DIW Berlin

German Institute
for Economic Research

Discussion Papers

296

Jürgen Bitzer
Philipp J. H. Schröder

Bug-Fixing and Code-Writing:
The Private Provision of Open
Source Software

Berlin, September 2002

Opinions expressed in this paper are those of the author and do not necessarily reflect views of the Institute.

Bug-Fixing and Code-Writing:
The Private Provision of Open Source Software

Jürgen Bitzer and Philipp J.H. Schröder*

July 2002

Abstract

Open source software (OSS) is a public good. A self-interested individual would consider providing such software, if the benefits he gained from having it justified the cost of programming. Nevertheless each agent is tempted to free ride and wait for others to develop the software instead. This problem is modelled as a war of attrition with complete information, job signaling, repeated contribution to the public good and uncertainty in programming. The resulting game does not feature any delay: software will be provided swiftly, by young, low-cost individuals who gain considerably by signaling their programming skills; the startup (and collapse) of an OSS project displays bandwagon dynamics.

Keywords: open source software, war of attrition, public goods.
JEL classification: H41, L86, L31

---

*DIW Berlin – German Institute for Economic Research. Corresponding author: Philipp J.H. Schröder, DIW Berlin, Königin-Luise-Str. 5, 14195 Berlin, Germany, Tel.:+ 49 30 89789692, Fax:+49 30 89789108, email: pschroeder@diw.de.

1

1 Introduction

Open Source Software (OSS) is a public good supplied by voluntary private contributions. The term OSS refers to the fact that everyone can access and alter the program source code; it is a public good, because under the common GNU GPL license, any additions made by one programmer/user must be made available at no cost to all other programmers/users. Contrary to what one might expect, there are no signs of under-provision or delays in supply.[1] Several OSS projects even capture market shares from their commercial competitors and have a reputation for innovation and reliability. Popular examples are Linux, which is currently estimated to run on some 18 million computers (Linux Counter (2002)), and the Apache Web Server, which is installed on 58% (i.e. 22 million units) of reachable web servers (Netcraft (2002)). These achievements are even more astounding, if one takes into account the fact that millions of lines of codes have been written and thousands of software bugs have been fixed for each large OSS project, all by individual volunteer programmers. These programmers only associate with each other through informal communities, internet news groups and the common interest in OSS. They dedicate their time and effort free of charge, even though they are not able to exclude others from the benefits of their work.

This paper addresses the OSS phenomenon as a game of the private provision of a public good (see also Johnson, 2001). Following on from the work of Bliss and Nalebuff (1984), Hendricks et al. (1988) and Bilodeau and Slivinski (1996), the paper starts by developing a simple war of attrition setting, which builds on existing results and assumptions, whilst reflecting OSS peculiarities. For example, information on the status of projects and the abilities (programming techniques) of other programmers is freely available within

---

[1]For an introduction to the history and success of OSS see for example Stallmann (1999), Rosenberg (2000), Feller and Fitzgerald (2002) or Hars and Ou (2002).

the community of OSS programmers, as the source code is open and various OSS internet newsgroups exist (See DeBona et al. (1999)), hence we solve the game assuming complete information. Furthermore we extend the existing literature on the private provision of public goods by explicitly modelling some additional OSS specific characteristics. First, signaling: the providers of the public good receive an additional payoff, on top of the utility from the public good, from being able to signal their programming abilities. This signal creates value both through improving prospects on the job market, Lerner and Tirole (2000) and Raymond (2000b), and enhancing reputations within the community of programmers (see Raymond (2000a) and Diamond and Torvalds (2001)); this signal only continues for a limited period of time after the contribution takes place. Such signals are institutionalised in OSS activity: the OSS license obliges its programmers/users to document all programme changes made (including naming the author of the changes) in the software itself. Second, the simple private-provision-of-a-public-good model – where the OSS is a one-off discrete unit of a public good – is extended to incorporate modular software programming; agents improve the available OSS by creating complementary software modules, thus making repeated contributions to the public good. This extension means that the certainty assumption in the model has to be adapted; this acknowledges the fact that the success of any programming effort is uncertain. The individual agents do not know *a priori* how many modules will actually be created at any moment in time, but they do know how many agents are expending effort on developing additional modules, and that the value of a new module increases with the number of other software modules developed for the same OSS project.[2]

---

[2]More precisely, the model assumes that the functionality and applicability of any software module depends on how many other modules are developed. For example the improvement of an operating system developed by one agent increases the value of a newly programmed driver by another agent, and vice versa.

The paper derives the following results from the formal model. In a standard war-of-attrition game, where open source software is a one-off discrete unit of public good to be developed, the software will be provided without delay. The actual individual who provides the unit can be identified. *Ceteris paribus* this individual receives a greater benefit from using the OSS, gains more from being able to signal his programming ability, has a longer time horizon (i.e. a younger individual), is more patient, and faces lower costs for software development. If the software development examined is modular in nature, i.e. if agents repeatedly contribute to the public good, the war-of-attrition feature of the game collapses. Instead, agents have to decide when to join and exit the programmer community. Once again, software will be provided sooner rather than later. Individuals are more likely to join the programmer community (and will start to free ride at a later point in time), when they receive a greater benefit from the OSS, have a longer time horizon, are more patient, have lower costs when developing software, when there is a higher success rate in programming activity, when there is more complementarity between software modules, and when the community of programmers is larger. A larger benefit from signaling – even though it initially attracts agents to join a OSS community – also induces agents to start to free ride at an earlier stage, i.e. agents aim to harvest the (now larger) value of the signal. More importantly, the model establishes that the start up (and collapse) of an OSS community of programmers displays bandwagon dynamics.[3]

The remainder of the paper is as follows. The following section introduces the model and presents the assumptions used for the basic OSS phenomenon that result in a war-of-attrition setup; it only deals with equilibria where software development (the public good) is treated as a single discrete unit. In section 3, the game is extended to deal with software development of

---

[3]The addition of one agent to the community of programmers immediately increases the probability of other non-members joining.

a modular nature, i.e. individuals can repeatedly contribute to the public good. Section 4 concludes the paper.

2 The Model

Consider a population of N individuals. Each individual j has the ability to develop one discrete unit of OSS. Once developed it exists forever, is identical for all individuals, and its consumption is characterised by non-rivalry and non-excludability. Time is continuous and individuals discount the future at the rate r_j. Utility flows are as follows. Without the OSS, individuals have to use a commercial closed source software alternative and receive the utility flow v_j. From the time when the OSS is introduced, all individuals receive the flow utility $u_j = v_j + z_j$, where z_j $(z_j \geq 0 \ \forall \ j = 1, ..., N)$ is the flow value from the OSS being available. Individuals can produce the software at cost C_j, which will be specified below. C_j is the present discounted value of the net costs to the individual j who develops the OSS, i.e. it is the actual development cost minus any gains from the signaling of programming ability and the improved reputation within the community of programmers. Given these specifications we can state:

Lemma 1. *No individual in group m, defined by $C_j \geq \frac{z_j}{r_j}$ for all $j = 1, ..., m$, would ever develop the OSS. The community of potential developers consists of $n = N - m$ individuals i, for which $C_i < \frac{z_i}{r_i}$ for all $i = 1, ..., n$.*

In other words, lemma 1 states that those individuals who face lower costs when developing software, or who benefit more from the OSS, are potential developers of the software.

Assumption 1. *Within the community n, all costs and benefits are common knowledge.*

If a simultaneous one-time choice is modelled using the strategy set $\{develop, do\ not\ develop\}$, then this game becomes a static game of chicken, where the winning agents are able to free ride and receive the payoff $\frac{u_i}{r_i}$, and the losing agent develops the software and receives the payoff $\frac{u_i}{r_i} - C_i$. If no one develops the OSS, the payoff for all is $\frac{v_i}{r_i}$. As usual, this type of game features a host of pure and mixed strategy Nash equilibria, in which anyone might be the actual developer of the software. Hence one cannot deduce who will actually develop the OSS.

By allowing individuals the option of postponing their decision, such that they can wait and see if someone else will develop the software instead, some dynamics can be added to the game. The length of time a member of the community n is willing to wait naturally depends on the benefit he would gain if the OSS were to exist, the costs of developing the software himself, and his time preference. In the normal form version of this game, a pure strategy is a time $t_i \epsilon [0, \infty)$, when i will develop the software if no one else has already done so.

The following payoffs can be stated. If the OSS is developed by individual $j \neq i$ at time t, i's payoff is:

$$F_i(t) = \frac{v_i}{r_i}\left(1 - e^{-r_i t}\right) + \frac{u_i}{r_i}e^{-r_i t} \tag{1}$$

The net cost of developing the software consists of a one-off development cost c_i, and a net utility flow s_i incurred for Δ periods. The term s_i denotes the gain from being able to signal programming skills, i.e. the signaling value, which could either accrue in a job market/wage negotiation setting (Lerner and Tirole (2000)) or from the improved reputation and social standing within the community (Torvalds and Diamond (2001)). The total net cost for agent i of voluntarily developing the software at time t is:

$$C_i(t) = c_i e^{-r_i t} - \frac{s_i}{r_i}\left(e^{-r_i t} - e^{-r_i(t+\Delta)}\right) \tag{2}$$

Assuming that Δ extends to infinity, then by lemma 1 we know that $u_i + s_i - r_i c_i > v_i$. Thus every individual in the community n would develop the OSS in period 0 rather than to live without it for ever. Now, if individual i develops the software at time t his payoff is

$$D_i(t) = F_i(t) - C_i(t) \tag{3}$$

Finally, if no one ever develops the software, individual i receives the payoff $R_i = \frac{v_i}{r_i} = lim_{t\to\infty} F_i(t) = lim_{t\to\infty} D_i(t)$.

Lemma 2. *Any individual i such that $c_i < \frac{s_i}{r_i}$ will develop the OSS voluntarily and immediately at time $t = 0$.*

Proof is given in the appendix. In plain speech, lemma 2 states that an individual, who gains considerably from being able to signal programming ability, simply develops the software, rather than waiting for someone else to provide it.[4] By lemma 2, the game ends at time $t = 0$. A more complex game emerges under the following assumption.

Assumption 2. $c_i \geq \frac{s_i}{r_i}$ *for all $i = 1, ..., n$.*

Through assumption 2 and lemma 1, we know $F_i(t) > D_i(t) > R_i$ for all t and the game becomes a n player continuous time war of attrition (Hendricks et al. (1988)).

Using the existing results in Hendricks et al. (1988) and Bilodeau and Slivinski (1996), one can characterise the following equilibria for this type of game. *For every individual i, there is a subgame perfect equilibrium outcome in which only i will develop the OSS immediately.* If no one else but i develops the OSS, then i's best strategy is to develop the OSS immediately, and if i develops the OSS immediately, it is everyone's best strategy to wait. So

---

[4]Implicitly we assume that several agents can develop the software, but that only one unit of software is created. However, all developing agents receive their signaling payoff.

the game still permits – as is usual for this type of game – many subgame perfect equilibria in which anyone might volunteer. As shown in Bilodeau and Slivinski (1996), the set of subgame perfect equilibria can be radically reduced once time horizons become finite, regardless of how distant in the future the time limit is. Using this assumption, it is now possible to characterise fully the individual who will actually provide the public good.

Assumption 3. *All individuals* $i = 1, ..., n$ *have a finite time horizon,* T_i.

That is, T_i marks the fact that i is a finitely lived agent, or shows the point in time at which i moves to a different job (where he is unable to devote any effort on open source programming), or is the point in time when i's human capital becomes outdated. The altered payoffs become:

$$F_i(t) = \frac{v_i}{r_i}\left(1 - e^{-r_i t}\right) + \frac{u_i}{r_i}\left(e^{-r_i t} - e^{-r_i T_i}\right) \tag{4}$$

$$D_i(t) = F_i(t) - c_i e^{-r_i t} + \frac{s_i}{r_i}\left(e^{-r_i t} - e^{-r_i T_i}\right) \tag{5}$$

$$R_i = \frac{v_i}{r_i}\left(1 - e^{-r_i T_i}\right) \tag{6}$$

The effect of a finite time horizon is that the game becomes non-stationary. Thus from an agents perspective, there is a time $\bar{t}$, where he will no longer choose to become developer of OSS. Beyond that point in time, even when the OSS is not provided, the dominant strategy is never to develop, i.e. $D_i(t) < R_i; \forall\, t > \bar{t}_i$. The critical time $\bar{t}$ is defined by $D_i(\bar{t}) = R_i$.

Lemma 3. *Individual* i *will not develop the OSS after time*

$$\bar{t}_i = T_i - \frac{1}{r_i}\ln\left(\frac{z_i + s_i}{z_i + s_i - r_i c_i}\right). \tag{7}$$

Note that if assumption 2 is violated, then $\frac{z_i + s_i}{z_i + s_i - r_i c_i} < 1$ and hence $\bar{t}_i > T_i$ holds, i.e. these individuals gain no utility from waiting. Also, if $z_i + s_i < r_i c_i$ then time $\bar{t}$ is not defined, which is in fact the condition of lemma 1, i.e.

8

individuals that are not members of the community n. Using lemma 3 it is possible to state (Bilodeau and Slivinski, 1996):

Proposition 1. *Given a finite time horizon for every individual in the community n and assuming that for all $i, j \in \{1, ..., n\}, i \neq j : \bar{t}_i \neq \bar{t}_j$, a unique subgame perfect equilibrium exists, in which the individual with the highest $\bar{t}_i$ volunteers at time $t = 0$.*

Proof is given in the appendix. The intuition for this proposition is straightforward. If an individual knows that he is the most interested in having the OSS, and if he knows that everyone else knows this as well, then he might as well give in right away. The unique developing individual is characterised by the highest $\bar{t}$. Formally,

Proposition 2. *Ceteris paribus an individual who*
  i) would gain more from the software, z_i
  ii) would gain more from signaling, s_i
  iii) has a longer time horizon, T_i (younger)
  iv) has a lower discount rate, r_i (more patient)
  and who
  v) faces lower costs for software development, c_i
is more likely to provide the OSS.

Proposition 2 follows on from proposition 1 and lemma 3. In particular, result *iv* – the effect of a change in the discount rate – differs from the results that Bilodeau and Slivinski (1996) derive using intuitive reasoning and indeed differs from the conventional intuition behind the war of attrition, where patience is a good strategy for winning. Formal proof of our result is given in the appendix.

3 Modular Software: Repeated Contribution to the OSS

In this section, the assumption that the OSS is a discrete unit of software is relaxed. Instead it is assumed that the OSS is modular in structure, i.e. the amount of the public good is determined by the number of contributing agents. Stock effects are eliminated by assuming that agents are only interested in new and additional advances in software, thus normalising the utility in a no-further-modules situation. It is assumed that commercial software has a certain exogenously given and fixed growth rate, generating the flow utility Δv_j. Similarly, the flow utility from progress and growth in the OSS is denoted by $Z_j(\Delta l)$, where Δl is the additional number of OSS modules developed at a certain point in time. Thus agent j's utility becomes $u_j = \Delta v_j + Z_j(\Delta l)$ when software is modular in nature.

Programming effort in OSS bears some uncertainty of success. Thus it is assumed that the occurrence of a new, functioning OSS module follows a Poisson distribution with the arrival rate λ, where λ represents the common programming ability in the community. So the probability that Δl modules are created at a certain point in time, if n agents are involved in programming is $\frac{(\lambda n)^{\Delta l}}{\Delta l!}e^{-\lambda n}$. A specification from the quality ladder model literature (e.g. Aghion and Howitt (1992)) is borrowed to capture the fact that the OSS's applicability and functionality increase as the number of modules developed by other agents in parallel increases. In particular we postulate that the utility flow from OSS is given by $Z_j(\Delta l) = z_j\gamma^{\Delta l}$, where $\gamma > 1$ measures the resulting gain in applicability and functionality stemming from the complementarity of software modules, and where $z_j \geq 0$ represents an agent's preference for OSS software over commercial software. The expected utility flow from newly developed OSS software at any point in time is thus

$$E(Z_j(\Delta l)) = \sum_{\Delta l=0}^{\infty} \frac{(\lambda n)^{\Delta l}}{\Delta l!} e^{-\lambda n} z_j \gamma^{\Delta l}$$
$$= z_j\, e^{(\gamma-1)\lambda n} \qquad (8)$$

The utility flow stemming from OSS depends on the number of other agents participating; this creates strategic interaction among agents. However, this is no longer a war of attrition: it is no longer possible for an individual to wait out his opponents. Rather, agents now have the dilemma of when or whether to join and leave the community of programmers, whereby their additional (withdrawn) effort increases (decreases) the probability of new, functioning modules occurring. This problem can be addressed by examining the expected lifetime utility of an agent i, who is a member of the community n. When all agents including i develop OSS modules throughout their entire lifetime, this gives the payoff

$$D_i = \frac{\Delta v_i + z_i e^{(\gamma-1)\lambda n} - c_i + s_i}{r_i} \left(1 - e^{-r_i T_i}\right) \qquad (9)$$

In an environment of continuous software development, the characteristics of signaling have to change to reflect the fact that reputation and job signal decay swiftly if not maintained. This could either be caused by dynamics within the community, where only agents who actually make contributions are part of the reputation culture, or on the job market, where, due to continuous technological progress, last season's human capital has no value. However at the moment in time when an agent leaves the community of programmers, he still retains his gain from signaling, without actually having to programme. Furthermore, having stopped developing modules himself, the agent continues to have access to the software developed by other agents, so he still receives the full benefit from the OSS, but without the cost of having to contribute himself. So the payoff for agent i, who starts to freeride at time t, assuming that all other agents continue in the community becomes:

$$F_i(t) = \frac{\Delta v_i + z_i e^{(\gamma-1)\lambda n} - c_i + s_i}{r_i}\left(1 - e^{-r_i t}\right) \tag{10}$$
$$+ \frac{\Delta v_i + z_i e^{(\gamma-1)\lambda(n-1)}}{r_i}\left(e^{-r_i t} - e^{-r_i T_i}\right) + s_i e^{-r_i t}$$

Solving $F_i(\bar{t}) = D_i$ defines the point in time $(\bar{t})$ at which an individual who is a member of the community – developing OSS software – will leave it and start to free ride.

Lemma 4. *Individual i will stop to contribute modules to the OSS at time*

$$\bar{t}_i = T_i + \frac{1}{r_i}\ln\left(1 + \frac{r_i s_i}{z_i e^{(\gamma-1)\lambda(n-1)} - z_i e^{(\gamma-1)\lambda n} + c_i - s_i}\right). \tag{11}$$

For freeriding to actually occur during an agent's lifetime, the following requirement has to be introduced.

Assumption 4. $z_i e^{(\gamma-1)\lambda n} - z_i e^{(\gamma-1)\lambda(n-1)} - c_i + s_i > r_i s_i$ *for all $i = 1, ..., n$.*

Assumption 4 ensures that the logarithm in (11) produces a negative number. For a very large c_i the condition will be violated, implying that such an individual is not a member of the community. Also, assumption 4 will eventually hold for a large enough n, implying that free riding will occur in large communities.

Differentiating (11) with respect to the various parameters,[5] we can characterise free riding individuals.

Proposition 3. *Ceteris paribus an individual will start to free ride later, i.e. $\bar{t}$ will be higher, when he*

i) gains more from the open source software, z_i

ii) has a longer time horizon, T_i (younger)

iii) has a lower discount rate, r_i (more patient)

---

[5]The precise expressions of the derivatives are provided in a separate appendix available from the authors upon request.

iv) faces lower costs for software development, c_i

v) is part of a community with a higher arrival rate, λ

vi) takes part in a project with larger complementarity among modules, γ

vii) is part of a larger community, n

and

viii) has a lower signaling value, s_i.

Proof that $\frac{\partial \bar{t}_i}{\partial r_i} < 0$ and $\frac{\partial \bar{t}_i}{\partial s_i} < 0$ is given in the appendix. A larger signaling value – proposition 3 *viii* – can induce free riding, because when the agent leaves the community, he harvests a one-off gain by maintaining the signaling value while not facing the costs of programming. As this gain increases, the free riding payoff also increases and the agent will thus exit the community sooner. Furthermore an important result on the dynamics of the community emerges from proposition 3 item *vii*.

Corollary 1. *The exit of any agent i from the community n makes all other $n-1$ agents revise their stopping time downwards.*

Thus, the collapse of an OSS project displays bandwagon dynamics. The exit of one agent increases the probability that in the next instant another agent will exit.

To examine the start-up of an OSS project, the incentives to join a programmer community have to be examined using the same principles. The payoff for an agent j who never develops any OSS modules himself, and when a community of n programmers exists, is

$$F_j = \frac{\Delta v_j + z_j e^{(\gamma-1)\lambda n}}{r_j} \left(1 - e^{-r_j T_j}\right) \tag{12}$$

If such an individual joins the community at time t and starts to develop modules himself, whose signaling value only occurs after the agent has actually started this activity, the payoff becomes

13

$$D_j(t) = \frac{\Delta v_j + z_j e^{(\gamma-1)\lambda n}}{r_j} \left(1 - e^{-r_j t}\right) \tag{13}$$
$$+ \frac{\Delta v_j + z_j e^{(\gamma-1)\lambda(n+1)} - c_j + s_j}{r_j} \left(e^{-r_j t} - e^{-r_j T_j}\right) - s_j e^{-r_j t}$$

Then solving $D_j(\underline{t}) = F_j$ defines the time up to which an agent would still want to join the community.

Lemma 5. *Individual j will not join the OSS community n after time*

$$\underline{t}_j = T_j + \frac{1}{r_j} \ln\left(1 + \frac{r_j s_j}{z_j e^{(\gamma-1)\lambda n} - z_j e^{(\gamma-1)\lambda(n+1)} + c_j - s_j}\right). \tag{14}$$

Of course, time $\underline{t}_j$ is only slightly less than the stopping time derived in (11), implying that after joining, an agent will continue at least a certain period of time before starting to free ride. Furthermore, since $\frac{\partial D_j(t)}{\partial t} < 0$ we can state:

Lemma 6. *Any agent j who joins an existing OSS programming community during his lifetime will do so as early as possible.*

Finally, because (11) and (14) have identical structures, the qualities laid out in proposition 3 are also possessed by those agents, who would still find it beneficial to join the programming community at a later point in time. It is important to note that the conditions in lemma 5 and lemma 6 might not be fulfilled in small communities, i.e. for low n. The time up until which an agent would consider joining the community might become negative for low n, implying that the agent would never become a member.

It then remains to be shown which characteristics an agent, who would actually start a OSS project from scratch, has. This type of individual fulfills $\underline{t}_j|_{n=0} > 0$. Plugging $n = 0$ into (14) and solving for the cost of programming gives:

Proposition 4. *(Starting module)* *An agent j with a programming cost c_j such that $c_j < s_j - z_j + z_j e^{(\gamma-1)\lambda} - \frac{r_j s_j}{1 - e^{-r_j T_j}}$ would develop an OSS starting module.*

The requirements for this proposition are in fact more restrictive than those in assumption 4. From proposition 4, it follows that younger agents (higher T_j), agents who have a lower discount rate, r_j, who gain more from signaling their programming skills, s_j, and who have a larger preference for OSS, z_j are more likely to start up an OSS project.

Finally, since $\frac{\partial t_j}{\partial n} > 0$ one can complement corollary 1:

Corollary 2. *The addition of an agent j to the community n makes all other non-member agents revise their latest joining time upwards.*

Thus, the startup of an OSS project displays bandwagon dynamics as well. A new member entering a community increases the probability that in the next instant another agent will join.

4   Conclusion

Even though OSS is a public good, it evolves at a rapid pace, developed for free by highly qualified, young and motivated individuals. The paper argues that once the OSS phenomenon is understood as the private provision of a public good, these features emerge quite naturally. This paper applies the standard war-of-attrition model of the private provision of public goods, due to Bliss and Nalebuff (1984) and Bilodeau and Slivinski (1996), to the OSS phenomenon. In order to capture OSS characteristics, we include the need for a particular software solution (Torvalds and Diamond (2001) and Raymond (2000a)) and the value of boosting one's reputation and job signaling (Raymond (2000b) and Lerner and Tirole (2000)) into the model. Further, driven by the observation that information within the community of programmers

is close to costless, the model is solved under the assumption of complete information. Assuming that only one unit of open source software is to be developed, the software is provided sooner rather than later; the individual who will actually provide the OSS *ceteris paribus* gains more from using the software, from being able to signal programming skills, is a younger individual (i.e. longer time horizon), is more patient (i.e. lower discount rate) and faces lower development costs. Thus, this model already yields results that compare well with accounts of the OSS phenomenon.

Nevertheless, important features of the OSS development process and important criteria, which influence the agents' decision to join the programmer community, are neglected in the basic model. Therefore the model is further extended to include modular programming; a key feature of the OSS development process. This means that agents can repeatedly contribute to the public good and that they make their decision under uncertainty on how many OSS modules will be developed by other agents. Thereafter, an agent's expected benefits and the resulting decision on whether to join or leave the programming community both depend on the number of other agents developing OSS modules, the complementarity effects between the single OSS modules, the programming ability within the community of programmers and the signaling value. The latter, in contrast to the basic model, now no longer lasts for the whole life of an agent, but decays swiftly following the exit of the agent from the community.

In addition to the results of the basic model we can, with these extensions, show that individuals are more likely to join the programming community (and start to free ride later), when the programming abilities in the community are better, when the complementarity between the single modules is higher and when the number of other programmers developing OSS modules is larger. In contrast to the basic model, the signaling value not only initially attracts agents to join the OSS community, but also induces agents to start

to free ride earlier, because the agents aim to harvest the benefits from the signal. Due to the impact of community size on the agents decisions, the start up (and collapse) of an OSS project displays bandwagon dynamics.

A Appendix

A.1 Proof of lemma 2

Proof. The condition $\frac{s_i}{r_i} > c_i$ implies $D_i(t) > F_i(t)$ for all t. Since $D_i(t)$ is monotone and falling in t, $D_i(0)$ maximises utility. □

A.2 Proof of proposition 1

Proof. [6] Relabelling individuals, the different $\bar{t}_i$'s can be ordered $\bar{t}_n > \bar{t}_{n-1} > ... > \bar{t}_1$. When the software has yet to be provided at time $t \in (\bar{t}_{n-1}, \bar{t}_n]$ agent n knows that no one else will ever develop the OSS. Since $D_n(t) > R_n(T_n) \forall t \epsilon (0, \bar{t}_n)$ and hence also for all $t \in (\bar{t}_{n-1}, \bar{t}_n)$, agent n's subgame perfect strategy is to develop the OSS, if any time $t \epsilon (\bar{t}_{n-1}, \bar{t}_n)$ is reached. Similarly at any time $t \in (\bar{t}_{n-2}, \bar{t}_{n-1}]$, agent n and $n-1$ are the last feasible candidates who could provide the software. But there is a time $\tilde{t} < \bar{t}_{n-1}$ and sufficiently close to $\bar{t}_{n-1}$, such that $F_{n-1}(\bar{t}_{n-1}) > D_{n-1}(\tilde{t})$, and hence $n-1$, and everyone else, will prefer to wait for n to volunteer at time $\bar{t}_{n-1}$. Hence, in any subgame perfect equilibrium, n will volunteer at some time $t \in (\tilde{t}, \bar{t}_n]$. By backwards induction, the unique subgame perfect equilibrium has n developing the OSS at $t = 0$. □

A.3 Proof that $\frac{\partial \bar{t}_i}{\partial r_i} < 0$ when software is a single unit

Proof. From (7) we derive

$$\frac{\partial \bar{t}_i}{\partial r_i} = \frac{-c_i}{r_i(z_i + s_i - r_i c_i)} + \frac{\ln\left(\frac{z_i + s_i}{z_i + s_i - r_i c_i}\right)}{r_i^2} \qquad (A.1)$$

which, given lemma 1 and assumption 2, consists of a negative and a positive term. Following proposition 2, we have to show that $\frac{\partial \bar{t}_i}{\partial r_i} < 0$. In manipulating

---

[6]This proof follows Bilodeau and Slivinski (1996).

(A.1), one can restate

$$\ln \left(\frac{z_i + s_i}{z_i + s_i - r_i c_i} \right) < \frac{z_i + s_i}{z_i + s_i - r_i c_i} - 1 \tag{A.2}$$

Which is always true since $a - \ln(a) > 1 \ \forall \ a > 0; a \neq 1$. □

A.4 Proof that $\frac{\partial \bar{t}_i}{\partial r_i} < 0$ when software is modular

Proof. From (11) we derive

$$\frac{\partial \bar{t}_i}{\partial r_i} = \frac{-b}{a - b} \ln \left(1 + \frac{-b}{a} \right) \tag{A.3}$$

where $a = -c_i + s_i - z_i e^{(\gamma-1)\lambda(n-1)} + z_i e^{(\gamma-1)\lambda n}$ and $b = r_i s_i$, and $a > b > 0$ holds due to assumption 4. Following proposition 3, it has to be shown that $\frac{\partial \bar{t}_i}{\partial r_i} < 0$. Rearranging (A.3) yields:

$$1 - \frac{a}{a - b} < \ln \left(\frac{a - b}{a} \right)$$

Finally, defining $x = \frac{a}{a-b} > 1$ this requirement can be restated as

$$0 < \ln \left(\frac{1}{x} \right) + x - 1 \tag{A.4}$$

Define the function

$$\begin{aligned} g(x) &= \ln \left(\frac{1}{x} \right) + x - 1 \\ &= x - \ln(x) - 1 \end{aligned}$$

Note that $g(1) = 0$ and $g\prime(x) > 0 \ \forall \ x > 1$. Condition (A.4) is therefore true for the relevant parameter range. □

19

A.5 Proof that $\frac{\partial \bar{t}_i}{\partial s_i} < 0$ when software is modular

Proof. From (11) we derive

$$\frac{\partial \bar{t}_i}{\partial s_i} = \frac{1}{c_i + a + r_i s_i - s_i + s_i \left((-1) + \frac{(r_i - 1)s_i}{-c_i - a} \right)} \tag{A.5}$$

where $a = z_i e^{(\gamma - 1)\lambda(n-1)} - z_i e^{(\gamma-1)\lambda n} < 0$; $\frac{\partial \bar{t}_i}{\partial s_i}$ is negative for:

$$c_i + a + r_i s_i - s_i + s_i \left((-1) + \frac{(r_i - 1)s_i}{-c_i - a} \right) < 0 \tag{A.6}$$

Solving A.6 for c_i leads to:

$$s_i - a > c_i \tag{A.7}$$

Inequality A.7 holds due to assumption 4. Thus, $\frac{\partial \bar{t}_i}{\partial s_i} < 0$.

$\square$

References

Aghion, P. and P. Howitt (1992), A Model of Growth Through Creative Destruction, *Econometrica*, Vol. 60, No. 2 (March), pp. 323-351.

Bilodeau M. and A. Slivinski (1996), Toilet cleaning and department chairing: Volunteering a public service, *Journal of Public Economics*, Vol. 59, pp. 299-308.

Bliss, C. and B. Nalebuff (1984), Dragon-slaying and ballroom dancing: The private supply of a public good, *Journal of Public Economics*, Vol. 25, pp. 1-12.

DiBona, C., Ockman, S. and M. Stone (eds.)(1999), *Open Sources: Voices from the Open Source Revolution*, O'Reilly: Sebastopol, CA.

Feller, J. and B. Fitzgerald (2002): Understanding Open Source Software Development, Amsterdam: Addison Wesley Longman.

Hars, A. and S. Ou (2002), Working for Free? Motivations for Participating in Open-Source Projects, *International Journal of Electronic Commerce*, Vol. 6 (3), pp. 25-39.

Hendricks, K., A. Weiss and C. Wilson (1988), The war of attrition in continuous time with complete information, *International Economic Review*, Vol. 29, pp. 663-680.

Johnson, J. P. (2001), Economics of Open Software, *Working Paper* (Based on authors Ph.D. Thesis, MIT), May 17, 2001.

Lerner, J. and J. Tirole (2000), The Simple Economics of Open Source, *NBER Working Paper*, No. 7600, Cambridge. (Forthcoming: Journal of Industrial Economics, 2002).

Linux Counter (2002): Online Information at [http://counter.li.org], downloaded 11.04.2002.

Netcraft (2002): Online Information at [www.netcraft.com], downloaded 19.03.2002.

Raymond, E. S. (2000a): Homesteading the Noosphere, Revision 1.22, 2000/08/24, first version 1998.

Raymond, E. S. (2000b): The Cathedral and the Bazaar, Revision 1.51, 2000/08/24, first version 1997.

Rosenberg, D. K. (2000): Open Source: The Unauthorized White Papers, B&T; IDG Books Worldwide.

Stallman, R. (1999), The GNU Operating System and the Free Software Movement, in: *Open Sources: Voices from the Open Source Revolution*, Chris DiBona, Sam Ockman, and Mark Stone (eds.), O'Reilly: Sebastopol, CA.

Torvalds, L. and D. Diamond (2001), *Just for Fun: The Story of an Accidental Revolutionary*, HarperBusiness.