Coordination practices within FLOSS development teams: The bug fixing process

Kevin Crowston[1] and Barbara Scozzi[2]

[1]Syracuse University School of Information Studies
4–206 Centre for Science and Technology
Syracuse, NY 13244–4100, U.S.A.

[2]Politecnico di Bari - Dipartimento di Ingegneria Meccanica e Gestionale
Viale Iapigia 182 70126 Bari, Italy

Abstract. Free/Libre Open Source Software (FLOSS) is primarily developed by distributed teams. Developers contribute from around the world and coordinate their activity almost exclusively by means of email and bulletin boards. FLOSS development teams some how profit from the advantages and evade the challenges of distributed software development. Despite the relevance of the FLOSS both for research and practice, few studies have investigated the work practices adopted by these development teams. In this paper we investigate the structure and the coordination practices adopted by development teams during the bug-fixing process, which is considered one of main areas of FLOSS project success. In particular, based on a codification of the messages recorded in the bug tracking system of four projects, we identify the accomplished tasks, the adopted coordination mechanisms, and the role undertaken by both the FLOSS development team and the FLOSS community. We conclude with suggestions for further research.

1 Introduction

In this paper, we investigate the coordination practices for software bug fixing used in Free/Libre Open Source Software (FLOSS) development teams. FLOSS is a broad term used to embrace software developed and released under an "open source" license allowing inspection, modification and redistribution of the software's source without charge. There are thousands of FLOSS projects, spanning a wide range of applications. Due to their size, success and influence, the Linux operating system and the Apache Web Server (and related projects) are the most well known, but hundreds of others are in widespread use, including projects on Internet infrastructure (e.g., sendmail, bind), user applications (e.g., Mozilla, OpenOffice) and programming languages (e.g., Perl, Python, gcc).

FLOSS development projects represent an interesting investigation area for researchers interested in the analysis of coordination practices within distributed teams. Many FLOSS development teams seem to benefit from the advantages of distributed work without suffering from its drawbacks, such as difficulties in coordination and

Crowston K. and Scozzi B. (2004).
Coordination practices within FLOSS development teams: The bug fixing process.
In *Proceedings of the 1st International Workshop on Computer Supported Activity Coordination*, pages 21-30
DOI: 10.5220/0002683000210030

knowledge transfer. Intriguingly, many traditional coordination mechanisms seem not to be used by FLOSS development teams [1]. Yet, “little is known about how people in these communities coordinate software development across different settings, or about what software processes, work practices, and organizational contexts are necessary to their success” [2]. Given the economic, legal and social implication, an analysis of the coordination practices of FLOSS teams could be useful to better understand the FLOSS phenomenon *per se*. As well, distributed teams of all sorts are increasingly used in many organizations. The analysis of practices adopted by FLOSS teams could be useful to managers considering adoption of this organizational form.

In the paper, coordination practices in FLOSS development processes are analyzed by adopting a process theory, i.e. we investigate which tasks are accomplished, how and by whom they are assigned, coordinate, and performed. To understand the projects’ coordination practices, we selected four representative FLOSS projects and inductively coded the steps involved in fixing various bugs as recorded in the projects’ bug tracking systems to reveal the nature of the processes adopted. We decided to examine the bug fixing process for three reasons. First, bug fixing provides “a microcosm of coordination problems” [3]. Second, a quick response to bugs has been mentioned as a particular strength of the FLOSS process: as Raymond [4] puts it, “given enough eyeballs, all bugs are shallow”. Finally, it is a process that involves the entire developer community and thus poses particular coordination problems.

To ground our discussion, we will first briefly introduce the bug fixing process, which consists of the tasks needed to correct software bugs. Crowston [3] described the bug fixing process observed at a commercial software company (to our knowledge, no description of the bug fixing process as performed in distributed teams is provided in the literature).

The process is started by a customer who finds a problem when using a software system. The problem is reported (sometimes automatically or by the customer) to the company’s response center. In the attempt to solve the problem, personnel in the center look in a database of known bugs. If a match is found, the fix is returned to the customer; otherwise, after identifying the affected product, the bug report is forwarded to an engineer in the marketing center. The assigned engineer tries to reproduce the problem and identify the cause (possibly requesting additional information from the reporter to do so). If the bug is real, the bug report is forwarded to the manager responsible for the module affected by the bug. The manager then assigns the bug to the software engineer responsible for that module. The software engineering diagnoses the problem (if she finds that the problem is in a different module, the report is forwarded to the right engineer) and designs a fix. The proposed fix is shared with other engineers responsible for modules that might be affected. When the feedback from those engineers is positive, the proposed design is transformed into lines of code. If changes in other module are needed, the software engineer also asks the responsible engineers for changes. The proposed fix is then tested, the eventual changed modules are sent to the integration manager. After approving, the integration manager recompiles the system, tests the entire system and releases the new software in the form of a patch.

The remainder of the paper is organized as follows. In section 2 we stress the relevance of process theory and explain why we adopted such a theoretical approach. The research methodology adopted to study the bug fixing process is described in Section 3. In Section 4 we describe and discuss the study’s results. Finally, in Section 5 we draw some conclusions and propose future research directions.

2 Processes as theory

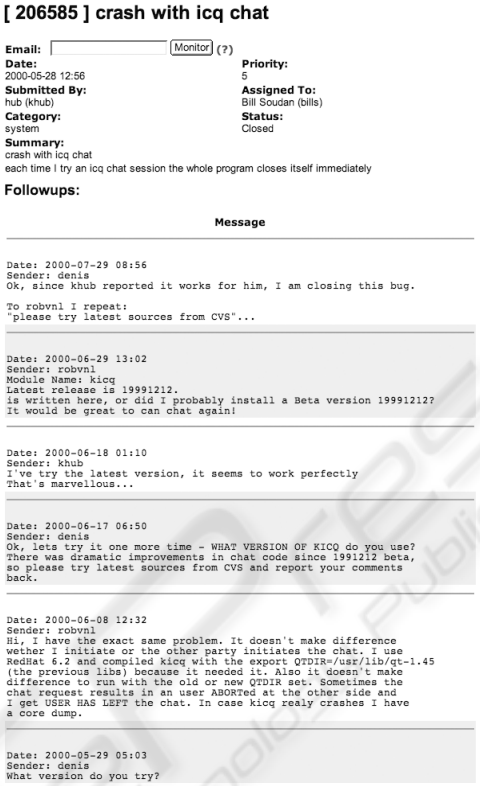
Most theories in organizational and information system research are variance theories, comprising constructs or variables and propositions or hypotheses linking them. Such theories predict the levels of dependent or outcome variables from the levels of independent or predictor variables, where the predictors are seen as necessary and sufficient for the outcomes. An alternative to a variance theory is a process theory [5]. Rather than relating levels of variables, process theories explain how outcomes of interest develop through a sequence of events [6]. Typically, process theories are of some transient process leading to exceptional outcomes, e.g., events leading up to an organizational change or to acceptance of a system. However, we will focus instead on what might be called "everyday" processes: those performed regularly to create an organization's products or services. " For example, Sabherwal and Robey [7] described and compared the processes of information systems development for 50 projects to develop five clusters of similar processes.

Kaplan [8, p. 593] states that process theories can be "valuable aids in understanding issues pertaining to designing and implementing information systems, assessing their impacts, and anticipating and managing the processes of change associated with them". The main advantage of process theories is that they can deal with more complex causal relationships than variance theories, and provide an explanation of how the inputs and outputs are related, rather than simply noting the relationship. Representing a process as a sequence of activities provides insight into the linkage between individual work and processes, since individuals perform the various activities that comprise the process. As individuals change what they do, they change how they perform these activities and thus their participation in the process. Conversely, process changes demand different performances from individuals. Information and Communication Technologies use might simply make individuals more efficient or effective at the activities they have always performed. However, an interesting class of impacts involves changing which individuals perform which activities and how activities are coordinated. The analysis is the aim of this paper.

3 Research methodology

To address our research question, how are bug fixes coordinated in FLOSS projects, a multiple case study of different FLOSS projects has been carried out. In this section, we discuss sample selection and data sources, data collection and data analysis. Projects to be studied have been selected among those available on Sourceforge, (http://sourceforge .net/), a web-based system that supports more than 75,000 FLOSS projects. Projects have access to a *home page*, a source code control system (CVS), mailing lists, a bug tracking system, software to manage activities and permanent file database. We selected several projects to study in-depth by employing a theoretical sampling strategy. First, we chose projects for which data we need for our analysis are publicly available (not all projects allow public access to the bug tracking system). Second, we chose teams with more than 8 members, since smaller projects seemed less likely to experience significant coordination problems. Finally, in the attempt to link coordination practices to project success, we tried to select more and

less successful development teams. To this aim we used the definitions of success proposed by [9], who suggest that a project is successful if it is active, the resulting software is downloaded and used and the code matures. Based on these criteria, 4 FLOSS projects were selected for analysis. A brief description of the projects is reported in Table 1. Based on the definition proposed in [9], Kicq, Gaim and PhPmyAdmin were chosen as examples of effective projects because they are active, the resulting software is downloaded and used and the code has been maturing. DynAPI was chosen as an example of a less effective project because the number of downloads, programming activity and rapidly decreased in the months leading up to the study.



[206585] crash with icq chat

Email: Monitor (?)
Date: 2000-05-28 12:56
Priority: 5
Submitted By: hub (khub)
Assigned To: Bill Soudan (bills)
Category: system
Status: Closed
Summary: crash with icq chat

each time I try an icq chat session the whole program closes itself immediately

Followups:

Message

Date: 2000-07-29 08:56
Sender: denis
Ok, since khub reported it works for him, I am closing this bug.

To robvnl I repeat:
"please try latest sources from CVS"...

Date: 2000-06-29 13:02
Sender: robvnl
Module Name: kicq
Latest release is 19991212.
is written here, or did I probably install a Beta version 19991212?
It would be great to can chat again!

Date: 2000-06-18 01:10
Sender: khub
I've try the latest version, it seems to work perfectly
That's marvellous...

Date: 2000-06-17 06:50
Sender: denis
Ok, lets try it one more time - WHAT VERSION OF KICQ do you use?
There was dramatic improvements in chat code since 1991212 beta,
so please try latest sources from CVS and report your comments
back.

Date: 2000-06-08 12:32
Sender: robvnl
Hi, I have the exact same problem. It doesn't make difference
wether I initiate or the other party initiates the chat. I use
RedHat 6.2 and compiled kicq with the export QTDIR=/usr/lib/qt-1.45
(the previous libs) because it needed it. Also it doesn't make
difference to run with the old or new QTDIR set. Sometimes the
chat request results in an user ABORTed at the other side and
I get USER HAS LEFT the chat. In case kicq realy crashes I have
a core dump.

Date: 2000-05-29 05:03
Sender: denis
What version do you try?

Fig. 1. Example bug report and followup messages (adapted from http://sourceforge.net/tracker/index.php?func=detail&aid=206585&group_id=332&atid=100332)

We collected data indicative of the success of each project, such as its level of activity, number of downloads and development status. We then collected data from the archives of the bug tracking system, the tool used to support the bug fixing process [10].. These data are useful because they are unobtrusive measures of the team's behaviors [11]. An example bug report in shown in Figure 1. In the bug tracking system, each bug has a request ID, a summary (what the bug is about), a category (the kind of bug, e.g., system, interface), the name of the team member (or user) who submitted it, and the name of the team member it was assigned to. As well, individuals can post messages regarding the bug, such as further symptoms, requests for more information, etc. From this system, we extracted data about who submitted the bugs, who fixed them and the sequence of messages involved in the fix. By examining the name of the messages senders, we can identify the project and community members who are involved in the bug fixing process. Demographic information for the projects and developers and data from the bug tracking system were collected in the period 17–24 November 2002. We examined 31 closed bugs for Kicq, 95 closed bugs for DynAPI, 51 bugs for Gaim and 51 for PhPMyAdmin.

Table 1. Four examined projects.

	KICQ	**DynAPI**	**Gaim**	**PhpMyAdmin**
Goal	ICQ client for the KDE project	Enhance the DynAPI Dynamic HTML Library	Multi-platform AIM client	Web-based database administration
Development Status	4 Beta, 5 Production Stable	5 Production Stable	5 Production Stable	5 Production Stable
License	GPL	LGPL, GPL	GPL	GPL
Open bugs /total number of bugs	26 /88	45/220	269 /1499	29 /639
Team members	9	11	9	9

For each of the selected bug reports, we carefully examined the text of the exchanged messages to identify the task carried out by each sender. By inductively coding the text of the messages in the bug tracking systems of the four projects, we identified the different elementary tasks carried out during the bug fixing process. For example the message:

> "I've been getting this same error every FIRST time I load the dynapi in NS (win32). After reloading, it will work… loading/init problem?"

Table 2. Coded tasks in the bug fixing process

1.0.0 **Submit (S)**
1.1.0 Submit bug (code errors)
1.1.1 Submit symptoms
1.1.2 Provide code back trace (BT)
1.2.0 Submit problems
1.2.1 Submit incompatibility problems (NC)
2.0.0. **Assign**
2.1.0 Bug self-assignment (A*)
2.2.0 Bug assignment (A)
3.0.0 **Analyze**
3.1.0 Contribute to bug identification
3.1.1Report similar problems (R)
3.1.2 Share opinions about the bug (T)
3.2.0 Verify impossibility to fix the bug
3.2.1 Verify bug already fixed (AF)
3.2.2.Verify bug irreproducibility (NR)
3.2.3 Verify need for a not yet supported function (NS)
3.2.4 Verify identified bug as intentionally introduced (NCP)
3.3.0 Ask for more details
3.3.1 Ask for Code version/command line (V)
3.3.2 Ask for code back trace/examples (RBT/E)
3.4.0 Identify bug causes (G)
3.4.1 Identify and explain error (EE)
3.4.2 Identify and explain bug causes different from code (PNC)
4.0.0 Fix
4.1.0 Propose temporary solutions (AC)
4.2.0 Provide problem solution (SP)
4.3.0 Provide debugging code (F)
5.0.0 **Test & Post**
5.1.0 Test/approve bug solution
5.1.1 Verify application correctness W
5.2.0 Post patches (PP)
5.3.0 Identify further problems with proposed patch (FNW)
6.0.0 **Close**
6.1.0 Close fixed bug/problem
6.2.0 Closed not fixed bug/problems
6.2.1 Close irreproducible bug (CNR) and close it
6.2.2 Close bug that asks for not yet supported function (CNS)
6.2.3 Close bug identified as intentionally introduced (CNCP)

Bug ID	Summary	Assigned to	Submitter
206585	crash with icq chat	bills	khub

Task	Person	Comments
(S)	Khub	
(V)	denis	asks what version khub is running
(R)	robvnl	reports the same problem as khub. submits information about the operating systems and the libraries (Qt/kde)
(V	denis	asks again what version both users are running
(W)	khub	reports the most recent version of kicq works
(T)	robvnl	reports version information
(C)		bug closed

Fig. 2. Coded version of bug report in Fig.1.

represents a report submitted by a user (someone other than the person who initially identified and submitted the bug). Such a user contributed to bug analysis. In particular, her message has been coded as "report similar problems". Table 2 shows the list of task types that were developed for the coding. The lowest level elementary task types were successively grouped into 6 main types of tasks, namely *Submit*, *Assign*, *Analyze*, *Fix*, *Test & Post*, and *Close*.

Each process starts with a bug submission (S) and finishes with bug closing (C). Submitters may submit problems/symptoms associated with bugs (Ss), incompatibility problems (NC) or/and also provide information about code back trace (BT). After submission, the team's project managers or administrators should assign the bug to someone to be fixed ((A); (A*) if they self-assign the bug). Other members of the community may report similar problems they encountered (R), discuss bug causes (T), identify bug causes (G) and/or verify the impossibility of fixing the bug. Bug fixing may be followed by a test and the submission of a patch (TP). This is a coordination task. However, as later explained, in the examined projects, this type of task is often neglected. In most cases, but not always, team members spontaneously decide to fix (F) the bug. Before doing that, they often ask more information to better understand bug causes (An). The bug is then closed (C). Bugs can may be closed either because they have been fixed or they cannot be fixed (i.e. they are not reproducible (CNR), involve functions not supported yet (CNS) and/or are intentionally introduced to add new functionality in the future (CNCP). Notice that the closing activity is usually attributed to a particular user.

A complete example of the coded version of a bug report (the one from Figure 1) is shown in Figure 2.

Table 3. Task occurrences and average number of tasks per projects.

Task / Project (bugs)	(S)	(Ag)	(An)	(F)	(TP)	(C)	Avr. tasks per bug
KICQ (31)	44	3	23	23	1	31	3.9
Dynapi (95)	121	0	83	57	16	95	4
Gaim (51)	56	0	65	29	12	51	4
Phpmyadmin (51)	53	1	69	49	10	51	4.4

4 Results

In Table 3, we describe the occurrences per task for the four projects and the average number of tasks to fix bugs. A χ^2 test shows a significant difference in the distribution of task types across projects ($p<0.001$). For all projects, the most common task sequence is *submit, analyze, fix, close*. In longer sequences, it is usually the *analyze* task that is repeated more times. Data about the percentage of submitted, assigned and fixed bugs both by team members and members external to the team for each project are reported in Table 4. Table 5 provides some observations of the nature of the bugs fixing process in the four projects.

5 Discussion

In the traditional bug fixing process, several tasks are coordination tasks. The search for duplicate bugs as well as the numerous forward and verify tasks are coordination mechanisms used to manage a dependency (Malone and Crowston's [12] definition of coordination). Database searching manages a dependency between two tasks that can

Table 4. The bug fixing process: Main results.

	Kicq	DynAPI	Gaim	PhpMyAdmin
Bugs submitted by team members	9.7%	21.1%	0%	21.6%
Bugs submitted by members external to the team	90.3%	78.9%	100%	78.4%
Bug assigned/self-assigned of which:	9.7%	0%	0%	2%
Assigned to team members	0%	-	-	100%
Self assigned	66%			0%
Assigned to members external to the team	33%	-	-	0%
Bug fixed, of which:	74%	60%	56.9%	96%
Fixed by team members	70%	35.1%	79.3%	89.8%
Bug fixed by members external to the team	30%	64.9%	20.7%	8.2%

have the same outcome. Forwarding and verifying tasks are coordination mechanisms used to manage dependency between a task and the actor appropriate to perform that task. In a large software company, many actors are involved, each of them carry out a very specialized task.

The above analysis provides some interesting insights on the bug fixing process for FLOSS development. Process sequences are averagely quite short (four tasks) and they seem to be quite similar: submit, analyze, fix and close. As shown in Table 3, formal task assignments are quite uncommon. Only few bugs are formally assigned. Such a coordination activity seems rather to spontaneously emerge. Based on bug description and analysis, those who have the competencies autonomously decide to fix the bug. That activity is facilitated by the supplied backtrace and analysis often undertaken by several contributors. The lack of assignment is one of main difference differentiating the process as it occurs in FLOSS development team from the traditional commercial process. As briefly described in section 1, within traditional processes assignments are coordination activities frequently carried out.

Testing is also quite an uncommon task in the logs. Most of the proposed fixes are directly posted presumably after personal testing. If no one describes the emergence of new problems with these fixes, they are automatically posted and the attendant bug closed. It is important also to note that some of the posted problems do not represent real bugs, so they are directly closed with that explanation.

A further difference is that in these projects, the process is performed by few team members (usually not more that two or three) working with a member of the larger community. Team members (usually project managers or administrators) are most involved in bug fixing. Surprisingly, only a few developers (of the team) are involved in the process. Most of the community is composed by active users who submit bugs or contribute to their analysis. However, only two or three members of the involved community are involved in fixing tasks and can be referred to as co-developers.

We also noted striking differences in the level of contribution to the process. The most active users in the projects carried out most of the tasks while most others contributed only once or twice. As expected, the most widely dispersed type of action was submitting a bug, while diagnosis and bug fixing activities were concentrated among a few individuals.

As we have few members of the team and few members of the community (co-developers) mostly involved in bug fixing and many users/members of the community (active users) mostly involved in bug submission, the organizational models proposed in the literature [13] seem to be valid for the bug fixing process. It would be interesting to further investigate if those, among the active users also involved in bug fixing, also contribute to software coding.

Also, based on the analysis of task carried out and the attendant coordination mechanisms we argue that the bazaar metaphor proposed by [4] to describe the OSS organization structure is still valid for the bug fixing process. As in a bazaar, the actors involved in the process autonomously decide the schedule and contribution modes for software development, making a central coordination action superfluous.

As apparently less successful, we expected to find that DynAPI had a smaller active user base than the other projects. However, as noted above, data shows the opposite. It seems likely that our estimation of the success of the two projects based on activity levels is mistaken, or at least an over-simplification. We plan to further ex-

plore this hypothesis by examining a larger number of projects (e.g., to examine the change in the population over time).

Table 5. Observed characteristics of the bug fixing processes in the four projects.

	Kicq	**DynAPI**	**Gaim**	**PhpMyAdmin**
Min task sequence	2	2	2	2
Max task sequence	6	12	6	11
Uncommon tasks	Bug assign-ment/ 3	Bug assign-ment/ 0	Bug assign-ment/ 0	Bug assign-ment/ 1
Community members	18	53	23	20
Team mem-bers' par-ticipation	2 of 9	6 of 11	3 of 9	4 of 10
Most active team mem-bers Role/ name	Project mgr denis Developer davidvh	Admin rainwater Ext member dcpascal also active	Admin-developer warmenhoven Developer robflynn	Admin-developer loic1 Admin-developer lem9
Max posting by single community member	2	6	4	3
Not fixable bug closed	8	5	5	-

6 Conclusions

We investigated the coordination practices adopted within four FLOSS development teams. In particular, we analyzed the bug fixing process, which is considered critical for FLOSS' success. The paper provided some interesting results. The process is mostly sequential and composed of few steps, namely *submit, analyze, fix and close*. Second, the process seems to lack traditional coordination mechanisms such as task assignment. As a consequence, labour is not equally distributed among process actors. Few contribute heavily to all tasks whereas the majority just submit one or two bugs. Third, the organization structure involved in the process resembles the one proposed in the literature for the FLOSS development process. Few actors (core developers), usually team project managers or administrators, are mostly involved in bug fixing bugs. Most of the involved actors are instead active users, who just submit bug reports. In between are few actors, external to the team, who submit bugs and contribute to fixing them. No evident association was found among coordination practices and project success.

The paper contributes to fill a gap in the literature by providing a picture of the coordination practices adopted within FLOSS development team. Besides, the paper proposes an innovative research methodology (for the analysis of coordination practices FLOSS development teams) based on the collection of process data by electronic archives, the codification of message texts, and the analysis of codified information supported by the coordination theory. However, the results are based on few projects, so further analyses are necessary to validate them. In the future, we intend to deepen the knowledge about the coordination practices adopted by the four projects by directly interviewing some of the involved actors.

References

1. Mockus, A., R.T. Fielding, and J.D. Herbsleb, *Two Case Studies Of Open Source Software Development: Apache And Mozilla.* ACM Transactions on Software Engineering and Methodology, 2002. **11**(3): p. 309–346.
2. Scacchi, W. *Software Development Practices in Open Software Development Communities: A Comparative Case Study (Position Paper).* 2002.
3. Crowston, K., *A coordination theory approach to organizational process design.* Organization Science, 1997. **8**(2): p. 157–175.
4. Raymond, E.S., *The cathedral and the bazaar.* First Monday, 1998. **3**(3).
5. Markus, M.L. and D. Robey, *Information technology and organizational change: Causal structure in theory and research.* Management Science, 1988. **34**(5): p. 583–598.
6. Mohr, L.B., *Explaining Organizational Behavior: The Limits and Possibilities of Theory and Research*. 1982, San Francisco: Jossey-Bass.
7. Sabherwal, R. and D. Robey, *Reconciling variance and process strategies for studying information system development.* Information Systems Research, 1995. **6**(4): p. 303–327.
8. Kaplan, B., *Models of change and information systems research*, in *Information Systems Research: Contemporary Approaches and Emergent Traditions*, H.-E. Nissen, H.K. Klein, and R. Hirschheim, Editors. 1991, Elsevier Science Publishers: Amsterdam. p. 593–611.
9. Crowston, K. and B. Scozzi, *Open source software projects as virtual organizations: Competency rallying for software development.* IEE Proceedings Software, 2002. **149**(1): p. 3–17.
10. Herbsleb, J.D., et al., *An Empirical Study of Global Software Development: Distance and Speed*, in *Proceedings of the International Conference on Software Engineering (ICSE 2001)*. 2001: Toronto, Canada. p. 81–90.
11. Webb, E. and K.E. Weick, *Unobtrusive measures in organizational theory: A reminder.* Administrative Science Quarterly, 1979. **24**(4): p. 650–659.
12. Malone, T.W. and K. Crowston, *The interdisciplinary study of coordination.* Computing Surveys, 1994. **26**(1): p. 87–119.
13. Cox, A., *Cathedrals, Bazaars and the Town Council*. 1998.