командата:

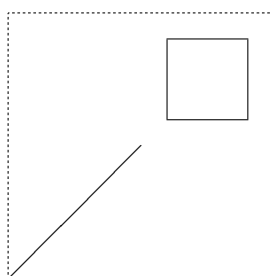
```
11 1 0
```

Ще предвижим молива с една точка надясно, независимо от текущата му позиция. Ако текущата позиция е (0,0), моливът ще се премести до точка (1,0), а ако текущата позиция е (19,45), моливът ще се премести до точка (20, 45). Ще видите колко е удобно това, когато рисуваме нашия квадрат. Сега да се опитаме да опишем командите, които си представихме във фиг. 2.3.

```
11 30 0 11 0 30 11 -30 0 11 0 -30
```

Тази последователност от команди ще раздвижи ръката, но няма да начертае квадрат, ако преди това не спуснем молива, за да опре в листа. Така че трябва да повторим движенията за квадрата отначало, за да получим това, което е изобразено на фиг. 2.4:

```
13 11 30 0 11 0 30 11 -30 0 11 0 -30
```



Фиг. 2.4. Крайният резултат от изчертаването на квадрата

В крайна сметка на листа се получи някаква странна фигура, която лично на мен ми прилича на хвърчило. Ако си спомняте, ние стигнахме дотук като подавахме команди – поредица от инструкции за машината, които тя изпълняваше докато накрая се появи хвърчилото. Целесъобразната последователност от инструкции, образуващи едно логическо цяло, наричаме *програма*. Да си спомним цялата последователност от инструкции, които изпълнихме,

за да стигнем дотук:

14 0 10 0 0 13 10 50 50 12 10 60 60 13 11 30 0 11 0 30 11 -30 0 11 0 -30

Поздравления! Това е програмата „Хвърчило“, нашата първа програма. Ние я съставихме за една въображаема машина, и така се оказахме на мястото на първия програмист в света. По подобен начин, в средата на 19 век лейди Ада Лавлейс¹ съставя последователност от команди за Аналитичната машина на Чарлз Бабидж², която била все още проект. Може би има нещо симптоматично в това, че първата програма е създадена във въображението, без налична машина.

Разбира се, Аналитичната машина на Бабидж е била предназначена за изчисления, а не за чертане, и лейди Лавлейс не е рисувала хвърчила, а се е концентрирала върху специфична математическа задача – изчислявала е т.нар. коефициенти на Бернули. Тук ние използвахме по-интересен и нагледен пример с рисуване на изображения. Това, което трябваше да се покаже, е всъщност онова, което лежи в основата на програмирането – описанието на *сложни* задачи чрез относително *прости* команди. Комбинирането и подреждането на простите, елементарни команди в целесъобразна последователност води до резултат, който е съвсем различен от простия сбор от команди. Ако отново разгледате набора от команди на нашата машина, там няма да видите никакво хвърчило.

Примерът с нашата рисуваща машина работи, защото се демонстрира този принцип. Нашата машина няма и представа от хвърчила. Колко команди ни предоставя тя? Броят се на пръстите на ръката – точно 5. Колко различни изображения биха могли да се нарисуват с тях? Хвърчила, петолъчки, архитектурни чертежи, може да се пише произволен текст като се изрисуват буквите, и още каквото ви хрумне – на практика това са безкраен брой изображения.

Тук има още една особеност, която трябва да се отбележи. Нашият пример е обвързан с въображаема машина за чертане, т.е. с конкретно проблемно-ориентирано устройство. Това устройство може да бъде използвано за чертане на изображения и нищо друго. В това отношение то не е компютър, а по-скоро наподобява „прародителя“ на компютъра – тъкачния стан на Жозеф-Мари Жакард. Станът на Жакард приемал команди и ги трансформирал в съответни механични движения.³ Командите се превръщали не в движения на молив (както в нашия пример), а в движения на стана, като по този начин се автоматизирал процеса на производство. Тези команди и движения на стана били специфични за производството на текстил. Важната крачка, която прави напред англичанинът Бабидж, е че не насочва тези, макар все още механични, действия към някаква определена приложна област. Целта на Бабидж е да автоматизира изчисленията, и неговата машина не е предназначена да рисува или тъче, а да извършва математически операции. Това от една страна предопределя първите приложения на изчислителната машина – за изчисления в научната сфера, а от друга страна прави машината *универсална*. Така всяка информация бива представена с числа. Именно защото не е насочена към определено приложение, машината вече може да бъде използвана за изчисляване на коефициенти на Бернули, за смятане на тръбите, пълнещи басейн с вода, за намиране на скоростта на влака, пътуващ от град А до град Б, и т.н – тоест, тя става

¹Августа Ада Байрон е родена на 10 декември 1815 г. в Англия, дъщеря на лорд Байрон. През 1833 става студент по математика, което е твърде необичайно по това време. Умира твърде млада през 1853, едва на 37 години.

²Чарлз Бабидж е роден на 26 декември 1791 г. в Девоншир, Англия. През 1810 г. постъпва в Тринити коледж, а през 1833 г. започва работа върху своята Аналитична машина и влага в нея своите пионерни идеи. Заради своя принос, днес Бабидж е известен като „башата на компютъра“. Той умира през 1871 г. в Лондон.

³Тук не се оспорват личните качества на Жакард, а само се набляга на на индустриалната необходимост от неговия стан. Би ли могъл Жакард да измисли стана си в стана на Аспарух? Същата уговорка се отнася за Бабидж и графиня Лавлейс. Доколкото „обстоятелствата създават хората в същата степен, в каквато хората създават обстоятелствата“ ([12] с.39), непременно трябва да се направи преглед на конкретната историческа ситуация, завихрянето на капитализма и развитието на промишлеността, в която се случват тези „чисто научни“ открития.

универсално приложима. Което е първата предпоставка за по-късната експанзия.

От едната страна стои Жакард, който е искал да направи революция в трудовия процес в своята област, в производството на текстил. Откритието му реално преобръща техническите и обществените условия на този трудов процес, повишавайки с това производителната сила на труда. От друга страна стои Бабидж, който, загърбвайки производството на стока, правейки изчислението своя цел, е сякаш неадекватен към заобикалящия го капиталистически водовъртеж. Но тази неадекватност е само привидна. Макар и математиката да е високоабстрактна наука, по времето на Бабидж вече е възникнала крещяща необходимост от нея. Доколкото всеки е принуден да участва в стоковата размяна, той е усетил тази необходимост. Развилите се пазарни отношения естествено възпитават любов към математиката. Но ако наемните работници смятат само с дребни числа, милионерите наистина изпитват необходимост от изчислителна мощ, за да могат да управляват капитала си. Освен това съществува една област – тази на банковото дело – където математическите изчисления стоят в сърцевината на самата дейност. Ако текстилната индустрия трябва да се оптимизира и претърпи революция, то банковото дело също изпитва необходимост от това. В тази светлина, съвсем не е случайно, че Бабидж е син на лондонски banker. Самият той не се занимава с управлението на банката на баща си, а отива да учи в Тринити коледж, където по-късно става знаменит Лукасов професор. Но това, което му се случва в личен план, вече е обществен факт. Капиталистическите отношения изискват постоянна революция в производството⁴, и това е дотолкова жизненоважно, че става възможно и необходимо да възникне и да се обособи науката като институция. Отделни хора да се ангажират с научна дейност, а останалите да ги поддържат като такива. Връзката между потребление и производство не е нито еднопосочна, нито непосредствена. Науката участва в една сложна система, усложнявайки я още повече чрез производството на материални и идеални посредници – „оръдия на труда“. Универсализацията и динамизацията на живота произвежда растеж на потребностите.

Така обществените отношения произвеждат Жакард, който произвежда своя стан. По същият начин, тази необходимост се проявява в Бабидж, който на 25 години се сдобива с налудничавата идея да конструира машина за изчисления. Една *универсално* приложима машина, която би могла да революционализира *всяко* производство.

Нещо повече – бивайки универсално приложима, такава машина може да бъде прилагана дори в собствената си област, да бъде предмет на самата себе си. Да си припомним нашия въображаем пример – и координатите, и командите, и цветовете на моливите (нали бяха номерирани) са числа. И ако командите там се използват само за движение на молива, в една изчислителна машина те се използват за изчисления. Ако изчисленията са операции с числа, то значи могат да се извършват операции дори върху самите операции.⁵ И резултатите от тях да бъдат наново използвани в следващите операции. Така едновременно стават възможни:

1. Универсалната приложимост на компютъра.
2. Отделянето на *софтуера* като самостоятелна сфера. Софтуер, често наричан още „програмно осигуряване“, „програма“, „код“, наричаме инструкциите, изпълнявани от даден компютър в противоположност на физическото устройство, на което те се изпълняват ([21] software). На технически жаргон, устройството се нарича *хардуер*⁶. Отчленяването и обособяването от хардуера предизвиква необходимост този създаден по аналогия неологизъм – *софтуер*.⁷

⁴ „Буржоазията не може да съществува, ако не революционизира постоянно оръдията за производство, а следователно и производствените отношения, ще рече и всички обществени отношения.“ ([11] с.27-29) Това твърдение е все още актуално, и всеки дръзнал да възрази би могъл да бъде съкрушен със Закона на Мур.

⁵ Голям принос за това по-късно има Джон фон Нойман (през 1946 г).

⁶ англ. *hardware* – железария, апаратура.

⁷ По-нататък за компютърните специалисти може да възникне объркване при общата употреба на термина *софтуер*, затова съм длъжен още тук да направя една техническа бележка: в тази работа софтуерът се разглежда

По-късно ще видим как тъкмо тази специфична рефлексивност на софтуера е втората главна предпоставка за експанзията му. Историята, която трябва да разкажем, е история на това прилагане на софтуера в собствената му област – там, където предметът, средството и продуктът съвпадат. Това съвпадение ще доведе до превръщането му в център на водовъртежа.

2.2. Една история на натрупването

2.2.1. Нула и единица

Вероятно вече сте чували клишетото, че компютрите познават само две цифри – 0 и 1. Исторически, това е наложено с оглед на намаляване на сложността на машината⁸. Това е двоичната бройна система. Първият въпрос, който хората задават като чуят това е «Като има само 0 и 1, как се изразяват по-големи числа?». По същият начин, по който в общоприетата десетична бройна система (с цифри от 0 до 9) се представят числа, по големи от 9 – чрез групиране на повече цифри. Разликата е само, че при двоичната бройна система групите от цифри нарастват много бързо.

Например, числото 14 в двоичен вид е 1110, т.е. в десетичната бройна система са необходими само две цифри за да се изрази това число, докато в двоичната бройна система са необходими четири цифри. Поради това, двоичните числа изглеждат хем доста дълги, хем доста еднообразни, което в очите на широката публика им придава особена тайнственост и неразгадаемост. Но както виждате, в тях няма нищо особено. Това са просто числа, които са изразени по начин, подходящ за компютрите, но неудобен за хората. Макар и съвременните компютри да могат да изобразяват всякакви числа, вътрешно те все още съхраняват и обработват числата именно по този двоичен начин. Ето какво ще се получи, ако в двоичен вид изразим нашата готова програма за рисуване на хвърчило:

в своето широко значение, което включва всякакви двоични данни в произволен формат – двоичен код, изходен код, изображения, С-програми, Java-байткод, JPEG, MP3 и т.н. Основанието за това е, че не може да се направи рязко практическо разграничение между програми и данни. Практически, всички програми са данни, а всички данни са програми. Двете са отчленими само мислено и условно в тяхното отношение. Затова тук и двете се наричат с общото име *софтуер*, фиксиращо една практически истинска абстракция.

⁸Конрад Цузе и Джон Атанасов са пионерите, които независимо един от друг откриват колко е подходяща двоичната бройна система.

```

00001110
00000000
00001010
00000000
00000000
00001101
00001010
00110010
00110010
00001100
00001010
00111100
00111100
00001101
00001011
00011110
00000000
00001011
00000000
00011110
00001011
11100010
00000000
00001011
00000000
11100010

```

Изглежда доста тайнствено, нали? Това е така нареченият суров *двоичен код*. Признайте си, щяхте ли от пръв поглед да се досетите какво прави тази програма? Аз определено не бих се досетил. Изобщо никой не би могъл да се справи без справочник и описание на машината, за която е предназначена програмата. Така стигаме до едно важно наблюдение, че всяка програма, написана като директни команди към машината, зависи от типа на същата тази машина. Една програма, написана за един тип машини, ще трябва да се пренапише, за да се изпълнява върху друг. Затова тези инструкции по-често се наричат *машинен език*. В действителност, в началото всички програми са се писали именно на машинен език.

Но трябва също да признаем, че дори записана с обикновени десетични цифри, нашата програма „Хвърчило“ не става кой знае колко по-разбираема. Какво прави командата 14? Не бихте могли да кажете, ако не погледнете в справочника, който дадохме в началото на главата или ако не сте запомнили наизуст всички команди.

Затова е измислен начин, по който да изразяват тези числа със съкратени имена, които са по-лесни за запомняне. За нашата въображаема машина с молив, можем да предложим съответствията, дадени в таблица 2.1.

Таблица 2.1. Съответствия на номерата на командите и техните имена

Команда	Съкратено име	Описание
10	ИДИ	ИДИ до точка с абсолютни координати
11	РЕЛ	иди до точка с РЕЛативни координати
12	ГОР	вдигни молива ГОРе
13	ДОЛ	спусни молива ДОЛу
14	МОЛ	вземи друг МОЛив

Сега според тази таблица можем да изразим програмата „Хвърчило“ така:

МОЛ 0
ИДИ 0, 0
ДОЛ
ИДИ 50, 50
ГОР
ИДИ 60, 60
ДОЛ
РЕЛ 30, 0
РЕЛ 0, 30
РЕЛ -30, 0
РЕЛ 0, -30

Най-напред се вижда, че записът на програмата стана много по-стегнат и кратък. Това, заедно с използването на буквени съкращения, вместо числа, прави програмата по-лесна за четене. На програмистки жаргон се казва, че тази програма е с по-висока „четимост“. Четимостта е важна характеристика на програмата, защото често (или по-скоро редовно) се налага, след като програмата е веднъж написана, някой друг програмист да се върне и да промени програмата. Но преди да я промени, той трябва да я разбере, казано пак на жаргон – да я „прочете“. Понякога това е същият програмист, който я е написал, но понеже е минало време, едно наистина динамично време (през което неговите нагласи и очевидности са се променили), той се отнася към себе си като към друг автор и към собствения си труд като към чужд труд. Продуктът на този труд е неразгадаем. Парадоксален факт е, че в този твърде чест случай програмистът се напъга, за да схване логиката на собственото си произведение. Следователно, чрез четимата програма се спестява голяма част от труда по време на четене, или казано по друг начин – в така записаната програма има повече натрупан труд, отколкото например в програмата, написана в двоичен код. Впрочем, след две седмици пробвайте да се вгледате в двоичния код на програмата „Хвърчило“ и ще почувствате колко е неразбираема. Сякаш някой друг я е съставил. Тогава, ако имате мотивация, отново ще трябва да положите усилие, за да вникнете в кода.

След историческото му възникване, този вид запис се нарича *асемблер* и идва да замести машинния език. Често се нарича и *асемблерен език* като противоположност на машинния. А пък *асемблерен код* е конкретната програма или фрагмент, написани на асемблерен език – т.е. ако асемблерният език е вида, асемблерния код е индивида. Както се вижда, асемблерът е напълно обвързан с машинния език. Всъщност той го повтаря, но в по-удобна за човека форма. Макар да е по-лесен и по-четим, асемблерът съответства на машинния език. Вече казахме, че колкото различни типа машини съществуват, толкова машинни езици има. Сега можем да добавим, че толкова са и асемблерите.

Тъй като машината не разбира нищо от съкращения като МОЛ и ИДИ, необходимо е асемблерния код да се превърне в машинен. Само така машината може да изпълнява командите. Процесът на превръщане на асемблерния код в машинен се нарича *асемблиране* и е автоматизиран, т.е. извършва се от програма. Тя приема асемблерния код и според таблица, подобна на дадената в таб.2.1, произвежда съответния му машинен код. Тази програма се нарича *Асемблер*.⁹ Получава се така, че асемблерните команди са команди към Асемблера, а не директно към машината. Изпълнявайки тези команди, Асемблерът от своя страна произвежда машинен код. За да се изпълнява самият Асемблер, обаче, е необходимо той да е във формата на машинен код. Целият този полукръг е възможен заради онази особена рефлексивност, за която вече стана дума – софтуерът се превръща в свой собствен предмет. Програмата Асемблер е софтуер, който като инструмент служи за създаване на софтуер. Нещо като струг, който служи за създаване на детайли и машини, включително и на други

⁹Когато се пише с малка буква, се има предвид асемблерния език, а когато се пише с главна буква „Асемблер“, се има предвид програмата, която превръща асемблерния код в машинен. Употребен като глагол „асемблиране“, терминът обозначава процеса, извършван от Асемблера. Казано иначе, Асемблерът асемблира асемблера.

стругове. И ако създаването на софтуер е дейност, създаваща стойност, Асемблерът се явява като форма на натрупан труд, като средство на труда.¹⁰ Така той участва в процеса на производство на софтуер.

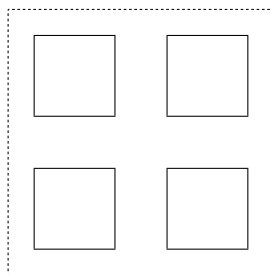
Особеностите са в това, че нашият струг-Асемблер не се амортизира.¹¹ Следователно производството на софтуер е потребителен процес само за „веществените елементи“ – компютри, периферни устройства, консумативи. Тъй като в нашия пример Асемблера не угасва в консумацията си, спрямо него програмистът на асемблер не се явява в истинския смисъл на думата потребител. При все това е абсурд да се смята, че програмирането не е трудов процес, защото все пак се превръщат ресурси – за да създаде продукт, на програмистът като наемен работник му трябва машина (компютър), кафе, бира, и т.н. Машината се амортизира, кафето и бирата са като спомагателен материал за продукта.

Наистина, тук биха могли да възникнат възражения, че съществуват и други продукти, които не угасват в консумацията си. Например един кино филм не се хаби, докато се гледа. Филмът не става по-малко филм, след като зрителите напуснат залата¹². Но все още не е открит начин, по който зрителите да бъдат накарани да гледат един и същ филм до безкрайност. Те могат да го гледат два или повече пъти, но твърде скоро филмът става изцяло „изконсумиран“.¹³ Със сигурност обаче има нещо общо между филма и програмата (второто е софтуер, а първото може да се превърне в софтуер).

Въвеждането на асемблера променя обществено необходимото работно време за производство на софтуер.

2.2.2. Подпрограми и библиотеки

Да си представим, че заредим нов празен лист в нашата въображаема машина и искаме да нарисуваме четири квадрата – нещо подобно, показано на фиг. 2.5. Четирите квадрата всъщност представляват един и същ квадрат, но нарисуван четири пъти на различно местоположение. Тоест, движенията за рисуване на квадрат се повтарят, но от четири различни отправни точки. Тези точки са (10,10), (10,60), (60,10) и (60,60).



Фиг. 2.5. Четири симетрични квадрата

¹⁰ „Когато една потребителна стойност излиза от трудовия процес като продукт, други потребителни стойности, продукти от предишни трудови процеси, влизат в него като средства за производство. Една и съща потребителна стойност, която е продукт на един труд, представлява средство за производство на друг труд.“ ([9] глава V, с.193)

¹¹ „Трудът консумира своите веществени елементи, своя предмет и своите средства, поглъща ги и следователно представлява потребителен процес. [...] Доколкото средствата и предметът на труда сами вече са продукти, трудът поглъща продукти, за да създава продукти, или поглъща продукти като средство за производство на продукти.“ ([9] глава V, с.195)

¹² Ако нещо се хаби при прожекцията, то това са ресурси като електричество, столовете в залата и т.н., хаби се дори лентата на филма. Но лентата е *носителят*, а не самият филм.

¹³ Личният рекорд на автора е 11 пъти гледане на филма „Завръщането на Джедаите“ в невръстна възраст, но дори това е несравнимо с използвания от него Асемблер, или с вероятната честота на използване на Microsoft Word от читателя. Истина е, че специално Microsoft Word и изобщо продуктите на Microsoft изглеждат, като че се амортизират, задръстват се с времето, но за това интересно изключение (потвърждаващо правилото) – по-късно.

Ако разпишем командите за рисуване на изображението, дадено във фиг. 2.5, ще се получи следната програма:

```
МОЛ 0
ИДИ 10,10
ДОЛ
РЕЛ 30, 0
РЕЛ 0, 30
РЕЛ -30,0
РЕЛ 0,-30
ГОР
ИДИ 10,60
ДОЛ
РЕЛ 30, 0
РЕЛ 0, 30
РЕЛ -30,0
РЕЛ 0,-30
ГОР
ИДИ 60,10
ДОЛ
РЕЛ 30, 0
РЕЛ 0, 30
РЕЛ -30,0
РЕЛ 0,-30
ГОР
ИДИ 60,60
ДОЛ
РЕЛ 30, 0
РЕЛ 0, 30
РЕЛ -30,0
РЕЛ 0,-30
```

Програмата стана доста дълга. Освен това, забележимо е как фрагменти от асемблерния код се повтарят, тъй както се повтаря и самото изображение, резултат от изпълнението на тази програма. Заради необходимостта от повтаряне на един и същ код, създаването на по-големи програми става монотонно и неефективно. Вместо да се съсредоточи в спецификата на своята задача, програмистът трябва да се занимава с повторения. Представете си, че той трябва да нарисова не четири квадрата, а шахматна дъска. Кодът на програмата ще стане километричен. Не може ли някак да се избегнат тези повторения?¹⁴ Този въпрос си задава адмирал Грейс Хопър¹⁵.

По време на Втората световна война Армията на Съединените щати финансира проект за създаване на изчислителна машина, предназначена за балистични изчисления. Проектът ангажира редица учени и дава организационна и финансова възможност за реализация на витаещите идеи. Проектът ENIAC¹⁶ се счита за първият компютър и е завършен едва след края на войната (1946 г.). Изобщо след края на Втората световна война се прекратяват много военни проекти, но не и ENIAC. Той така и не успява да изпълни първоначалното си предназначение, но е използван за други военни цели, най-известната от които е разработването на водородната бомба. Федералното правителство запазва интереса си към високоскорост-

¹⁴Тук можем да открием зародиша на ценностите на хакерската култура, в отношението към рутината. На този въпрос ще се спрем по-късно в глава 3.

¹⁵Грейс Хопър е родена на 12 септември 1906 г. в Ню Йорк. На 34 години вече е професор по математика и изведнъж решава да постъпи в армията. В началото не е приета, заради ниското ѝ физическо тегло (47 кг), но тя е упорита и успява да си издейства специално разрешение с което през 1937 г. постъпва във военноморските сили на Съединените щати, където се издига до чин контраадмирал. Тя конструира серията компютри Mark. Умира на 1 януари 1992 г.

¹⁶ENIAC е акроним от Electronic Numerical Integrator And Computer.

ните изчисления, защото те намират голямо приложение във въоражаването. Бидейки вече факт, ролята на ENIAC е много по-голяма и обещаваща, отколкото Армията на САЩ би могла да предвиди. ENIAC бележи началото на цяла една революция. След него следват нови проекти, в които се доразвиват стари и се експериментира с нови идеи.

Случаят с ENIAC е симптоматичен, защото показва как военната сфера се явява важен катализатор на генезиса на изчислителната техника и компютърната индустрия. Затова и не е чудно, че легендарната Грейс Хопър има чин контраадмирал. На нея принадлежат множество важни открития,¹⁷ но тук ще разгледаме само едно от тях. Тя решава да изолира повтарящия се фрагмент код и да му даде име. Така, когато има необходимост от този фрагмент, той се извиква по име, като последователно се изпълняват всички команди от „тялото“ му. Такъв относително самостоятелен фрагмент с тяло и име, адмирал Хопър нарича *подпрограма*. В нашият случай, нека изолираме фрагмента:

```
РЕЛ 30, 0
РЕЛ 0, 30
РЕЛ -30, 0
РЕЛ 0, -30
```

Или с други думи, това е тялото на подпрограмата КВАДРАТ. Можем да използваме тази подпрограма като пренапишем нашата програма за четирите квадрата така:

```
МОЛ 0
ИДИ 10, 10
ДОЛ
КВАДРАТ
ГОР
ИДИ 10, 60
ДОЛ
КВАДРАТ
ГОР
ИДИ 60, 10
ДОЛ
КВАДРАТ
ГОР
ИДИ 60, 60
ДОЛ
КВАДРАТ
```

Както се вижда, разликата в обема на кода е голяма. Но зад чисто текстуалната разлика, съществуват две много важни особености:

1. С концепцията за подпрограмите, натрупването на програмистки труд става още по-ефективно. Ние написахме подпрограмата КВАДРАТ само веднъж, а после я използвахме 4 пъти. Теоретически, можем да я използваме отново още безкрайно много пъти, а практически – колкото ни е необходима.
2. Писането на подпрограми може да се обособи като самостоятелна дейност, като по този начин отваря път пред оптимизирането на организацията и разделението на труда. Един програмист може да е зает с писането на подпрограми, а друг – със сглобяването им в програми.

Подпрограмата е фрагмент код, който се използва многократно. Подпрограмата е нат-

¹⁷Именно на Грейс Хопър дължим термина *bug* (англ. насекомо, буболечка), с който се обозначават компютърните грешки. Тя разказва, че когато компютърът Mark II сгрешил, проблемът бил открит в контактните клеми на едно от релетата, където била попаднала буболечка, променяйки по този начин състоянието на релето и от там – крайния резултат. Почистването от буболечки се превърнало в рутинна операция и се нарекло *debugging*. Този жаргон се превърнал в технически термин, който в българоезичната литература е прието да се превежда като *отстраняване на грешки*.

рупан труд, чиято потребителна стойност може да се измери с броя на използването ѝ. Подпрограма, която изобщо не се използва, не се счита за труд. Тази повишена ефективност се засилва от това, че подпрограмите могат да бъдат вложени една в друга, т.е. – една подпрограма да се възползва от друга или повече други подпрограми. Тогава натрупването на минал труд расте на степен.

Ако често ви се налага да рисувате различни сложни фигури с нашата въображаема машина, сега първо ще отделите известно време, за да съставите няколко подпрограми, които ще влязат в употреба. Ще напишете например **КВАДРАТ**, **КРЪГ**, **ТРИЪГЪЛНИК**, **ПЕТОЛЪЧКА** и т.н., каквото намерите за добре. Такъв проблемно-ориентиран пакет от подпрограми се нарича *библиотека*.

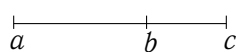
Сетне, когато дойде конкретното задание за рисуване, ще използвате готовите подпрограми, т.е. готовата библиотека, за да сглобите крайното решение. Така крайната задача бива разбита и разделена на подзадачи със съответните си подрешения. Това разделение, което направихме току-що, исторически се случва като разделение на труда. Първо програмистите сами започват да структурират дейността си и да пишат библиотеки. По-късно, в рамките на едно предприятие, екипи от програмисти създават различни библиотеки, а друг екип ги използва, за да произведе крайния продукт. В момента съществуват цели предприятия, чиято дейност е насочена единствено в създаването на библиотеки. Така се постига общо ускоряване на производствения процес. Обособяват се производствени системи и подсистеми, между които възникват специфични отношения. Различните системни части се разделят и изпълняват от различни субекти.

Писането на софтуер без подходящи готови библиотеки се оказва толкова непроизводително, че на всеки компютър започва най-напред да се съхранява и изпълнява някакъв конкретен ансамбъл от основни библиотеки и програми с общо предназначение. Този ансамбъл се нарича *операционна система*. Съществуват различни операционни системи – напр. популярните DOS и Windows за които най-вероятно сте чували, но има и много други – ITS, VMS, Unix и т.н. Различните операционни системи имат различен замисъл и предлагат различни функции. (Впрочем DOS и Windows са толкова бедни в сравнение с Unix, че често се оспорва доколко те могат да бъдат наречени „операционни системи“.)

От гледна точка на софтуера като продукт, асемблерът и библиотеките се явяват като средства за производство. Те са софтуер, чиято единствена цел е създаването на нов софтуер. При производството на софтуер се използва вече готов софтуер, следователно най-голямата цел на този софтуер е неговото използване, отново и отново.

2.2.3. Набиране на височина

Асемблерът показва нещо много важно – че някаква част от самото програмиране може да бъде извършено от програма. Това отваря път към възникването на средствата за разработка – програми, които помагат в създаването на други програми. Пътят на развитие на тези средства е определен от главната цел: автоматизираната работа да бъде колкото се може повече. Ако общото количество труд, необходимо за създаването на един софтуерен продукт представлява отсечката ac (виж фиг. 2.6), то отсечката ab е трудът, актуално необходим на програмиста, а отсечката bc е спестеният, минал труд, кристализирал в средствата за разработка, които той използва.



Фиг. 2.6. Схема на необходимото работно време

Следователно развитието на тези средства води до намаляването на отсечката ab за

сметка на увеличаването на *bc*. Така продуктът бива получен за по-малко работно време, отделено от програмиста. Но от това никак не следва, че програмистът вече работи на по-малък работен ден. Това по-скоро създава възможността софтуерните разработчици, от една страна, постоянно да увеличават принадлежната стойност¹⁸, и да повишават сложността на проектите – от друга. Това е предпоставка за движението от простото към сложното, от решаването на тривиални задачи към решаването на грандиозни (по съставност).

Нито за момент не бива да се забравя, че тези средства от своя страна също са продукти на друг трудов процес, т.е. все пак са изразходвана човешка сила. Как тогава се постига това „спестяване“ на човешката сила? Какво значение има това след като средството на труда по същество представлява консервиран минал труд, или както би запитал Маркс: „Какво значение има обстоятелството, че това работно време е изтекло по-рано и че се намира в отдавна минало време, докато трудът, който е изразходван непосредствено в заключителния процес на преценето, стои по-близо до сегашното време и се намира в минало свършено време?“¹⁹

Наистина, производството на средствата за разработка изразходва определено количество човешка работна сила, но потенциално²⁰ произвежда труд с безкрайна величина. Работата е там, че вече създаденият код може да бъде използван, т.е. потребен отново и отново, и то основно по два начина:

1. Чрез многократното изпълняване на програмата, която може да работи до безкрайност, без при това да се отегчи.
2. Чрез използването на готов код като основа за създаване на друг код.

При това положение, полезната функция на работната сила, т.е. трудът се умножава с всяко следващо потребление. Има някаква ирония в това, че тази клетка човешка сила може едновременно да се превърне в труд с нулева или с безкрайна стойност. Но тази безкрайност е „лоша“ безкрайност. При създаването на средства за разработка със средства за разработка и т.н. се променя драстично кривата на натрупване.

Ето как исторически се случва това. Ако си представим един асемблерен език, насочен към извършване на изчисления, вместо за рисуване, един негов код би изглеждал например така²¹:

ЗАРЕДИ a
ДОБАВИ b
УМНОЖИ c
ЗАПАЗИ d

Можем да запишем този фрагмент така:

$$d = (a + b) * c$$

Както е видно, вторият вид запис е много по-кратък и ясен.²² Би било чудесно математическите операции да се записват по този подходящ за човека начин, вместо на асемблерен език. Така кодът се чете много по-лесно и, следователно, изисква по-малко труд при създаването му (и, което е важно, по-малко труд при промяна). Но за да е възможно машината да изпълнява този код, нужно е той да се превърне в непосредствен машинен код. Процесът на това превръщане се нарича *транслиране*, и, както вече се досещате, се извършва от програма. Тя се нарича *транслатор*.

¹⁸ „Машините са средство за производство на принадлежна стойност.“ ([9] с.385)

¹⁹ „Ако за построяването на една къща е необходимо определено количество труд, напр. 30 работни дни, то общото количество на вложеното в къщата работно време никак не се изменя от това, че 30-ият работен ден е влязъл в производството цели 29 дни след първия работен ден.“ ([9] с.200)

²⁰ Казвам *потенциално*, защото „трудът винаги се разглежда с оглед на неговия полезен ефект“ ([9] с.53). В този смисъл, производството на безполезно средство за разработка не се счита за труд.

²¹ Този пример е взет от ([16] р.188)

²² Тук знакът * означава операцията „умножение“.

Исторически, първият транслятор е разработен от Джим Бакус в лабораториите на ИВМ през периода 1954-1957 г.²³ Транслаторът получава команди и ги превръща в машинен код. Командите, обработвани от транслятора, са подобни на показаните в горния фрагмент и не са нито непосредствени машинни команди, нито асемблерни, а имат собствен синтаксис и семантика, образувайки по този начин самостоятелен език за програмиране *от високо ниво*. Първият такъв език е ФОРТРАН²⁴.

ФОРТРАН език е от високо ниво, защото е вече относително отделен от машинния език, който представлява ниското ниво, т.е. нивото, което е най-близо до машината. Асемблерът беше програма, която преобразуваше една проста текстова форма в нейния непосредствен двоичен еквивалент. От своя страна ФОРТРАН извършва много по-сложно преобразуване и с това предизвиква истинска революция. Програмистката работна сила става много по-производителна.

Но появата на езиците от високо ниво е предопределена от друг фактор – развитието на хардуера. През 50-те години на 20-ти век вече има известно количество различни компютърни системи, създадени след ENIAC – EDSAC, EDVAC, UNIVAC и т.н. През това време компютърът излиза от чисто научната и военната сфера и се промъква в промишлеността. Хардуерът претърпява истинска революция чрез напредъка в електрониката, особено когато по-късно електронните лампи се заменят с транзистори. Това не може да остане без последици за най-близката област – софтуера.²⁵ Компютрите се умножават по вид. Но всеки отделен вид компютър разполага със свой машинен език и една програма, написана на него, трябва да бъде написана отново, за да се използва върху друг компютър. Това възпрепятства натрупването на програмистки труд и катализира противоречие, което трябва да бъде разрешено. ФОРТРАН е рожба на това противоречие между хардуера и софтуера. Изобщо, историята на средствата за програмиране и на софтуера като цяло, е история на преодоляването на пречките пред натрупването на програмистки труд.

Голямото предимство на ФОРТРАН е, че се абстрахира от машината. Програмистът пише код на него, а трансляторът го превръща в машинен код. Ако на всяка машина разполагаме с транслятор, тогава програмата на ФОРТРАН става универсална. С относително малки промени, тя може да се изпълнява навсякъде. Чрез разнообразието от транслятори за различни машини, абстрактният език ФОРТРАН става практически истински. Той става не само възможен, но и *необходим*. Затова писането на програми на този език е много по-лесно, т.е. за еднакво работно време се постига по-голяма производителност на труда, отколкото при асемблерните или машинните езици. Това абстрахиране от машината е именно относително, а не абсолютно. ФОРТРАН все още зависи от конкретната машина, но тази зависимост е значително по-малка, отколкото при езиците от ниско ниво.

Тук трябва да се отбележи, че ФОРТРАН не изпраща в забвение машинните езици и асемблерите. След появата му, те продължават да се използват, използват се и до днес. Трябва да се вземат предвид следните две особености:

- За да се изпълняват програми на ФОРТРАН, трябва да има наличен транслятор. А трансляторът трябва да бъде написан на език, който компютърът „разбира“, т.е. на асемблер или машинен език. Така езиците от ниско ниво все още имат необходимо приложение, доколкото само чрез тях е възможно да се създаде *първото* средство от високо ниво.
- Сферата на приложение на езика от високо ниво е ограничена. Остават специфични задачи, които могат да бъдат разрешени само на ниско ниво.

Затова ФОРТРАН не унищожава езиците от ниско ниво, нито пък има такава задача. Но появата му променя начинът на употреба на тези езици, променя условията на производ-

²³([15] с.59)

²⁴на англ. *FORTRAN* идва от *FORmula TRANslator*, т.е. преводач на формули

²⁵„Превратът в начина на производството в една сфера на промишлеността обуславя също такъв преврат и в други сфери.“ ([9] с.398)

ствения процес, в който те участват. Другото му значение е, че неговата появата изпуска „духа от бутилката“. Неслучайно, много скоро се появяват и други езици от високо ниво – КОБОЛ, ЛИСП и т.н., всеки от които, малко или много, преобръща *начините на производство* на софтуер.

В тази нова епоха, ФОРТРАН и КОБОЛ играят ролята на стана на Жакард. Те си приличат с него по това, че са проблемно-ориентирани. ФОРТРАН е насочен главно към научни изследвания, а КОБОЛ²⁶ е бизнес-ориентиран език. Затова е необходимо да се появят езици с общо предназначение, които да се възползват от новите идеи, но да ги универсализират. В действителност днес съществуват стотици езици за програмиране. Някои от тях са с общо предназначение, а други са разработени за разрешаването на някакъв определен клас задачи.

2.2.4. Производство на отчуждение

Езиците от високо ниво донесат едно качествено ново отношение – разликата между изходен и двоичен код. Изходният код²⁷ е кодът, написан на конкретния език от високо ниво. Например:

```
print "Карл Маркс"
```

Ще изведе името ‘Карл Маркс’ на екрана.²⁸ Транслаторът ще приеме този код, т.н. изходен код и ще го превърне в съответния машинен код. В нашия пример, транслаторът ще генерира:

```
10011001 01001110 00001101 00001010 01001001 01110000 00101010 00111010
01100011 00000000 00000000 00000000 00000000 00000001 00000000 00000000
00000000 01110011 00001111 00000000 00000000 00000000 01111111 00000000
00000000 01111111 00000010 00000000 01100100 00000000 00000000 01000111
01001000 01100100 00000001 00000000 01010011 00101000 00000010 00000000
00000000 00000000 01110011 00001010 00000000 00000000 00000000 11001010
11100000 11110000 11101011 00100000 11001100 11100000 11110000 11101010
11110001 01001110 00101000 00000000 00000000 00000000 00000000 00101000
00000000 00000000 00000000 00000000 01110011 00001000 00000000 00000000
00000000 01101000 01100101 01101100 01101100 01101111 00101110 01110000
01111001 01110011 00000001 00000000 00000000 00000000 00111111 00000010
00000000 01110011 00000000 00000000 00000000 00000000
```

Това е двоичният код, еквивалент на горната едноредова команда. Тук той е подреден в осем колони, вместо в една-единствена, за да се избегне разполагането му в няколко страници. Тъй като ние вече разгледахме двоичния код сам по себе си, тук той ще ни интересува само в отношението си към изходния код.

Изходният и двоичният код си съответстват, доколкото те са само различни форми на един и същ софтуер. Но докато изходният код лесно се превръща в двоичен, обратният процес е много мъчен, в много случаи дори невъзможен. Следователно изходния код съдържа двоичния си еквивалент в себе си, но не и обратно.²⁹

Наистина, двоичният код може да бъде анализиран и в резултат на това да се правят заключения за неговия първообраз. Това е сравнително лесно за някои малки програми или фрагменти от по-големи. Но като цяло, двоичният код не се поддава на манипулация, неговата съпротива е твърде голяма. Често пъти то е толкова голямо, че се оказва по-

²⁶ на англ. *COBOL* е акроним от „COmmon Business-Oriented Language“.

²⁷ английският термин е *source code*.

²⁸ В този пример използваме езика Питон. За повече информация за този език, вж. <http://www.python.org>.

²⁹ Сега говорим най-общо. Наистина съществуват и частни случаи, когато това не е точно така, т.е. има особености.

ефективно изходния код да се състави наново, вместо да се извлича от двоичния.

Да предположим, че се наложи да изведем на екрана 'Фридрих Енгелс', вместо 'Карл Маркс'. Задачата е почти същата и следователно можем да използваме готовия код за основа. Ако погледем изходния код, промяната, която трябва да се направи там, е очевидна. Просто едното име трябва да се замени с другото. Програмата ще изглежда, разбира се, така:

```
print "Фридрих Енгелс"
```

После трансляторът само трябва отново да свърши своята работа и да преобразува този код в двоичен. Така ще получим програма, която ще прави точно това, което искаме.

Но сега да си представим, че разполагаме само с двоичния код. Как бихме могли да го променим, за да работи така както желаем? Поставена по този начин, задачата съвсем не е лесна. Трябва да се положи известно усилие, за да се разбере какво се крие зад тези числа, да се разтълкуват, и от там, ако е възможно, да се променят по подходящ начин, така че да се реши задачата. В действителност, нашият пример е толкова прост, че това може да се отдаде на доста хора. Но при една сложна система, такава промяна на двоичния код клони към невъзможност.

Така софтуерът придобива двойствена природа – във формата на изходен код, той е подчинил двоичния, правейки го само своя функция, и във формата на непосредствен двоичен код, който е застинал, неманипулируем.

Първата форма е сравнително подходяща за промяна, а втората е подходяща само за изпълнение. Първата форма е подходяща за производителя на софтуер, а втората – само за неговия потребител. Разделянето на софтуера като изходен и двоичен код помага за действително отделяне на производителя от потребителя. *Щом производството не съвпада с потреблението, софтуерът е готов да стане стока.*

Макар това разделение (между изходен и двоичен код) да не е единственото условие, то е съществено. Ние видяхме, че софтуерът може да бъде потребяван по много начини – хем със своето непосредствено предназначение, хем като основа за създаване на друг софтуер. Възможностите за потребление на софтуера клонят към безкрайност. Това е крайно неизгодно за производителите на софтуер. (Все едно някой производител на ябълки да произвежда и продава такива особени ябълки, които никога не свършват, никога не могат да бъдат изядени докрай. Ябълки, които винаги могат да бъдат консумирани, без при това да се изчерпват. На каквато и цена да се продава една ябълка, нейната потребителна стойност е несъизмерима. Веднъж снабдили се с една такава ябълка, потребителите никога няма да си я купят отново и на нашият производител не му остава нищо друго, освен сам да си яде останалите ябълки, което би било едно наистина сизифовско начинание.)

Затова производителят трябва да използва всички средства, за да ограничи (начините, а понякога и времето за) потреблението на своя продукт. В този план, разликата между двоичен и изходен код е много важна. Ако производителят разпространява софтуера си само в двоична форма, потребителите ще са силно ограничени да го потребяват с целия му потенциал. По-горе ние използвахме пример с една проста програма, която иска да се придвижи от Маркс към Енгелс. Сега си представете, да речем, една счетоводна програма, която трябва да бъде променена, за да съответства на току-що гласувания в Народното събрание нов данъчен закон. Ако потребителят разполага само с двоичния код на програмата, ще е принуден да се върне при производителя и да си плати отново, при което производителят най-любезно ще му предложи преференциална цена. Но дори и да разполагаше с изходния код, потребителят най-вероятно не е програмист, а счетоводител, и така или иначе нямаше да може да настрои програмата според новите условия. При все това той би могъл да се обърне към произволен програмист, а не да се върне при първоначалния производител, което никак не е в интерес на производителя.

Това е причината, поради която софтуерът започва да участва на пазара (т.е. да се

продава и купува) главно в двоична форма. Тя съответства повече на класическата стокова форма, макар и да не съвпада докрай с нея. Проблемът е, че дори и в тази си форма софтуерът е все още софтуер.

Дори в двоична форма, софтуерът се копира с лекота. При това, нито копие, нито оригиналът са различими. Нищо в оригиналът не говори, че някога е бил използван за копиране. Нищо в копието не говори, че не е оригинал „от първа ръка“. Нищо в копието не наемва дали оригиналът му не е бил, от своя страна, също копие. Следователно за софтуера няма разлика между копия и оригинали. Той *съществува като набор от копия*. Самият конкретен софтуер изчезва с изчезването на последното му копие.

Копията, обаче, винаги са копия върху определен *носител*. Да разгледаме една дискета, например. Когато някой ви даде една дискета, той може с право да каже: «Това е софтуер». Дискетата, обаче, има двойствена природа. От една страна, тя е съвкупността от пластмаса, метал, и други вещества, които я образуват и я правят определен полезен предмет. От друга страна, това е информацията, която дискетата съдържа, променяйки леко своите собствени характеристики.

Носителят не е софтуер в истинския смисъл на думата. Софтуерът е това, което носителът носи. Той е носеното, а не носещото. Носителят може да съществува без да съхранява софтуер. Обратно, софтуерът не може да съществува без да стои върху определен носител. Носителят не притежава основните свойства на софтуера. Не се копира лесно и се „хаби“. Например, когато някой се кани да копира дискета, в действителност той се кани да копира не дискетата, а софтуерът върху нея. При актът на копирането, дискетите не стават две. Напротив, предварителното наличие на две дискети е *условие* за това копиране. Също така, всеки носител има свой собствен *живот* – времето, през което той се изразходва докато загуби свойствата си на носител. За разлика от софтуера, носителът угасва в собствената си консумация.³⁰

Но описаното свойство на софтуера е един вид *разумна абстракция*. В зората на компютъра, когато не е имало разнообразие на носители, софтуерът съвсем не е бил с безкраен живот, дори самият софтуер като такъв само мислено би могъл да се отдели от цялата система. Днес, при една развита компютърна индустрия, когато съществуват много (като видове, и безумно много като брой) компютри, софтуерът наистина няма нищо общо с конкретните си носители и е станал независим. Той се е превърнал в *абстракция практически истинска*. Необходимо е голямо разнообразие на достъпни носители, за да може практически да се фиксира, да кристализира едно качество в софтуера – лекотата, с която се копира, да се размножава без загуба. Необходимо е компютърът да се е разпространил в различни сфери на живота, дори да е влязал в дома, т.е. да е престанал да бъде истински луксозна стока.³¹ Необходимо е общественото производство да достигне до дадено стъпало, за да произведе този кристал като такъв. Иначе той не би бил *възможен*.

От една страна, това му качество е удобно за производителя на софтуер, а от друга страна – смъртоносна заплаха.

³⁰ Не можем да отменим факта, че за съществен често се възприема именно носителът. За типичен можем да вземем нашумелия преди няколко години кино-филм „Мрежата“, където главната героиня през цялото време е преследвана, извършват се машинации, убийства и пр. заради... една дискета. Тая клета дискета е в центъра на цялото действие, но за жалост никому не хрумва, сякаш и преследваните и преследвачите са забравили, че софтуерът върху нея може да бъде размножен върху друг(и) носител(и), под друга форма. Важна е не пластмасата на дискетата, а това, което тя носи, но акцентът е *изместен* върху физическия предмет, върху конкретния конгломерат от пластмаса и метал. Изглежда носителът е важен, от което и следва, че трябва да бъде преследван притежателят на тоя носител. По същество, това е един особен вид *фетишизъм*.

³¹ „Измежду всички стоки същинските луксозни стоки имат най-малко значение за технологическото сравнение на различните производствени епохи.“ ([9] с.192) Луксозната стока преди всичко е достъпна за малцина и е слабо разпространена. Луксът е „излишно потребление“ – нещо, без което може. В луксозната стока не се проявява никаква *обща необходимост*. Тя не е дифузирала в достатъчна степен в обществените отношения, за да бъде техен непосредствен белег. В нашият конкретен случай, компютърът трябва да се разпространи масово, за да е възможна реалността на софтуера – такава, каквато я познаваме днес. Това е двустранен процес.

- То е удобно, защото ако производителят иска да продаде 2,000 копия от своя нов софтуер, той няма да го произвежда 2,000 пъти, а ще го произведе само веднъж и после ще го размножи до 2,000 пъти. Това го поставя в превилигирирана позиция спрямо един производител на автомобили, да речем. Този производител не може да създаде 1 автомобил и да го продаде 2,000 пъти.³² Изобщо, като цяло, това поставя софтуерната индустрия в превилигирирана позиция.
- То е заплаха, защото потребителите имат средствата да разпространяват софтуера без участието на производителя. Когато ми дадете книга на заем, вие се лишавате от възможността да ползвате книгата, вие се отказвате от нейното потребление. Напротив, когато копирате някаква програма, тя си остава при вас, вие не се отказвате от потреблението ѝ, докато в същото време я потребява и някой друг. Така потреблението нараства, а производителят е получил като „еквивалент“ някаква единична, несъизмерима стойност.

Тъй като спецификата на продукта е такава, че потребителят не може да бъде физически принуден да плати двойно за двойна употреба, необходимо е да се въведат някакви суперструктурни регулатори. Правото се притичва на помощ на производителя и започва да му асистира. Това се случва именно поради естеството на продукта. Когато си купувате ябълки от пазара, продавачът не ви ги връчва с тревога в сърцето, дали случайно не сте намислили да ги споделите с приятелите си. Ябълките така или иначе ще се изчерпат в своята мяра, независимо дали ги ядат 20 души или само един. В този смисъл, дали един човек ще си купи 20 ябълки или 20 души ще си купят по една ябълка, е безразлично за продавача. Затова и той не ви кара да подписвате никакви споразумения, че няма да злоупотребите с ябълките, т.е. да действате срещу неговия интерес. Вие просто няма как да го направите, затова и таква регулативна норма е излишна.³³

И тъй, от една страна, заради свойството да се размножава (копира), а от друга, защото даден софтуерен продукт *съществува веднъж, като един*, но върху множество носители, се казва, че софтуерът не се продава в строгия смисъл на думата, а се продават правата за неговото използване при определени условия. Възникват софтуерните лицензи.³⁴ Като правни документи те задължават потребителя да не разпространява софтуера, който е получил. И не само това, но обикновено му се отнема и правото изобщо да поглежда, та камо ли да анализира двоичния му код. Това се счита за престъпление и се преследва като такова.³⁵

Чрез тези лицензи не само се регулират отношенията *производител-потребител* и *потребител-потребител*, но и отношенията *производител-производител*. Затваряйки изходния код, превръщайки го във т.нар. фирмена тайна, отделните производители слагат прегради помежду си, за да не могат взаимно да се възползват от натрупания си труд. Не само софтуерният продукт за крайния потребител е стока, но стока е и библиотеката, която един софтуерен производител лицензира на друг.³⁶ Той трябва да плати нейната стойност и,

³²Производителят на автомобили в известен смисъл създава автомобила си само веднъж, доколкото всички 2,000 автомобила са екземпляри от един и същ модел. Производителят проектира модела си веднъж, но после процесът на получаване на екземпляр (конкретния автомобил) е много сложен и скъп, и тук се крие истинската разлика със софтуера. При все това е интересно да се отбележи, че някаква част от автомобила представлява софтуер – това е не отделния автомобил, а проектът на модела, към който той принадлежи.

³³„Ако тези тривиалности бъдат сведени към тяхното действително съдържание, те говорят повече [...] Именно, че всяка форма на производството създава свои собствени правни отношения, форма на управление и т.н.“ ([10] с.229)

³⁴Една фолклорна поговорка гласи, че в действителност Microsoft са наели повече юристи, отколкото програмисти.

³⁵Един типичен лиценз гласи: „Ограничения върху обратното инженерство, декомпиляцията и дизасемблирането. Не можете да инженерствате по обратен път, декомпилирате, или дизасемблирате СОФТУЕРНИЯ ПРОДУКТ, с изключение и само в степента доколкото такива действия са изрично позволени от действащото право въпреки това ограничение.“ ([19])

³⁶Трябва да се припомни, че *производител* и *потребител* не са субстанции, а отношения. Един производител може да бъде едновременно потребител за друг производител, който от своя страна да бъде потребител за трети

следователно, да я потребява само по определен начин. Всяко допълнително потребление се третира като кражба.

Но за разлика от всяка друга кражба, софтуерната кражба е особена с това, че „ограбеният“ може никога да не разбере, че е ограбен, тъй като откраднатият софтуер не е изчезнал от склада му. Ако не може да открие това по косвен начин, той никога няма да може се възползва от гаранцията, която му дава лиценза. Понеже законът тук е недостатъчен, възниква необходимост от друг регулатор отвъд него. Тази роля изиграва т.нар. *компютърна етика*. Едно от нейните правила гласи: „Потребителите трябва да зачитат авторските права и лицензите на софтуера. Ако хората не си плащат за софтуера, няма да има софтуер.“ ([16] с.99). Тук вече производителят започва да морализаторства. Всички форми на допълнително потребление на софтуера се наричат с общото име „софтуерно пиратство“. Тоест, това е особено разбойничество, това е един аморален акт. Чрез него потребителят изпада от реда на порядъчните хора и става пират – обществено-опасен, нравствено и биографично пропаднал човек. Нарушаването на интереса на производителя води до развала в нравствената цялост на потребителя, която „бавно и сигурно го разяжда“. (Някои лозунги против софтуерното пиратство са твърде близки по съдържание до лозунгите срещу зависимостта от наркотиците, например: „Спрете навреме!“.)

Така чрез описания по-горе комплекс, софтуерът се превръща в стока. Процесът на това превръщане е също и процес на обособяване на потребителите и производителите му. Обособени по този начин, те могат да влязат в почти класически капиталистически, пазарни отношения на размяна. Отношенията на конкуренция могат да започват.

и т.н. Следователно отношението между тях ще бъде отношение на отношения, структура от структури.

Производството на свободния софтуер. История и теория.

Има нещо много вярно в твърденията за обществената опасност на софтуерното пиратство, а именно, че когато софтуерът, вместо да се разменя на пазара, започне да се разпространява свободно, тогава ще се подкопае цялостната структура на отношенията. В този случай, обаче, нито софтуерът като такъв ще изчезне, нито ще изчезнат хората, които го правят, нито светът ще спре, а само сегашният начин на производство ще се промени, и следователно, ще се промени светът.

Отношенията на „споделяне“ на софтуера са възможни, заради самата му специфика. (Много по-лесно е да се споделя софтуер, вместо ябълки.) Тъй като в началото софтуерът се прави от учени, той се обменя по адекватния за тях начин – всичко е публикувано и споделено, достъпно за всеки, който се интересува. В тази комбинация между научната общност и софтуера е зародишът на хакерската субкултура. В личността на хакера се съчетават ролите на програмиста и потребителя. Двете роли съвпадат, защото работещият върху софтуера е пряко заинтересуван, интелектуално възбуден от предмета си. Понеже целта и средството съвпадат, хакерството е една тоталност, не просто едно определение на личността, а *начин на живот*. Ценностите на тази общност гласят: ([13])

1. Светът е пълен с пленителни проблеми.
2. Никой и никога не трябва да решава един проблем два пъти.
3. Скуката и рутината са злини.
4. Свободата е благо.

Тъкмо защото в началото на 70-те години на 20. век потребители в чист вид все още няма, в този исторически момент терминът „хакер“ не съществува, защото не съществува фонът, на който той трябва да изпъкне. Появата на потребители с различно отношение към софтуера като предмет и към свързаните с него дейности, кара компютърните специалисти да се самоосъзнаят като общност и да въведат новия термин, за да обозначат новата реалност.¹ Въпросът е от къде се взимат тези потребители и защо те стават по-многобройни от хакерите?

¹В дефиницията на термина „хакер“ се казва: „Личност, която обича да изследва всяко кътче на програмируемите системи и начините за разширяване на техните способности, в противоположност на повечето потребители, които предпочитат да научат само необходимия минимум.“ ([21] hacker). Тук съм длъжен да направя уговорката, че съществуват две основни значения на този термин. В дадената по-горе дефиниция, както и навсякъде в тази работа, се има предвид оригиналното значение. Но в публичното пространство се е наложило и друго значение – на компютърен злосторник, което не се приема и е обидно в общността на хакерите. Затова е по-правилно злосторниците да се наричат *кракери*. Кракерите нямат нищо общо с хакерите и дори нещо повече – двете определения могат да се разглеждат като противоположности. Защо кракерите публично са познати като „хакери“ е въпрос, който е извън настоящото изследване. Повече информация за това може да се открие в „Как да стана хакер?“ и „Кратка история на Хакерландия“ на Ерик Реймънд [13, 20].

3.1. История на GNU. Ричард Столман.

През втората половина на 70-те години компютърът експанзира и разгръща своето приложение в най-различни области, което е особено маркирано с появата и разпространението на *персоналните* компютри (Apple][, IBM PC, и много други), софтуерът излиза от чисто научната сфера и завладява различни други сфери на дейност. Субектите на тези дейности разглеждат компютъра изобщо, и софтуера в частност, изключително като средство, а не като цел. Счетоводителите и финансистите се интересуват от счетоводство и финанси, и очакват компютъра да им помогне именно в тази тяхна област. Те нямат никакво намерение да стават програмисти и за това имат потребност от готови програми. Възниква обширен пазар за софтуер, който трябва да бъде доволен. Ала това е възможно само ако софтуерът се произвежда като стока. Тази роля не може да бъде поета от университетите и научните институти, затова се създават нови компании, които да организират производството по начин, съответстващ на новата пазарна ситуация.²

На новите компании им е необходим персонал, а той може да бъде набран само от работещите в учебните заведения и научните лаборатории. Пазарният глад дава възможност на компаниите да предложат сравнително високи заплати, и така голяма част от учените се превръщат в наемни работници, които започват да участват в нови за тях отношения. Продуктът на труда им вече не им принадлежи. И те нямат право да го споделят. Настъпва *отчуждение*, което в ретроспекция се преживява като истинска трагедия всред самоосъзналите се хакери ([20]), вече отчетливо изпъкващи на фона на останалите потребители.

Произведеният като стока софтуер залива пазара. Сега в обратната посока, той започва да завладява научната сфера. Понеже софтуерната продуктивност на учебните заведения е отслабнала, те стават клиенти на новите софтуерни компании и пак потребяват продукта на бившите си членове, но този път като участници в коренно различни отношения на двойно отчуждение. От една страна, това е отчуждение от продукта на труда³, а от друга страна, отчуждение от самата производствена дейност, от самия труд⁴. Но отчуждението не е липса на отношение, а специфична форма на отношение.

Точно това преживява и Ричард Столман, програмист-вълшебник в Лабораторията за изкуствен интелект към Масачузетския технологичен институт (AI Labs at MIT). Отказвайки да вземе участие в тези отношения, той напуска опустошения оазис на лабораторията, но не за да постъпи на висока заплата в някоя компания, а за да проповядва и създава свободен софтуер – безплатен, с достъпен изходен код, който всеки може да използва както намери за добре. За Столман, свободният софтуер е въпрос на морал, връщане към ценностите от миналото. Специфичният му консерватизъм не е проява на някаква необяснима ексцентричност, а е *съществува, неделима част* от позицията му. ([24]) Столман пише:

„Разбрах, че златното правило изисква, ако аз харесвам дадена програма,

²Например, създаването на Microsoft през 1975 г. е пряко обвързано с появата на персоналните компютри (Altair, Apple). През първите години на своето съществуване, Microsoft работи изключително за този пазар. Наистина, компютърни компании са съществували и по-рано (напр. гиганти като IBM, Hewlett-Packard, Motorola и т.н.), но техните печалби са предимно от продажби на оборудване, и следователно, споделянето на софтуера не само не нарушава интересите им, а дори донякъде е в съзвучие с тях – колкото повече достъпен софтуер съществува, толкова повече продажби на хардуер могат да се очакват. Напротив, Microsoft е софтуерна компания от чист вид, която разчита на това софтуерът да не се споделя между потребителите. Затова е напълно разбираема появата на отвореното писмо на Бил Гейтс през 1976 г. с което призовава потребителите да плащат за софтуера. ([18]) Това отношение към софтуера не е естествено (в смисъл на природосъобразно, че нещата винаги са стояли така и така трябва да бъдат), а е произведено като резултат от търговския интерес.

³„Произведеният от труда предмет, продукт на труда, се противопоставя на труда като *чуждо съществуване*, като *независима* от производителя *сила*.“ ([7] с.83)

⁴„...работникът се отнася към *продукта на своя труд* като към *чужд* предмет. [...] *Отчуждението* на работника в неговия продукт има не само това значение, че неговият труд се превръща в предмет, превръща се във *външно* съществуване, но че и живее *извън него*, независимо от него, като чужд на него, и става самостоятелна сила против него, че животът, който той е дал на предмета, му се противопоставя – враждебен и чужд.“ ([7] с.84)

тогава трябва да я споделя с другите хора, които я харесват. Не мога без угризеня на съвестта да подпиша споразумение за неразкриване или споразумение за софтуерен лиценз.

Така че за да мога да продължа да използвам компютри без да нарушавам принципите си, реших да натрупам достатъчно количество свободен софтуер, така че да мога да мина без какъвто и да било софтуер, който не е свободен.“ ([23])

Така през 1983 г. се ражда проектът GNU⁵ който има амбициозната цел да създаде цялостна операционна система, съвместима с Unix. Столман основава и организация, която да поддържа проекта – *Free Software Foundation (FSF)*.

В очите на своите колеги, Столман е единственият оцелял, „последният истински хакер“. Една фигура, чиито идеи, изглеждащи като неадекватни на действителността, имат едновременно висока морална стойност и звучат странно, дори налудничаво. Но Столман е непоколебим и през следващите години неговата дейност и авторитет увлича известен брой последователи. Сътрудниците на амбициозния проект GNU работят дълги години в изолация.

3.1.1. Паралелни светове

Преди всичко, Столман дефинира що е „свободен софтуер“. За да бъде свободен, един софтуер трябва да отговаря на четири условия, трябва да ви дава четири *свободи* ([25]):

- Свободата да изпълнявате програмата, с каквато и да е цел (свобода 0).
- Свободата да изучавате как работи програмата, и да я адаптирате за ваши нужди (свобода 1).
- Свободата да разпространявате копия, така че да помагате на ближния си (свобода 2).
- Свободата да подобрявате програмата, и да публикувате вашите подобрения, така че цялата общност да се ползва от тях. (свобода 3).

С една дума – *няма ограничения* върху потреблението. За да е възможно това, необходимо е изходния код да бъде достъпен. Свободният софтуер може да бъде потребяван до безкрайност, във всичките възможни за софтуера форми. По този начин, свободният софтуер се връща, реабилитира първоначалната идея за софтуер.⁶ Но идеята за софтуер и идеята за свободен софтуер са напълно тъждествени само в интелигибленото небе.

На практика, за да може свободният софтуер на Столман да оцелее, трябва по някакъв начин да се гарантира, че той *винаги* ще остане свободен. (Затова Столман ще каже, че «истинският свободен софтуер винаги остава свободен».) Той трябва да бъде защитен от възможността да бъде превърнат в несвободен.

Но субстанцията на свободата на софтуера не се съдържа в самия софтуер като такъв, тъй както в една вещь не се спотайва фактът, че тя е нечия собственост. *Свободата на софтуера се крие в отношенията, в които този софтуер кристализира.*⁷ Следователно, гаранцията за свободата трябва да бъде дадена със средствата на самата обществена форма. Но как от една страна, свободният софтуер ще се впише в тази форма, а от друга, не трябва да участва в традиционните отношения, които ще го превърнат в несвободен?

Столман решава тази гатанка като се обръща към концепцията за авторското право. Той проектира универсален лиценз, един напълно сериозен и правно издържан документ, който регулира условията за копиране, разпространение и модифициране на защитения с него

⁵GNU е акроним от GNU's Not Unix, т.е. това е едно безкрайно рекурсивно отрицателно определение.

⁶Софтуерът *изобищо* се копира лесно и без загуба, но това му качество е „забранено“ при софтуера, който се произвежда като стока.

⁷„Всяко производство е присвояване на природата от страна на индивида *в рамките на и посредством определена обществена форма.*“ ([10] с.230 (курсивът е мой – К.Д.))

свободен софтуер. Но по съдържание, този лиценз е напълно противоположен на своите събратя. Така на бял свят се появява GNU General Public License (GPL) ([17]), чийто дух може да бъде съгъстен в два от абзаците на преамбюла му:

Нашият Общ Публичен Лиценз е направен така, че да Ви осигури свободата да разпространявате копия на свободен софтуер...; да Ви предостави изходните текстове на програмите или възможността да ги получите, ако искате; да Ви позволи да промените софтуера или да използвате части от него за създаването на нови свободни програми; и да сте знаете че можете да правите всичко това.

За да защитим Вашите права, се налага да поставим ограничения, които забраняват на който и да е да Ви откаже тези права или да Ви помоли да се откажете от тях. Тези ограничения водят и до определени отговорности за Вас, ако разпространявате копия на свободен софтуер, или ако го промените.⁸

Столман прилага GNU GPL към своите програми и кани всички програмисти да го използват за техните. Така свободният софтуер бива защитен с лиценз, чиито клаузи не позволяват той да влезе в „несвободен“ производствен процес. Парадоксът се състои в това, че свободата на софтуера се гарантира по начин, призван да изпълни точно обратната мисия. Но това е необходим парадокс. Столман виртуозно го илюстрира с неологизма „copyleft“ – като противоположност на „copyright“.

Строго казано, термините „свободен софтуер“, „copyleft“ и „софтуер, защитен от GNU GPL“ не са напълно тъждествени. Те по-скоро се отнасят както общото към частното. Всеки GNU GPL софтуер е свободен софтуер, но не всеки свободен софтуер е защитен от GNU GPL. Тяхното точно съдържание и отношение е дадено в „Категории свободен и несвободен софтуер“ [22].

В GNU GPL е заложена още една много важна концепция – че в полето на софтуера изобщо, свободният софтуер трябва да нараства. Ако някой иска да използва такава свободна програма като основа за друга, или само да използва малка част от свободната програма в друга, то другата програма задължително попада под условията на GPL. Примамливо е да използваш готова програма като предмет за ново производство, но условието за това е да дариш труда си по същия начин, по който си получил този предмет. Това от една страна стимулира превръщането на много софтуер в свободен, а от друга страна кара много компании и частни програмисти да се отдръпнат, стараяйки се за съхранят своето за себе си, лишавайки се от това, което им предлага свободния софтуер. Но това отдръпване не е загуба за движението за свободен софтуер, тъй като те и без това не биха допринесли с нищо към него.

Така свободният софтуер стъпва здраво на краката си и постепенно тръгва нагоре по една крива, която в началото е спокойна и равномерна, но с течение на времето става все по-стръмна и рязка.

Границата, която GPL полага, е граница между два свята – светът на свободния софтуер и светът на софтуера, защитен чрез традиционното лицензиране. Те се намират в относителна изолация. Двата свята съществуват паралелно в двояк смисъл. От една страна, съществуват по едно и също време, като всеки създава своя основа и механизми за натрупване. Като исторически появил се по-късно, светът на свободния софтуер е в позицията на догонващ в натрупването. Но тъкмо заради механизмите на споделяне, които елиминират дублирането на труда, свободния софтуер се развива с много високо ускорение. От друга страна, с изключение на фигури като Столман, участниците в света на свободния софтуер принадлежат и на другия свят. Те са главно наемни работници в софтуерните компании, а през *свободното си време* създават свободен софтуер. Това, което е логически паралелно, се явява като паралелно в биографичен план. Тези програмисти участват в два различни

⁸цитирано е с малки корекции по превода на Цвятко Йовчев.

вида производствени отношения, в различни видове общуване.⁹

Първоначално, проектът GNU има за цел да създаде цялостна завършена операционна система. Но Столман не се захваща отведнъж с тази задача. Напротив, като опитен програмист, той започва да създава различни средства за разработка – текстов редактор, компилатор¹⁰ (GCC), различни полезни библиотеки и десетки други компоненти. Това е фундаментът на едно ново натрупване, което трябва да се извърши наново, „начисто“, за да не може никой да предяви претенции към свободния софтуер. Разполагайки със солидни свободни основи, количеството свободно-достъпни средства за разработка расте, а от там расте и свободният софтуер като цяло. С това Столман увлича все повече сътрудници около себе си. И така кръгът на натрупване се затваря.

По-късно Линус Торвалдс, за който ще стане дума в следващата подглава, ще признае: „Значението на компилаторите бе една от причините да реша да лицензирам Linux под GNU GPL. GPL е лиценз за компилатора GCC. Мисля, че всички други проекти на групата GNU изглеждат незначителни за Linux, в сравнение с GCC. Той е това, което истински ме интересува.“ ([26])¹¹

3.2. Катедралата и базарът

При положение, че проектът GNU работи в относителна изолация (до края на 80-те той се радва само на частична поддръжка от малко на брой софтуерни и хардуерни компании), интересен става въпросът, защо движението за свободен софтуер придобива такава актуалност, каквато има днес. В края на краищата, проектът GNU можеше никога да не изплува на повърхността на публичния живот. Сътрудниците му можеха и досега да човъркат из своите програми, без дори да сме чували за тях. Съществуват някакви причини *отвъд* свободния софтуер като такъв, които определят неговата *необходимост*. Има нещо отвъд морала на Столман – един морал, който далеч не се споделя от всички програмисти, които по някакъв начин сътрудничат на проекта в началото на 90-те години.

Хакерската субкултура като цяло и моралът на Столман в частност бяха абсолютно необходими за генезиса на свободата като принцип на производство – и като действителна идея, и като действителен резултат във формата на конкретен софтуер. Но само с тях не може да се обясни мощния водовъртеж, който се образува цяло десетилетие по-късно и обхвана цялата индустрия. Едва ли има останало кътче от нея, което вече да не е засегнато по един или друг начин от свободния софтуер.

През годините, проектът GNU успява да натрупа голямо количество софтуер, но към цялото остава да липсва една съществена част – един критичен компонент от системата,

⁹Това подсказва и как финансово се издържа цялото начинание. В своята иначе чудесна книга ([2]), г-н Бабалиевски задава въпроса: „Е, кой им плаща на тези хора?“ – и достига до прозрението: „Ами бащите им...“ Той е прав, че финансирането на това начинание е проблем. Греша обаче в това, че то е резултат от някаква външна сила. Роднинската зимопомощ е силно валиден принцип в България, но не и в развития атомизиран капиталистически свят. Работещите програмисти в бизнес сектора сами даряват труда си, програмистите от академичната сфера използват свободния софтуер като поле на своите изследвания, експерименти и за образователни цели, а Фондацията за свободен софтуер е създадена, за да могат да се събират дарения в пари и в компютри от други организации (били те търговски или нетърговски) и частни лица. Списъкът на дарителите е публично достъпен на адрес: <http://www.gnu.org/thankgnus/thankgnus.html>. Фондацията също продава печатна документация за своя свободен софтуер (самата документация е свободна, но доколкото е върху печатен носител, за нея се взимат пари) и сувенири. В началото известна роля изиграва и неразвитостта на Интернет. Тогава не всеки желаещ е можел да изтегли софтуера по мрежата. В такъв случай интересуваният се е трябвало да плати за външен носител и Фондацията е можела да вземе такса срещу самата услуга по копирането върху носителя.

¹⁰Компилаторът е вид транслятор. Само информативно можем да споменем, че всъщност трансляторите, за които говорихме вече в глава 2, най-общо се делят на интерпретатори и компилатори. Тук няма нужда да навлизаме в по-големи технически подробности.

¹¹Преводът е на Йовко Ламбрев.

наречен „ядро“.¹² Това е задача с изключителна сложност, с която като че ли проектът GNU няма ресурс да се справи. Екипът по-скоро се надява да получи правото да използва някое от наличните по това време готови ядра. След дълги години на преговори и планове по този въпрос, през 1993-94 започва да назрява един абсурд.

Един никому неизвестен студент от Университета в Хелзинки, Финландия, с помощта на стотици (а по-късно хиляди) сътрудници от цял свят, създава солидно ядро. Това е Линус Торвалдс, а ядрото е наречено на негово име – Linux. Сътрудниците му не са под купола на никаква формална организация, нито проектът се финансира целенасочено. Linux е свободно достъпен софтуер под клаузите на GNU GPL.

Въпросът „Как е възможен Linux?“ се явява с голяма острота пред сътрудниците на GNU. Затова не е изненада, че един от тях, Ерик Реймънд, пръв формулира този проблем в своята студия „Катедралата и базарът“:

„Linux е подмолен. Кой би си помислил само преди пет години (1991), че от хакерските занимания в свободното време на няколко хиляди разработчици, пръснати из планетата, и свързани единствено от тънките нишки на Интернет, като на магия ще се получи операционна система от световен клас?

Определено не и аз. По времето, когато Linux се появи на радарата ми в началото на 1993, аз се занимавах с Unix и разработка на софтуер с отворен код вече 10 години. По средата на 80-те години на 20-ти век бях един от първите сътрудници на GNU. Бях пуснал доста свободен софтуер в мрежата, разработвайки и съразработвайки няколко програми [...], които и днес все още са широко употребявани. Смятах, че зная как стават тези неща.

Linux преобърна голяма част от това, което си мислех, че зная. От години проповядвах Unix-евангелието на малките инструменти, бързото създаване на прототипи и еволюционното програмиране. Вярвах обаче, че има определена критична сложност, отвъд която е необходим по-централизиран, априорен подход. Вярвах, че най-важният софтуер (операционните системи и наистина големите инструменти [...]) трябва да бъдат съградени като катедрали – внимателно майсторени от отделни вълшебници или малки групички магове, работещи в уютна изолация...

Стилът на разработка на Линус Торвалдс – „пускай рано и често; възлагай на други желаещи всичко, което можеш; бъди открит до степен на безразборност“, ми дойде изненадващо. Нямахме го тихото, благоговейно катедрално разработване – Linux-обществото по-скоро ми приличаше на огромен шумен базар с различни планове и подходи (доста точно уподобени от местата с Linux архиви, които приемат софтуер от *всеки*), от който само по чудо би могла да се роди съгласувана и стабилна операционна система.

Фактът, че този базар като че ли работеше, при това добре, ми дойде като гръм от ясно небе. Докато работех усилено по индивидуални проекти, а освен това се опитвах да разбера защо Linux-светът не само че не се разпада, но и като че ли става все по-силен и по-силен, при това със скорост, за която „катедралните строители“ едва ли биха могли да мечтаят.“ ([14] с.2)

Наистина, абсурдът, чудесно описан от Ерик Реймънд, е пълен. Но Реймънд има теоретична нагласа. Неговата студия няма за цел просто да посочва абсурда, а да го *обясни*. Тази задача се разрешава по виртуозен начин. Изобщо, формулирането на всеки новоизпъкнал абсурд вече само по себе си е постижение. А обяснението му е възможно само при разместването на цели пластове, при откриването на нови хоризонти.

¹²За целта на настоящото изложение не е необходимо да се изяснява какво точно представлява ядрото. Тук е достатъчно само да знаем, че това е един изключително сложен компонент, който се проектира и реализира много трудно.

Евристичността на подхода на Реймънд се крие още в началните предпоставки. От самото начало Реймънд строи два противоположни идеални модела, които той нарича *катедрален* и *базарен*. Новото е, че не се противопоставя свободния софтуер на несвободния, а се извършва анализ в рамките на самия свободен софтуер. Това не е противоречието между GNU и, например, Microsoft, а е противоречието между GNU и Linux, персонализирано съответно в Столман и Торвалдс. Резултатите от този анализ излизат извън пределите на свободния софтуер и се прилагат върху производството на софтуер изобщо. В този план, традиционният модел на работа в проекта GNU изглежда по-близо, попада в една категория с производството на несвободен софтуер.

Но „Катедралата и базарът“ е шедьовър в два плана: едновременно по форма и по съдържание. Като форма цялото произведение е построено като история на един личен експеримент на автора. Там изводите, до които той достига, едновременно с това се проверяват от опита, водят до действителни резултати. Теорията и практиката са обвързани органично. Тъкмо затова след първото си публикуване „Катедралата и базарът“ има огромно влияние върху индустрията, а авторът става знаменитост. „Катедралата и базарът“ е *съзнанието* за Linux. Двете образуват обща суперструктура, която днес играе ролята на авангард.

Преодолявайки традиционния фетишизъм, Реймънд измества фокуса от продукта към процеса, който произвежда този продукт. Въпросът тогава е: *как* се произвежда Linux?

1. Премахнато е разделението между потребители и разработчици. Всеки, който използва продукта, може да допринесе за подобряването му, като по този начин участва в процеса не само като крайно звено. Започнал като индивидуален проект през 1991, Linux постепенно увлича множество хакери-доброволци. Няма формална процедура по „кандидатстване“ за членство към проекта. Всеки, който подобри някакъв фрагмент от цялото, е поканен да го изпрати на Линус Торвалдс, който от своя страна взема решение дали да го включи в общия код. И не само това. Всеки, който открие проблем в продукта, го докладва. Ранният доклад за един проблем е първата предпоставка за бързото му отстраняване.¹³
2. Ускорен е цикъла на публикуване/тестване/подобряване. Със своята стратегия „пускай рано и често“, Линус Торвалдс бързо обнародва получените поправки и нововъведения, така че новите усилия и доклади за грешки не се дублират. От друга страна това стимулира участниците, тъй като техният труд мигновено застива в продукта.¹⁴
3. Трите класически за бранша процеси на проектиране, разработване и отстраняване на грешки съвпадат.¹⁵

Разликата между катедралния и базарния модел се състои във *формата на общуване*. Традиционните *роли* на участниците в това общуване *съвпадат* като по този начин се превръщат в излишни категории. Щом *потребител* = *съразработчик* = *изпитател*, защо

¹³ „8. Ако има достатъчно голяма база от бета-изпитатели и съразработчици, почти всеки проблем ще бъде характеризирен бързо, а поправянето му ще бъде очевидно за някого.

Или казано не така формално – Ако има достатъчно очи, всички грешки се виждат. Аз го наричам *Законът на Линус*.

Моята първоначална формулировка беше, че всеки проблем ще е „очевиден за някого“. Линус обаче смяташе, че този, който разбере и оправи проблема, не е непременно, нито обикновено същият човек, който го характеризира. «Някой вижда проблема», казва той, «а някой *друг* го разбира. Продължавам да твърдя, че да го откриеш е по-голямото предизвикателство». Важното е обаче, че и двете неща се случват бързо.“ ([14] с.6)

¹⁴ „В онези ранни времена (около 1991) нерядко му се случваше да пуска ново ядро повече от веднъж на ден! [...] Линус постоянно стимулираше и възнаграждаше своите хакери/потребители – стимулираше ги възможността да свършат някаква част от работата, задоволяваща егото им, а ги възнаграждаше гледката на постоянно (дори *всекидневно*) подобряване на работата им.“ ([14] с.6)

¹⁵ „Успоредно може да бъде не само отстраняването на грешки. Разработката и (може би в удивителна степен) изследването на пространството на дизайна също могат. Когато начинът ви на разработка е рязко итеративен, разработката и подобряването на софтуера могат да се превърнат в частен случай на отстраняването на грешки, т.е. поправяне на „грешки поради недостиг“ в първоначалните възможности или в концепцията за софтуера.“ ([14] с.10)

да правим това чисто логическо разделение? Абстракцията *потребител* вече е престанала да бъде практически истинска и се е завърнала в своята разумна форма. Същото се отнася и за трите процеса, които се сливат в един: *проектиране* = *разработване* = *остраняване на грешки*. В „класическото“ комерсиално разработване на софтуер, трите процеса са не само мислено отчлени, но са и практически отделени. В една типична компания като Microsoft, например, групата разработваща един продукт е различна от групата, която го тества. Тази разлика особено изпъква в отношението към грешките.¹⁶ Ерик Реймънд пише:

„Тук според мен се намира основната разлика между катедралния и базарния стил. В „катедралните“ възгледи за програмирането грешките и проблемите при разработка са тайнствени, коварни, дълбоки феномени. На малцината посветени са необходими месеци усърдна работа, докато се добие увереност, че всичко е чисто. Оттам идват и дългите интервали между изданията и неизбежните разочарования, когато дългоочакваните издания не са съвършени.

При базарния възглед за нещата, от друга страна, се приема, че грешките общо взето са просто явление, или поне се оказват прости, когато бъдат изложени пред хилядите нетърпеливи съразработчици, които се нахвърлят на всяко ново издание. Съответно се правят по-чести издания, за да има повече поправки, като една от добрите страни на това е, че имаш да губиш по-малко, ако някоя грешка случайно успее да се промъкне незабелязана.“ ([14] с.6)

Реймънд е напълно прав, като заключава: „Всъщност най-умното и значително нещо, което Линус направи, беше не толкова конструирането на самото ядро Linux, колкото изнамирането на модела за разработка на Linux.“ ([14] с.5) Реймънд показва, че този производствен процес не се движи от ирационално ентузиазирани щастливци, разтворени в своята общност. Напротив, всеки участник има *интерес* в проекта. Ентузиазмът идва по-късно, той първоначално е следствие, което след това *катализира* процеса. Така Реймънд извежда три задължителни условия, за да се движи един проект в базарен стил:

1. Обещаваща начална основа.¹⁷ Линус Торвалдс използва кода на Minix и дизайнът на Unix за основа.
2. Координаторът на проекта трябва „да може да разпознава добрите идеи в дизайна от други хора.“ ([14] с.13)

¹⁶Стив Магуайър, бивш програмист и тествач служител на Microsoft, чудесно описва как стоят нещата в неговия свят: „Колко пъти сте чули програмисти да казват: «Надявам се изпитателите да са открили всички грешки», точно преди програмният продукт да бъде поставен в кутията, опакован и изпратен на търговците? Те просто кръстосват ръце и се надяват на най-доброто. [...]»

Ако насочите камерата на шпионски сателит към една типична софтуерна къща, ще видите програмистите в нея приведени над клавиатурите в преследване на докладвани грешки. В друг случай, ще откриете изпитателите да атакуват най-новата версия за вътрешна употреба, бомбардирайки я с входни данни и очаквайки да намерят някоя и друга грешка. Може дори да видите някой от тях да проверява дали някоя от старите грешки не се е промъкнала отново незабелязано в програмата. Ако смятате, че търсенето на грешки по този начин отнема огромни усилия... вие сте прави. Този начин на търсене предполага също и голям късмет.

Късмет?

Да, късмет. Когато някой от изпитателите открие грешка, това не е ли, защото той е случил да забележи, че някое число не е правилно, или че някоя част от програмата не се държи така, както се очаква от нея, или че програмата е забила? [...]

Може да звучи страшничко, но изпитателите тъпчат програмите с входни данни и се надяват, че спотайващите се грешки сами ще се покажат отнякъде... стратегията е „удряй и наблюдавай“.

Не ме разбирайте погрешно. Аз не казвам, че това което правят изпитателите е неправилно. Аз твърдя, че е трудно да се тества програма, гледайки на нея като на черна кутия, защото всичко, което може да направи един изпитател, е да напъха данни в програмата и да гледа какво изскача от нея на изхода. Това е все едно да се опитате да определите дали един човек е ненормален. Питате го някакви въпроси, чувате отговорите му и правите преценка. Накрая, никога не сте напълно сигурни, защото не знаете какви са мислите на другия. Вие винаги се питате: «Дали му зададох достатъчно въпроси? Това ли бяха правилните въпроси?» ([6] с.19,30-31)

¹⁷„Вашата зараждаща се общност от разработчици има нужда от нещо, което може да се стартира и да се изпитва; нещо, с което да може да си играе.“ ([14] с.13)

3. От решаващо значение са считаните досега за „несъществени“ лични качества на водача.¹⁸

Реймънд посочва един важен социален фактор – развитият Интернет. На Линус Торвалдс му е необходима глобалната мрежа, за да използва ресурсите на заинтересовани хора по целия свят. Следователно развитият Интернет е условие за базарния стил и историята на неговото развитие ще бъде съответно предистория на базарния стил. Самата история на Интернет е умален модел на историята на цялата компютърна индустрия. Тук ние само бегло ще я маркираме.

Началото на Интернет е поставено през 1969 от един военен проект на DARPA¹⁹, наречен DARPA NET. Проектът има за цел да построи военна мрежа, която да устои на евентуална ядрена атака. За целта мрежата трябва да бъде децентрализирана, така че при загуба на някой възел, информацията да преминава по заобиколни пътища. ([3] с.14)

ARPANET свързва различни организации, ангажирани с отбраната на САЩ, и постепенно се разпространява из научните институти и университетите. Тя позволява провеждането на съвместни експерименти и научен обмен. Но по естеството си общуването между учените не признава националните граници. Така към мрежата се свързват научни институти и изследователски центрове от Европа и други континенти, образувайки по този начин новата глобална мрежа, наречена Интернет (International Network). Така мрежата се разраства и обхваща целия свят, но остава в рамките на научната общност. Учените интензивно обменят електронна поща и документи, но мрежата е факт само за техния свят.

Интернет започва да изплува на повърхността на публичния живот, когато през 1989 Тим Бърнърс-Лий създава World Wide Web. Неговата първоначална идея е да предостави по-удобен начин за обмен на научни документи. Но създадената от него техническа основа не е ограничена само до научната област, а е универсална. Местата в Интернет бързо започват да преобразуват натрупаната информация, публикувайки я в Web. Това стимулира много любители да търсят начини за включване в мрежата, мигрирайки от BBS-системите към Интернет. Това създава пазар за Интернет-услуги и определя появата Интернет-доставчици за масовия потребител. Но с растежа на участниците в мрежата расте и публикуваната информация, растат ресурсите, предлагани от нея. Мрежата става все по-привлекателна и така процесът се усилва и завихря по своя собствена логика.

Новонатрупаната наличност на разнородна маса в едно ново пространство привлича вниманието на търговските компании. През 1994 г. те постепенно започват да усещат потенциала на Web.²⁰ Отваря се пътят на бизнеса и така всички навлизат в това пространство.

И тъй, в основата си Интернет стартира като военен проект, после се поема и се усилва от научната сфера, и накрая се масовизира, дифузира във всички сфери на дейност. Интернет става публичен факт, макар и около четвърт век след своето зараждане. (Понеже Web е причината за това, днес много неспециалисти отъждествяват Интернет с Web.)

Развитият Интернет е резултат от една дълга еволюция. Но характерът на новите отношения, които носи, е революционен. Мрежата позволява на хората да общуват, без да се виждат и без да чуват гласа си. Това важи и за конвенционалната поща, но в много по-малка степен. Навремето, конвенционалните пощенски услуги също са резултат на и условие за големи социални промени. Глобалната мрежа е новият начин за общуване, ускорен

¹⁸„За да се създаде общност от разработчици, трябва да привлечаш хора, да ги заинтригуваш с това, което правиш, и да бъдат доволни от количеството работа, която вършат. [...]“

Неслучайно Линус е готин пич, който кара хората да го харесват и да искат да му помогнат. Неслучайно аз съм енергичен екстровеърт, който обича да работи с много хора, и притежава някои от характерните черти и инстинктите на неизявен комик. За да сработи базарният модел, от огромна помощ би било, ако можете поне мъничко да очаровате хората.“ ([14] с.14)

¹⁹DARPA е акроним от *Defense Advanced Research Projects Agency* – това е централната организация за изследвания и развой на Министерството на отбраната на САЩ.

²⁰Тази технология, за разлика от другите Интернет-технологии каквато е електронната поща, позволява на търговските компании да атакуват сетивата на потребителя).

и обобщен до немислими досега граници. Чрез конвенционалната поща никой не би могъл да проведе диалог с друг човек от другата страна на света, без при това да са необходими месеци.

Участвайки в електронна дискусия в мрежата, никой не вижда расовата принадлежност на събеседника си, неговия пол, цвят на очите, външен чар и т.н. Представата за другия се изгражда изключително на основата на неговите мисли и идеи. Човек се явява като човек изобщо – абстрахиран от раса, пол, възраст и т.н., той се превръща в *абстракция практически истинска*, която обаче ражда безразличие не в смисъл на индиферентност ([8] с.97).

Изключение прави единствено езикът. В електронната дискусия, мислите замръзват в текст. Текстовете, които човек създава, се явяват неговото истинско лице. Това обстоятелство произвежда нерелевантната хакерска ценност „Научи се да пишеш добре на родния си език. Не изпращай писма или електронна поща с правописни грешки.“²¹

3.2.1. Парадоксални отношения

Истинската заслуга на Реймънд е, че поставя въпроса за *социалната връзка*, която сплотява общностите на свободния софтуер. Но на него му е трудно да я характеризира, поради нейната парадоксалност. От една страна, той нарича това „програмиране без его“, но от друга страна посочва, че хакерите са силни индивидуалисти и всъщност репутацията е тази, която стимулира личната изява в такава общност. Проблемът за „програмирането без его“ не е чисто терминологичен, както той се опитва да го извърти. Наистина не може да се каже, че един хакер е „без его“, това е противоречие в определението. Самата му същност е дълбоко засегната, а не само думата. В крайна сметка Реймънд нарича това явление „култура от самоорганизиращи се егоисти“, което според него не е нищо друго, освен един *свободен пазар*. Това ни препраща към самото заглавие на произведението „Катедралата и базарът“.

Можем ли изобщо да наречем „пазарни отношения“, такива отношения, които не се основават на стокова размяна? Отношения, където няма продавач и купувач, където всеки допринася доколкото може, според собствената си мяра, а получава целия продукт при това с правото да го използва колкото му е необходимо? Всеки дава по малко, а получава всичко. (И понеже разполага с всичко, той използва само колкото му е необходимо.) Производството е общо, присвояването – също. Нещо повече, на никой от разработчиците на Linux не му е хрумвало да възроптае, че с нарастването на популярността му, Linux започва да се използва и от хора, които като че ли нямат пряк принос в разработването му.²²

В началото всички роли са се концентрирали в едно лице. Компютърният специалист е създавал хардуер, пишел е програми за този хардуер, предназначени за самия него като специалист или за други, подобни на него специалисти. Всички тези дейности са една и съща негова дейност.²³ С развитието на сферата, ролите се разпръскват като отделни и самостоятелни. Първо се разделя хардуериста от програмиста. После разработчика от потребителя, и т.н. Между тях, между тези нови субекти, възникват опосредени отношения. Възниква *отчуждение*. Производството на софтуер като стока, и не само като стока, но и

²¹Ерик Реймънд дори се изненадва: „Удивително множество хакери, включително всички познати ми хакери са качествени писатели.“ ([13])

²²Първо, всеки един *чист потребител* на Linux е потенциален съразработчик. Дори ако открие даден проблем или недостатък на продукта, това е принос. Второ, тук общността на Linux излиза извън собствените граници на самия Linux. Приложните програми, имайки свой цикъл на живот, създават необходимост от употребата на Linux, което означава че той се тества все повече и повече и следователно те подобряват самия Linux, макар той да се явява тяхно условие. Така приложните програми и ядрото се обуславят взаимно.

²³Например легендарният Сеймор Крей, дизайнер на линията суперкомпютри Cгау, проектира и въвежда своя собствена операционна система в проектирания от самия него компютър [20], или пък друг пример – Стив Возниак (често наричан „магьосникът от Воз“) съвсем сам проектира компютрите Apple I, Apple II, цял куп периферни устройства за тях, малка операционна система и транслятор [27].

като масова стока (т.н. не луксозна, а широко разпространена), предполага трудовия процес да се извършва в развити организации, където търговската администрация е отделена от програмистите. А администрацията, като относително самостоятелна, има твърде малко общо с програмирането като *специфичен* трудов процес.²⁴ И тъй като това производство е производство *за пазара*, връзката не е директна *потребител-програмист*, а минава през продуцента и търговеца. А ние вече знаем, че връзката *потребител-търговец* също не е лична, а опосредена.

Свободният софтуер разчупва тази схема и отново събира всичко в една точка. Какъв е Линус Торвалдс за Linux – менажер, проектант, набиращ персонал служител в „Личен състав“, говорител или програмист? Всичко това заедно. Това е възможно само защото в същността си производството на свободен софтуер не е стокотранспортно, не е производство за пазара, а за потреблението.²⁵ Да вземем някое типично производство за пазара. Например е очевидно, че Coca-Cola не произвеждат безалкохолни напитки, защото са жадни. Нито пък Microsoft произвеждат Windows, само защото не могат да намерят операционна система за компютрите си. Разликата между Microsoft (като софтуерна компания) и Coca-Cola е, че Microsoft реално използват Windows (както и C/C++ компилаторите си) като средство за производството на други свои продукти. Голямата разлика между Линус Торвалдс от една страна, и Microsoft и Coca-Cola от друга, е, че Линус е започнал да работи по ядрото за свое собствено потребление. Без сам да знае това (дори без да го желае), Ерик Реймънд доказва това в „Катедралата и базарът“, доставяйки материал, който би трябвало да доведе до противоположни на неговите изводи. Там хаотично, макар и по несъзнателен начин се посочва:

1. Свободният софтуер не възниква с цел печалба, а с цел потребление.²⁶
2. Развитието на един свободен софтуер е телеологически определено от изискванията на неговото потребление.²⁷
3. Задоволи ли се това потребление, производството на конкретния свободен софтуер става излишно. Той е „готов“.²⁸

Накратко, принципът на възникване, траене и закриване на един свободен продукт, не е собствено капиталистически. Капитализмът е уязвен в сърцевината си. На негова територия, в най-развитите сфери са възникнали отношения, които противоречат на неговите предпоставки.

²⁴Така например през 1984 г. изпълнителен директор и президент на Apple Computer, Inc. става Джон Скъли, който преди това е бил вицепрезидент в PEPSI CO.

²⁵„В стокотранспортното производство потребителните стойности се произвеждат изобщо само затова и дотолкова, защото и доколкото те са материален субстрат, носители на разменна стойност.“ ([9] с.198)

²⁶„1. Всеки качествен софтуер възниква, за да начеше кращата на разработчика. [...] 18. Ако искаш да разрешиш интересен проблем, първо започни да търсиш проблем, който е интересен за самия теб.“ ([14] правила 1 и 18) Тук направо се изхожда от тезата, че разработчикът трябва да е потребител на собствения си продукт. Потребител и производител трябва да съвпадат.

²⁷„Една специфична характеристика на ситуацията с Linux, която много допринася... е фактът, че работещите по всеки даден проект се самоизбират. [...] контрибуции се получават не от случайно подбрани хора, а от такива, които са достатъчно заинтересувани да използват софтуера, да научат как работи, да се опитат да намерят решения на проблемите, които срещат, и всъщност да направят несъмнено разумна поправка. Всеки, който отговаря на тези условия, е много вероятно да има какво да допринесе.“ ([14] с.7) и по-нататък: „За да направя fetchmail толкова добър, колкото тогава видях че може да бъде, трябваше да пиша не само за собствените си нужди, но също така да включавам и поддържам възможности извън моята орбита, но необходими за други хора.“ ([14] с.11)

²⁸„Всъщност, тъй като го ревизирах в края на май 1997, поради една интересна причина списъкът започна да губи членове след като достигна своя връх от около 300. Няколко души ме помолиха да ги отпиша, тъй като fetchmail им служел толкова добре, че те вече нямали нужда да следят трафика от списъка! Може би това е част от нормалния цикъл на живот на един зрял проект от базарен тип.“ ([14] с.8) По същият начин, Линус Торвалдс признава през 1999 г.: „Чистата истина в момента е, че не предвиждам големи преработвания на ядрото. Един успешен софтуерен проект трябва да достигне зрялост по някое време, след което темпото на промените се забавя. За ядрото не предстои кой знае колко на брой големи нововъведения.“ ([26])

Трябва да се разгледа защо Ерик Реймънд не вижда и не може да види това, което е между редовете на собственото му произведение. Напротив, той привижда разглежданите от него отношения като пазарни. Той много добре си дава сметка, че те са различни от досегашните отношения, но като такива ги обявява за „по-пазарни“ от тях. Двойна грешка при Реймънд.

От една страна, в новите отношения има качествена разлика. И понеже разликата е толкова съществена, че тези отношения са свързани с нов тип собственост – една обща собственост, която на практика е на всички и на никого. По този начин Minix лесно може да се превърне в готов суров материал за Linux, или rpmclient да послужи за създаване на fetchmail в случая на Ерик Реймънд. В сферата на свободния софтуер са познати и цикли на няколко нива, където един готов продукт неколккратно се превръща в материал за друг и т.н.²⁹ Общата собственост е условие за това движение.

На традиционния начин на производство Реймънд противопоставя „отворения“ начин (и от там – традиционните отношения срещу „отворените“). Но понеже новите отношения били уж по-пазарни, т.е. в очите на един либерал това трябва да са едни наистина добри отношения, то в тяхната светлина старите не изглеждат да са никак пазарни. Затова новите се наричат базарни, а старите – наопаки – били катедрални, т.е. затворени, едва ли не феодални. Затова откровенията на Кропоткин за отношенията *владетел-крепостни* стават актуални за Реймънд и той ги цитира с охота, но със своето историческо безгрижие пропуска да забележи, че феодалните отношения са отмерили именно заради новите капиталистически отношения, които той критикува. Но понеже Реймънд не може да си позволи критика на капиталистическите отношения, той я заменя с критика на феодалните, тихо-мълком слагайки знак за равенство между двете съвсем различни форми. Програмистите биват наречени „затворени в кутийки ратаи с управители, които размахват камшици наоколо им“ ([14] с.18). Тук изобщо не се рефлектира върху това, че именно пазарния живот на софтуера, т.е. животът му като стока, произвежда т.нар. традиционен, катедрален начин на производство.

Първо, новите отношения не са пазарни, а анти-пазарни. Затова не е коректно да се наричат „базарни“.

Второ, исторически, процесът на издигане на катедралите съвсем не е такъв, какъвто си го представя Ерик Реймънд. Факт е, че този трудов процес не е нито катедрален, нито базарен (в смисъла на Реймънд), но ако говорим за някаква алегория, парадоксът е, че той е по-скоро базарен, отколкото катедрален.³⁰

²⁹ Например продукта Window Maker е базиран на AfterStep, който от своя страна е продължение на BowMan на Бо Йънг. А пък BowMan използва код от fvwm на Робърт Нейшън. От своя страна пък fvwm е използвал twm и така нататък...

³⁰ „Строителството на готическите катедрали е забележителна страница в историята на художествения живот. За място на катедралите обикновено избирали градския площад, *открит от всички страни и твърде оживен* по време на панаири. Строежът на катедралите изисквал *големи средства и много време*. Градската община, решила да строи катедрала, изпращала из околностите и в далечни краища събирачи на *дарения*. Когато набирали нужната сума, полагали основния камък и често пъти издигали само хора, за да може да започнат богослужението. Много катедрали така си и останали недовършени. Миланската и Кьолнската катедрали придобили завършения си вид едва през 19. век. Хората не били уверени, че ще видят плода на своите усилия. Но нищо не било в състояние да угаси творческия пламък на онова време.

[...] Най-старите готически катедрали са построени във владенията на френските крале. Строежът на покъсните готически катедрали също бил поощряван и подпомаган от владетелите. За това свидетелствуват между другото и т.нар. кралски галерии по фасадите на тези катедрали. Формиращата се през това време монархия, също както борещото се срещу феодалите градско съсловие, взела под свое покровителство новия тип църкви. Но през 12. век монархията не била още достатъчно укрепнала, за да упражнява решително влияние върху художественото творчество.

В строителството на готическите катедрали е намерил израз все още здравият комунален строй. Нужна е била крепката сплотеност на огромна общност, за да се доведе до успешен край започнатият строеж.

[...] В строежа на катедралите *участвали много хора*. Тук били и представителите на градското самоуправление, които се занимавали с всички въпроси на строителството, и духовенството – църковни служители

Вярно е, че това са общности на самоорганизиращи се егоисти, но те не са свързани помежду си с пазарни отношения. Между отделните участници няма противоречие, подобно на противоречието между производителя на ябълки и производителя на ябълков сок, където единият иска да „изиграе“ другия, за да спечели повече. Нищо подобно. Тук успехът на един е условие за успеха на друг. Частният интерес съвпада с общия. И понеже продуктът на труда принадлежи на всички, не се противопоставя като *външен* и *чужд*, егоистите не участват в отношения на отчуждение и самоотчуждение. Това е спойката на един *коллективизъм, основан на индивидуализъм*, или казано иначе – *модерен комунизъм*, „една асоциация в която свободното развитие на всекиго е условие за свободното развитие на всички.“ ([11] с.58).

3.2.2. Поуката от Mozilla

Свободният софтуер разчупва софтуера така, че превръща всеки софтуер в *библиотека* – като натрупан труд, чиято цел е да бъде използван отново за създаването на нов софтуер и/или прилаган в друга област. И не само като натрупан труд, но и като всеобщо усъвършенстване на производителните сили. Затова свободният софтуер предполага:

1. нова форма на общуване, и
2. нова форма на собственост (защитена от т.нар. свободни лицензи)

В „Катедралата и базарът“ се разглежда само първия компонент, при това на места с виртуозност, а на места по „странен“ начин. Епилогът на „Катедралата и базарът“ е много симптоматичен. Произведението е прието с бурни аплодисменти из общността на свободния софтуер, но оказва и значително влияние в индустрията. Ръководителите на Netscape решават да отворят изходния код на своя web-четец Netscape Communicator под името Mozilla.³¹

В последствие се оказва, че проектът е затлачен, че потребителите не се превръщат в съразработчици. Към проекта Mozilla не се присъединява никаква общност и той остава да бъде разработван в изолация, макар и пред очите на целия свят. Какво се е случило?

Най-напред, погрешно е да се смята, че при общуването в самата Linux-общност няма отношения на подчинение. Линус Торвалдс е този, който в крайна сметка преценява и богослови, внасящи своя дял на ученост, но естествено решаващо значение имали артелите на строителите, зидарите и каменоделците, сред които поколения наред се възпитавал художественият вкус и се трупал технически опит.

[...] Покрай множеството безименни строители от онова време до нас са достигнали имената и произведенията на няколко завършени творчески личности.“ ([1] с.154-155, курсивът е мой – К.Д.)

Ако трябва да се позовем на англоезичен автор, можем да приведем думите, с които Кенет Кларк описва строежа на Шартърската катедрала: „От старите хроники знаем нещо за хората, чийто душевен мир разкриват тези лица. Там се говори, че в 1144 г., когато кулите се издигали сякаш по чудотворен начин, вярващите впрягали каруците, с които пренасяли камъни, и ги теглели от каменоломните до катедралата. Въодушевлението обхванало цяла Франция. Мъже и жени идвали от далечни места, носейки тежки товари с хранителни припаси за работниците – вино, зехтин, жито. Сред тях имало благородни господа и дами, които теглели колите редом с останалите. Царяла съвършена дисциплина и най-дълбока тишина. Всички сърца били сплотени и всеки прощавал на враговете си. Тази всеотдайност към един голям идеал на цивилизацията се разкрива дори още по-внушително, когато минем през портала и влезем в самата църква...“ ([5] с.76-77)

Нима това е катедралният труд на Реймънд? Ето какво нарича той *катедрален модел*: „съграждан като катедрала [...] – внимателно майсторен от отделни вълшебници или малки групички магове, работещи в уютна изолация.“ ([14] с.2).

³¹В епилога Ерик Реймънд пише: „Ерик Хан, изпълнителен вицепрезидент и старши технолог в Netscape, малко по-късно ми изпрати следната електронна поща: «От името на всички в Netscape, искам на първо място да Ви благодаря, заедно ни доведохте дотук. Решението ни беше съществено повлияно от Вашите размисления и съчинения.»

През следващата седмица отлетях за Силициевата долина, откликвайки на поканата на Netscape за еднодневна стратегическа конференция (на 4 февруари 1998) с някои от висшите им администратори и технологи. Заедно проектирахме стратегията за пускане на изходния код и лиценза на Netscape.“ ([14] с.21)

постановява какво ще се възприеме в новата версия на Linux. Той неведнъж е отхвърлял предложения за поправки и код, изпратени от останалите разработчици.³² Но по някакъв начин разработчиците му се доверяват, той има авторитет, макар и да не принуждава никого да му се подчинява. Как така съществува подчинение без принуда, йерархия без привилегии? Едно много важно условие за това е, че всеки разработчик винаги има *действителната възможност да не приеме волята* на Линус Торвалдс и да за създаде алтернативен вариант на Linux, където са се реализирали други решения, материализирала се е друга воля. Понеже всеки е практически свободен и пред него са открити безкрайни възможности, той остава да следва една чисто практическа линия. Състезанието с останалите разработчици не е актуално.

Когато потребителните стойности клонят към безкрайност, производството престава да бъде самоцел. То вече може да бъде или насочено към задоволяването на непосредственото потребление, т.е. да има потреблението за своя цел, или да бъде естетическо (целесъобразност без цел), работа за удоволствие. Тъй като в този случай продуктът на труда не се противопоставя като чужд на работника, няма и отчуждение от самия труд като произвеждаща предмета дейност. Трудът вече не е акт на отчуждение.³³ Точно тук е мястото на удоволствието в разработката на свободен софтуер, а съвсем не там, където го поставя Ерик Реймънд.³⁴

За третирането на потребителя като съразработчик не са необходими само свойски покани за участие, похвали и потупване по рамото. Потребителят трябва действително да има равностойни потенцици с разработчика. Както разработчика може да направи всичко с продукта, така и потребителят трябва да може.

Според Реймънд, поуката от Mozilla е, че „отварянето няма задължително да спаси един съществуващ проект, който страда от лошо дефинирани цели, подобен на спагети код или която и да било друга хронична болест на софтуерното инженерство“ ([14] с.21).

Нима? Но личният експеримент на Реймънд – проектът fetchmail, също е страдал от лошо дефинирани цели.³⁵ Но именно базарните практики коригират това.³⁶ Не се намериха потребители, който да направят това и с Mozilla, не им беше позволено от самия статут на Mozilla – едно чуждо тяло, собственост на Netscape. Важно условие за успеха на Linux е, че кодът е защитен от клаузите GNU GPL. При такава гаранция, доброволците могат да допринасят свободно и спокойно, сигурни че даряват труда си на общността (плодовете на който труд в крайна сметка се връща при тях), а не на някой търговски субект. Експери-

³²За подробни материали вж. архивите на пощенския списък linux-kernel на Университета в Хелзинки <http://www.cs.helsinki.fi/linux/>

³³За разлика от „класическото“ стокотранспортно производство, където „[...] за работника трудът е нещо *външно*, т.е. не спада към неговата същност, и че поради това той не се утвърждава в своя труд, а се отрича, не се чувства щастлив, а нещастен, не разгръща свободно никаква физическа и духовна енергия, а изтезва тялото си и разрушава духа си. И затова едва извън труда работникът чувства, че е при себе си, а в процеса на труда – извън себе си. Той е у дома си, когато не работи, а когато работи не е у дома си. [...] Външният труд, трудът, при който човекът се отчуждава, е самопожертвувателен труд, труд на самоизтезаване. И най-последно, отчуждеността на труда се проявява за работника в това, че трудът не е негова собственост, а на друг, че не му принадлежи, че в труда той не принадлежи на самия себе си, а на някой друг.“ ([7] с.84)

³⁴Реймънд манифестира: „Нашата творческа игра носи технически, пазарни и интелектуални успехи с невяротно темпо.“ ([14] с.19) Но творчеството на програмистите в една софтуерна компания от типа на Microsoft, например, също носи технически, пазарни и интелектуални успехи. Така че това е общото, а не съществения признак за постигането на удоволствие. Проблемът за удоволствието е неделим от проблема за отчуждението.

³⁵„12. Често най-поразителните и новаторски решения идват след откритието, че представата ти за проблема е погрешна.“

Аз съм се питвал да реша грешен проблем, като съм продължавал да разработвам popclient като комбиниран MTA/MDA с всички изначанени видове доставяне на място. Дизайнът на fetchmail се нуждеше от основно преосмисляне...“ ([14] с.9)

³⁶„...идеята за SMTP-препращането е най-голямата отплата, която получих от съзнателния ми опит да подражавам на методите на Линус. Един потребител ми даде тази страшна идея...“

11. Щом нямаш добри идеи, тогава разпознавай добрите идеи на своите потребители. Понякога второто е за предпочитане.“ ([14] с.9)

ментът с fetchmail също е проведен под защитата на GNU GPL. Но с помощта на Реймънд, Netscape измайсторяват свой лиценз, несъвместим със съществуващата общност на свободния софтуер. Лиценз, който поставя Netscape в особено превилигирана позиция, белязана от една неадекватна търговска конвулсия, един отживял, безсмислен рефлекс. Затова процесът на производство на Mozilla протече по-скоро по стария, а не по новия начин. Неговите предпоставки бяха такива, и само чудо би могло да привлече общност от ентузиазирани себелюбиви доброволци.

Ерик Реймънд двойно закопава Netscape. Първо ги подвежда, като в „Катедралата и базарът“ изобщо не говори за необходимостта от обща собственост като условие на базарния модел. С това повлиява на тяхното решение да го поканят за сътрудник и проектант на стратегията за пускане на Mozilla. И втори път, като проектира лиценз, който не взема предвид тази собственост, с което лишава Mozilla от възможността да акумулира общност от разработчици. Всъщност това са само две страни на една и съща грешка.

Не случайно производението завършва с проблема за Mozilla. Тази тема е не само текстуалният край, но и *самата граница* на възможностите на изследването на Ерик Реймънд. Там теорията катастрофира, демонстрирайки това чрез абсурдните причини за самотата на проекта.

3.3. Общество в криза

Ако Реймънд подвежда Netscape, то от своя страна Netscape са били „подведими“. Вече е съществувало някакво противоречие, което предварително подготвя почвата така, че да стане възможно „подвеждането“. В този план, случаят с Netscape е симптоматичен.

Всеки код, доколкото може да се преизползва, е натрупан труд. А доколкото може да се преизползва безкрайно (теоретически), т.е. колкото е потребен (практически), той се явява като усъвършенстване на производителните сили. Цялата тайна е в това отношение. Ако даден софтуер (като цяло, или някаква негова част или форма) е тайна, т.е. е достъпен само в дадена организация, той представлява натрупан труд, който позволява на тази организация да произвежда под обществено необходимото работно време.³⁷ Ако пък софтуерът е публично достояние, той позволява на всички да произвеждат под това работно време, т.е. променя обществено необходимото работно време и представлява усъвършенстване на производителните сили.

Организацията, която държи софтуера си в тайна, има предимство пред останалите конкуренти. Това е причината кодът да се крие ревниво с всички средства. Но ако конкуренцията изработва сроден софтуер, обществено необходимото работно време също се променя, изравнява се, и предимството на първата организация се стопява. Това води до настървена надпревара в производството.

Това е твърде неефективен начин за промяна на обществено необходимото работно време, тъй като по тази схема организациите паралелно извършват еднакъв труд, който всяка по отделно трябва да натрупа за себе си. Макар тази неефективност да е факт, тя не е само-разбиращ се, очевиден *проблем*. В крайна сметка това е нормалният начин на производство, пряко следствие от пазарните отношения. Има ли причина, поради която да смятаме, че този начин ще се промени?

Днес той *става проблем*, когато пред индустрията се изправят изисквания, които тя не може да покрие. През 2000 г. две развити държави – Съединените щати и Германия – обявяват програми за импорт на компютърни специалисти. Тази политическа воля отразява един индустриален факт. Това означава, че според техните разчети и прогнози, компютърните специалисти са недостатъчно и тази липса ще става все по-остра. В световен мащаб

³⁷ „Обществено необходимото работно време е онова работно време, което при съществуващите нормални за дадено общество условия на производството и при обществено средна степен на умение и интензивност на труда е необходимо за изработването на някоя потребителна стойност.“ ([9] с.49)

съществува дефицит на работна сила в този бранш, макар че никога не е имало толкова много програмисти, колкото днес.

Как се стигна дотук? Софтуерът е дифузен. Той лавинообразно експанзира от своята тясна сфера и залива всички останали. Няма производство, което да не може да бъде оптимизирано чрез софтуера. Някаква част от всяко производство може да се превърне в софтуер. Това всеобщо състезание води до растящите потребности във:

- всички сфери на дейност
- самата IT индустрия (софтуер за себе си)

По този начин „производството“ на програмисти не може да догони растящата обществена необходимост. Още повече, че скоростта на растеж на тази необходимост е по-голяма от скоростта на реакция от страна на процеса на (интелектуално) производство на програмисти, който изисква време. Затова работните места са повече, отколкото работниците. Днешното положение на програмистите, мнозинството от които наемни работници, съществено се различава от положението на работниците по времето на Маркс.³⁸

Съединените щати и Германия скрито предпоставят, че щом трябва да се извърши повече труд, значи е необходима повече работна сила, като по този начин изпадат в обясним дребен фетишизъм. Напротив, щом не може да се произведе необходимото количество работна сила, тогава трябва да се увеличи ефективността на съществуващата работна сила. Свободният софтуер е решението на този проблем. Ако организациите споделят постиженията си, обществено необходимото работно време ще се променя мълниеносно. В тази критична ситуация, общият интерес взима превес над частния.

Индустрията е изправена пред проблем, който не може да разреши, оставайки на своето стъпало. А доколкото тази индустрия е дифузираща във всички сфери на дейност, това е обществен проблем.

³⁸ Ако си позволим по-фриволен израз, днес съществува една специфична „диктатура на пролетариата“, доколкото можем да привидим в Ричард Столман и Линус Торвалдс лицата на световния пролетариат.

Заклучение

Чрез проблематизирането на софтуера като философска задача би могло адекватно да се постави вълнуващият въпрос за свободния софтуер. Анализът на софтуера като специфичен продукт, облягайки се и същевременно конструирайки една нова интерпретация на текстове на Маркс, прави възможно проследяването на необходимостта в: 1) историята на свободния софтуер и 2) собствената му рефлексия. В светлината на Маркс свободният софтуер може да ни се яви като особен начин на производство. Този начин на производство носи със себе си като следствие и като причина – нова форма на общуване и нова форма на собственост.

Изяснихме, че в основата на производството на софтуер (програмирането) лежи описанието на *сложни* задачи чрез относително *прости* команди. Макар простите команди да са краен брой, задачите, които могат да се решат с тях са безкрайно много. Софтуерът е своеобразно рефлексивен – бивайки универсален по природа, той може да бъде прилаган във всяка област на производството, включително, и най-вече в своята собствена област. Той може със свои средства да решава свои задачи, подобно на барон Мюнхаузен, който излиза от блатото като се издърпва за косата си.

От своя страна, програмисткият труд се натрупва в различни форми – езици за програмиране (еволюцията на езиците за програмиране), библиотеки, стандартизация. Това натрупване се явява основа за ново производство и ново натрупване, то е самозавихрящо се натрупване. В това ново производство, натрупаното вече играе ролята на простото – предоставя арсенал за ново безкрайно по разнообразие натрупване от по-високо ниво.

Спецификата на софтуера от една страна не му позволява лесно да влезе в пазарните отношения, да бъде произвеждан като стока, а от друга страна го прави привилегирована стока. Това постоянно противоречие движи софтуера в търсене на неговата адекватна стокова форма, пораждайки правни и морални норми. Създадените норми се опитват да разрешат противоречието, но едновременно с това го заострят по парадоксален начин.

Самото възникване на изчислителната машина се разкрива не като резултат от *изначалната* човешка любознателност, а е обвързано с конкретен исторически контекст, със завихрянето на капиталистическото производство, с необходимостта от постоянна оптимизация на това производство. При това важен катализатор е надпреварата във въоражаването. В своето развитие компютърната индустрия се разпада вътрешно на отделни подсфери и се задълбочава разделението на труда. Софтуерът и свързаната с производството му дейност се обособяват като самостоятелна област със своя логика и структура. Веднъж кристализирал, софтуерът притежава качества, които трябва да бъдат потиснати, за да може той успешно да участва в стокотранспортното производство, да бъде произвеждан като стока. С последвалата експанзия на софтуерната индустрия, излизат наяве, оголват се противоречия, които правят възможно движението за свободен софтуер. Свободният софтуер идва да „снее“ тези противоречия и същевременно с това да унищожи собствените си предпоставки.

Тъкмо затова в индустрията има привърженици на идеята за свободен софтуер. Тази идея е действително необходима, макар и подмолно да подкопава основите и предпоставките на капиталистическото производство. Това остро противоречие днес е действително и е отразено в разнообразното отношение на „старите и новите“ представители на индустрията. Индустрията трябва да поеме по този път, но ще трябва да се реорганизира съобразно новия

начин на производство, и на мястото на сегашните субекти ще изплуват други, но не непременно субстанциално други, а други по определенията на отношенията, в които участват. Обществената функция на Ерик Реймънд е да смекчи това противоречие. Той замаскира цената, която трябва да се плати, правейки необходим компромис. Това е главната предпоставка за неговата популярност, която стои извън самия него. Управителните съвети и „добрите американци“ като цяло не искат да слушат Столман. Реймънд гали ухото.

Понеже начинът на производство се променя, променят се всички отношения и правила в областта. За една традиционна компания като Microsoft, операционната система GNU/Linux се явява като пазарен конкурент на Windows¹. Потенциални потребители на Microsoft се превръщат в действителни потребители на GNU/Linux.

Как Microsoft би могла да противодейства? Освен чистата пропаганда, използвана в комерсиалните реклами и съобщения в медиите, един гигант като Microsoft би могъл да купи конкурента си и да използва или закрие продуктите му. Но нито софтуерът на GNU, нито Linux могат да бъдат купени. Никъде не се продават акции на GNU и Linux. Впрочем, зад Linux дори не стои никаква формална организация. Кой произвежда Linux? Една дифузна общност. А Впрочем, дори не може да бъде фиксирана държавата в която Linux се произвежда – светът е неговият истински цех.

Как да купим Linux и да го закрием? Как да победим конкурента? На 12 септември 2000 г. Стив Балмър, главен изпълнителен директор на Microsoft, признава, че „феноменът Linux“ е една от най-големите заплахи за Microsoft. «Но», добавя Балмър, «не съм сигурен кои Linux-компани да назова.» Липсата на търговски субект обърква и води до това, че се задават традиционни въпроси, които вече не могат да получат пряк отговор.² Когато се задават такива плоски въпроси, това означава че те вече не са адекватни, че ситуацията се е изменила коренно. Богатството на свободния софтуер има *друга природа*.³ За да може Microsoft, или която и да е друга компания от „стар тип“ да се съизмери със свободния софтуер, тя трябва да приеме неговия начин на производство. Тя трябва да стане от *един и същ порядък* с него, което едновременно с това значи, че те ще престане да съществува като такава.

Историята на софтуера е история на средствата на труда. Всяко развитие на тези средства на труда представлява развитие на производителните сили и следователно – белег за нова епоха, която влече със себе си нови обществени отношения.⁴ Свободният софтуер е новото средство, което неминуемо установява нови отношения с постоянстваща скорост и неизбежност.

Настоящата работа беше създадена изцяло със свободен софтуер.

¹Тук Windows се употребява като общо име за всички продукти с марката Windows 95/98/NT/2000 и прочие.

²Същият месец Microsoft като че ли разпозна своя противник и „превзе“ Corel за да принуди компанията да закрие своята версия на Linux, както и основните си продукти за Linux. Corel беше привидяна като основен доставчик на Linux, като една типична компания от старо поколение и, следователно, един удобен противник. Уви, Linux далеч не се изчерпва с Corel Linux. Напротив, Corel Linux беше само маргинално проявление сред истинските GNU/Linux системи, създавани и поддържани от общности, а не от корпорации. Така действията на Microsoft ни се разкриват като един отчаян акт, една безсмислена атака срещу безплътния мираж. По същество GNU/Linux ни най-малко не пострада от усилието на Microsoft. Свободният софтуер е призрак и само призрак може да му се изпречи на пътя. Старите методи на борба с конкуренцията не работят, напротив, те само увеличават жизнеспособността на призрака. Впрочем, политиката на Corel доведе компанията до гибел и започна това, което Microsoft само довърши. И двете компании са жертви на собствения си ограничен хоризонт.

³„Например една stockjobbing nation [нация от борсови спекуланти] не може да бъде ограбена по същия начин както една нация от скотовъдци.

Грабежът на роби е направо грабеж на производствения инструмент. Но тогава производството на страната, за която става грабежът, трябва да има такава структура, че да допуска робски труд, или (както е за Южна Америка и т.н.) трябва да бъде създаден такъв начин на производство, който да съответствува на робите.“ ([10] с.242)

⁴„Средствата на труда са не само мярка за развитието на човешката работна сила, но и показател за онези обществени отношения, при които се работи.“ ([9] с.192)

БИБЛИОГРАФИЯ

- [1] Алпатов, Михаил В.; *История на изкуството*, том II. Изкуството на Средновековието. изд. „Български художник“, София, 1982 г.
- [2] Бабалиевски, Филип; *Научно публикуване чрез електронна поща*, „Гей-Либрис“, София, 1997 г.
- [3] Басмаджиев, Борис; *E-mail и още...*, изд. „Интернет в България“, София, 1995 г.
- [4] Енгелс, Фридрих; *Писма за историческия материализъм*, Партиздат, София, 1989 г.
- [5] Кларк, Кенет; *Цивилизацията*, изд. Български художник, София, 1977 г.
- [6] Магуайър, Стив; *Как да пишем надеждни програми*, изд. Нисофт, София, 1994 г.
- [7] Маркс, Карл; *Икономическо-философски ръкописи от 1844 г.*, в: *Съчинения*, том. 42, Издателство на Българската Комунистическа Партия, София, 1983 г.
- [8] Маркс, Карл; *Икономически ръкописи 1857-1859 г.*, в: *Съчинения*, том. 46, част първа, Издателство на Българската Комунистическа Партия, София, 1978 г.
- [9] Маркс, Карл; *Капиталът*, Партиздат, София, 1975 г.
- [10] Маркс, Карл; *Увод към критиката на политическата икономия*, в: *Към критиката на политическата икономия*, Печатница на Бълг. Комунистическа Партия, София, 1949 г.
- [11] Маркс, Карл; Енгелс, Фридрих; *Манифест на комунистическата партия*, изд. „ГАЛ-ИКО“, София, 1999 г.
- [12] Маркс, Карл; Енгелс, Фридрих; *Немска идеология*, в: *Съчинения*, том. 3, Издателство на Българската Комунистическа Партия, София, 1957 г.
- [13] Реймънд, Ерик С.; *Как да стана хакер?*,
оригинал: <http://www.ccil.org/~esr/faqs/hacker-howto.html>
превод на български: <http://linux.gyuvet.ch/html/paper/hackers.html>
- [14] Реймънд, Ерик С.; *Катедралата и базарът?*, 1996-2000 г.,
оригинал: <http://www.tuxedo.org/~esr/writings>
превод на български: <http://catb-bg.sourceforge.net>
- [15] Хсу, Джефри; *Програмни езици за микрокомпютри*, изд. „Техника“, София, 1992 г.
- [16] Downing, Douglas A.; Covington, Michael A.; Covington, Melody Mauldin; *Dictionary of Computer and Internet Terms*, Sixth Edition, Barron's Educational Series, Inc., New York, 1998.
- [17] Free Software Foundation; *GNU General Public License*,
<http://www.gnu.org/copyleft/gpl.html>
- [18] Gates, Bill; *An Open Letter to Hobbyists*,
<http://junior.apk.net/~firebug/computer/gates.txt>

- [19] Microsoft; *End-User License Agreement for Microsoft Software, DCOM95 for Windows 95, version 1.3*, <http://www.microsoft.com/Com/DCOM/Dcom95/eula.asp>
- [20] Raymond, Eric S.; *A Brief History of Hackerdom*,
<http://www.tuxedo.org/~esr/writings/hacker-history/>
- [21] Raymond, Eric S.; *Jargon File Resources*,
<http://www.tuxedo.org/~esr/jargon/>
- [22] Stallman, Richard M.; *Categories of Free and Non-Free Software*,
<http://www.gnu.org/philosophy/categories.html>
- [23] Stallman, Richard M.; *GNU Project – Initial Announcement*,
<http://www.gnu.org/gnu/initial-announcement.html>
- [24] Stallman, Richard M.; *The GNU Project*,
<http://www.gnu.org/gnu/thegnuproject.html>
- [25] Stallman, Richard M.; *What is Free Software?*,
<http://www.gnu.org/philosophy/free-sw.html>
- [26] Torvalds, Linus; *The Linux Edge*, в: *Open Source: Voices from the Open Source Revolution*, O'Reilly & Associates, 1999.
<http://www.oreilly.com/catalog/opensources/book/linus.html>
превод на български: <http://linux.gyuvet.ch/html/paper/kernel.html>
- [27] Wozniak, Steve; *Letters – General Questions to Woz Answered*,
<http://www.woz.org/letters/general/index.html>

GNU Free Documentation License

Version 1.1, March 2000

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being

those of Invariant Sections, in the notice that says that the Document is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, \LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A.2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
- State on the Title page the name of the publisher of the Modified Version, as the publisher.
- Preserve all the copyright notices of the Document.
- Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.
- Include an unaltered copy of this License.
- Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- Delete any section entitled “Endorsements”. Such a section may not be included in the Modified Version.
- Do not retitle any existing section as “Endorsements” or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

A.6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a

single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.7. Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

A.8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

A.9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.10. Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no Front-Cover Texts, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.