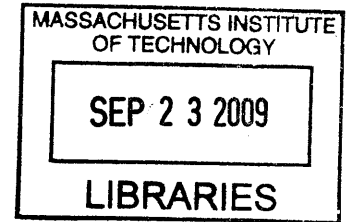


# Managing projects utilizing self-managed teams and managerial toolkits

by  
**Praveen Mathur**  
M.B.A, Rutgers University  
B.E., Bangalore University



Submitted to the System Design and Management Program  
in Partial Fulfillment of the Requirements for the Degree of

**ARCHIVES**

**Master of Science in Engineering and Management**  
at the  
**Massachusetts Institute of Technology**

May 2009

[Signature]

© 2009 Praveen Mathur. All Rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium not known or hereafter created.

[Signature]

Signature of Author \_\_\_\_\_

[Signature]

Praveen Mathur  
System Design and Management Program

Certified by \_\_\_\_\_

[Signature]

Eric A. von Hippel, Thesis Supervisor  
T Wilson (1953) Professor in Management  
MIT Sloan School of Management

Accepted by \_\_\_\_\_

[Signature] Patrick Hale Director,  
System Design and Management Program

## **Acknowledgements**

I would like to thank Prof. Eric von Hippel, my advisor, for his guidance, support and encouragement during the course of this thesis. His direction and valuable insights challenged me and gave me an opportunity to learn during the entire process.

I would like to thank my wife Deepti for the great sacrifices she made during the past two years. Her support and patience during this entire time gave me the opportunity to follow my dream and I feel very fortunate to have her by my side. To my son, Arnav, without his endless love and understanding this journey would have been impossible.

I would also like to acknowledge the SDM program and my fellow classmates for making this a thoroughly enjoyable educational experience that I will cherish forever.

## TABLE OF CONTENTS

Acknowledgements .....	2
Abstract .....	5
Motivation .....	6
PART I .....	7
Why Is Project Management a Critical Issue? .....	7
Why Is Project Management so Difficult? .....	7
Role of Organization .....	11
Role of Project Manager .....	11
Responsibilities of a Project Manager .....	12
Skills required for Project Manager .....	13
Project Essentials .....	13
Project Teamwork .....	14
Project Plan .....	15
Leadership and management elements .....	16
Communication and Information sharing .....	17
Applying Fayol’s model to Project Management .....	18
Project Management Toolkit .....	18
Why projects fail? .....	21
PART II .....	23
Self-Directed Work Teams (SDWT) .....	23
Communication in Self Directed Work Teams .....	24
Change Management .....	24
Project Scope and control .....	24
Leadership in the Change Process .....	25
Free and Open Source Software Development Model .....	26
Background: Free and Open Source Software .....	26
Introduction .....	26
Formation and Growth of Open Source communities .....	27
Open source Beliefs, Values and Norms .....	28
Motivation to participate .....	29
Communication & Information Flow .....	30
Common Vocabulary .....	31
Project Control .....	32
Project Team .....	33
Project Teamwork .....	35
Coordination .....	36
Mechanisms to Coordinate Massive Amount of Individual Efforts .....	37
Project Leadership .....	40
Balancing Anarchy with Control .....	41
Quality .....	42
Comparing Traditional Project Development with Open Source Project Development .....	42
Application of an Project Management Toolkit in Open Source .....	44
PART III .....	49
Open Source Project Management at Enterprise Level .....	49
Introduction and Background .....	49
Analyzing Motivations on the Firm’s Level .....	51
IBM’s Motivation to Participate .....	52
Business Models .....	52

IBM's Business Model.....	54
Leadership .....	55
Licensing.....	56
Participation .....	56
Project Visibility .....	57
Building Community.....	57
Automating Project Management .....	60
Case Study: Analysis of Linux based desktop client .....	62
Applying Fayol's principles and project management toolkit.....	63
Planning .....	63
Organizing.....	67
Controlling .....	69
Coordination.....	71
Commanding.....	72
Conclusion .....	75
References .....	77

## **Abstract**

Project Management is an essential function in most software companies today. With increasing complexity and inter connectivity between software projects, it is not surprising that managing such large scale development projects can be expensive and extremely time consuming for the sponsoring organization. In large scale complex software projects the project manager has to ensure that enough resources are allocated to the project and foster an environment of communication and teamwork, but accomplish all this with little authority over the project team. This traditional approach to managing project relies on the skills and experience of a project manager but is fraught with pitfalls that can lead the project in the wrong direction if corrective action is not taken in a timely fashion. Any misstep during the project lifecycle due to scope creep or miscommunication can ultimately push the project to miss deadlines or be over budget.

Another alternative approach to software development is using self organizing teams. Free/Open Source software development approach uses the concept of self organizing teams to collaborate at a global scale using communities of developers. The F/OSS paradigm, based on cooperation and collaboration among developers from all over the world, introduces methodologies and development models different from those usually utilized within the proprietary software industry. In it, communities of developers and users share a common interest in a project and interact regularly with one another to share knowledge collaboratively solve a common problem. This approach reduces the overhead required in communication and coordination by sharing information with all members of the project and relies on automating some of the essential elements of the project. The thesis synthesizes the use of automated tools as it applies to the project toolkit and uses case studies to understand how F/OSS development approach can be used in organizations to reduce project's dependence on a project manager.

Thesis Supervisor: Eric A. von Hippel

Title: T Wilson Professor in Management, MIT Sloan School of Management

## Motivation

I am a project manager for a large scale complex technical project at IBM and after having spent more than three years managing multiple releases of the application I have come to realize that beyond delivering the project on time and within budget, I am mainly the conduit of information between one group and the other. The information related to the project is held at different levels in the hierarchy and as the project manager my primary role is to assimilate all relevant information from the different levels of hierarchy and allocate resources to act on it. This highly structured approach has helped IBM create standardized processes and raise managerial talent. However this approach also leads to inefficient use of information, resources and creates bottlenecks in the system. Using the working paper<sup>1</sup> and applying some of the key finding from it to my project, I am convinced that much of the work project manager is responsible can be done by self-organizing teams using the right managerial toolkit.

I would like to use my current project as a case for analyzing and understanding what are some of the key responsibilities of a project manager and how can the role be substituted with alternative approach.

Looking at the traditional project management approach and using the working paper as viewing lens, I would like to understand how project activities can be replaced by managerial toolkit of some kind. My view is that the technical aspect of the project management like coordination can be automated by using automated tools. However, there are other issues such as project leadership, requirement gathering and project control that require a deeper understanding. During the project lifecycle the demands of the project are different and understanding what tools can be used under certain circumstances that reduce the project overhead, will greatly improve management practices used in companies today.

Lastly, I would focus on issues such as company culture, reward and compensation that directly affect the project externally and can have adverse effects on the project if it is not addressed appropriately. Utilizing self organizing teams will also require complete shift from existing processes that impact employee motivation, compensation and evaluation.

# PART I

## Why Is Project Management a Critical Issue?

*It is best to do things systematically, since we are only human and disorder is our worst enemy.*

Hesoid,  
8th century B.C.

The future of most organizations depends on successful projects. Whether to survive or to sustain market leadership; projects are the key in the new era of world competition. They are essential to the vital aspects of any business for:

- Developing new products and services that meet customer needs.
- Shortening time-to-market for new developments
- Improving efficiency and productivity.
- Strengthening competitive positions in national or world markets.

Project management is the key and essential component through out the product cycle, from inception through completion. Large scale projects are complex as they require high degree of collaboration and planning. Complexity is further increased by limited resources and tight schedules. Internal and external confusion leads to poorly implemented project leading typically to waste, inefficiency, and costly errors. Many projects fail by repeating either the technical or business mistakes of others, or simply by not implementing Lessons Learned from other projects. To succeed, the project team needs training and support from upper management. Unfortunately, relatively few companies comprehend the full power of project management.

### Why Is Project Management so Difficult?

Several major challenges add to the difficulties in managing a project, mainly the inherent temporary nature of a project, demanding business environment, misguided management, and role of projects in non-traditional organizations. Managing projects becomes increasingly demanding due to uncertainty added to the project due to changing business environment, resources in the organization and culture in an organization.

External conditions like business cycle, general economic market conditions and globalization, forces project managers to look outside of the organization to continuously adjust to rate of change. In addition, Staffing challenges contribute significantly to the difficulty of project management.

Therefore, selecting the right project manager is critical to project success. The project manager must fulfill the requirements of the customer or user, must answer to senior management by generating a fair return on investment, and must provide a stimulating, positive work environment for the project team. Project managers must be skilled in technical, business, legal, financial, and personnel matters. Above all, project managers must have leadership qualities. They must be able to deal with all levels in the organization, from stakeholders, to geographically distributed workforce.

Project management is made difficult by the urgent need for a newly formed project group to achieve proficiency as a team. The temporary nature of project teams often brings together people who have little or no experience working with one another. Furthermore, people who are attracted by project assignments are generally motivated by intangible factors such as the work itself or the technical challenge, rather than being part of the team. The newly formed group usually includes specialists who have excelled as individual contributors. This independence—both managerial and technical—conflicts with the interdependence required for teamwork.

The evolution of a typical project, such as a new product or new business development, usually follows three steps

**Proposal:** Start of the project often in a functional organization with a proposal in response to a request

**Development:** A cross functional team formed in a Project Organization

**Production:** Standard production or operational support ,returned to functional organization

Traditional management approaches deal well with the proposal and production stages of the project. These emphasize that the manager is responsible for a productive work environment and a consistent climate including:

- Stable work environment.
- Minimum of conflict among employees.
- Ambitious employees driven to be their personal best by perks and personal competition.



- Simple, clear reporting structure and organization.
- Responsibility matched with authority.
- Maximum creative freedom.

By contrast, project management is more narrowly focused on the specific objectives of the project at hand. Like task forces and other temporary groups, project teams are drawn from various long-term permanent organizations. But unlike other temporary groups, projects are managed to a defined plan including a budget, schedule, and specific output—usually a product or service.

Projects are requirements driven. The customer or user defines the requirements to be met by the project team. Unlike the activities that occur wholly within traditional, functional organizations, project work depends on lateral flow. Therefore, projects lend themselves to some form of matrix organization such as shown in Figure 1. Horizontal dotted line interfaces need to be encouraged and strengthened rather than used reluctantly as exceptions to the linear chain of command.

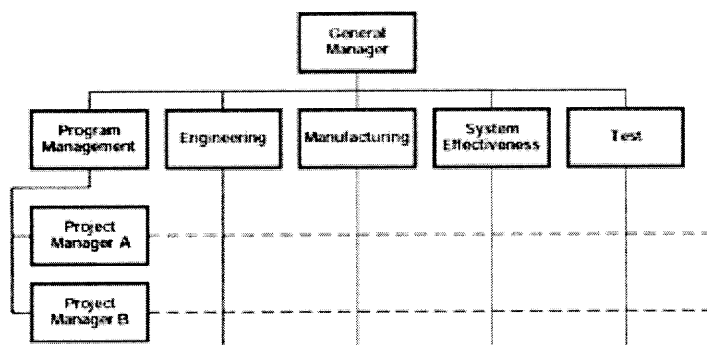


Figure 1: Typical matrix organization

Henri Fayol's<sup>[ii]</sup> management principles are widely accepted views of what is necessary for effective general management. His five elements can be viewed as viewing lens to model the nature of project management.

- Planning
- Organizing
- Coordinating
- Commanding
- Controlling

These combined with Fayol's 5 elements, his 14 principles provide a structure for project management direction. Most management models are based on Henri Fayol's 5 elements and 14 principles.

1. Division of work.
2. Responsibility matched to authority.
3. Discipline.
4. Unity of command.
5. Unity of direction.
6. Subordination of the individual's interests to the general interest.
7. Remuneration of personnel.
8. Centralization.
9. Scalar chain (line of authority).
10. Order.
11. Equity.
12. Stability of tenure of personnel.
13. Initiative.
14. Esprit de corps.

Project managers perform the traditional management functions of planning, organizing, coordinating, directing, and controlling. Four principles among Fayol's principles, namely responsibility matched to authority, unity of command, scalar chain of command, and stability are often omitted because some believe they are not desirable or cannot be achieved in the project environment.

Project management is further complicated because of incorrect sequencing of project events. Some elements of project management are sequence-driven and some are situation-driven management. Viewing the project solely as a sequence of events paints a distorted and biased image of the overall project management process.

## Role of Organization

For a project to be successful the organization plays an important role, organizations that are project driven are more likely to implement the right mechanism for a project to succeed. This may come in terms of the creating right project environment, the project team or selecting an appropriate project manager. In project-driven organization, such as construction or aerospace, all work is characterized through projects, with each project as a separate cost center having its own profit-and-loss statement. In the non-project-driven organization, such as low-technology manufacturing, profit and loss are measured on vertical or functional lines. Project management in a non-project-driven organization is generally more difficult because projects may be few and far between, not all projects have the same requirements and therefore cannot be managed identically. A lack of proper training and understanding leads to mismanagement of projects which in turn has an adverse effect on the project outcome.

## Role of Project Manager

Given how critical projects are to organization, the project manager plays an important role in implementing a project. The project manager's roles are broad—like those of general managers—going from administration to technical to leadership. But the focus is shorter range than that of a line manager who must manage the long-term strength of the organization. By contrast, the project manager should be correctly focused on the relatively short-term results of the project. The project manager must convert the inputs in the form of capital, material, equipment, personnel etc into output of products or services within the allocated timeframe and produce a benefit to the organization. The project manager is responsible for coordinating and integrating activities across multiple, functional lines. The integration activities performed by the project manager include:

- Integrating the activities necessary to develop a project plan
- Integrating the activities necessary to execute the plan
- Integrating the activities necessary to make changes to the plan

In order to do this, the project manager needs strong communicative and interpersonal skills, must become familiar with the operations of each line organization, and must have knowledge of the technology being used. The project manager also plays the part of interface manager between the traditional organization structure, the functional organization, senior management, customers and the project team. The line manager interface is required for personnel allocation and senior management involvement is required to provide guidance to the overall project. This requires the project manager to

report status of the project timely and accurately to stakeholders and sponsors for making effective project decisions.

Role	Complications
Manage the project to the Project Cycle.	Meet an aggressive schedule.
Balance technical schedule, and cost performance	Implement state-of-the-art technology
Solve problems expeditiously as they arise.	Perform within the budget by using limited funds and resources.
Inspire and motivate the entire team.	Optimize the mix of dedicated, shared, and contract personnel.

### Responsibilities of a Project Manager

In many organizations a project manager is viewed as the general manager for a project. To be effective as a project manager, an individual must have management as well as technical skills. The project manager has the overall responsible and accountable for the overall project but has little authority. Broad responsibilities increase the need for information and collaboration, forcing the project manager to cut cross organizational lines, just like a general manager.

The major responsibility of the project manager involves planning - considering trade-offs and resolving conflicts. In most cases, the project manager provides overall or summary definitions of the work to be accomplished, but the line managers (the true experts) do the detailed planning. Although project managers cannot control or assign line resources, they must make sure that the resources are adequate and scheduled to satisfy the needs of the project, not vice versa. As the architect of the project plan, the project manager is responsible for

- Complete task definitions
- Resource requirement definitions
- Defining milestones
- Definition of end-item quality and reliability requirements
- Performance measurement

- Establishing the project vocabulary
- Pursuing opportunities and managing risk
- Ensuring project controls
- Ensuring visibility for the project
- Determining the content, frequency, and the detail of project status reviews
- Executing timely corrective action to correct variances from the plan

Project managers are responsible for project administration and, therefore, must establish the team policies, procedures, rules, guidelines, and directives—to control the project. Establishing project administrative requirements is also part of project planning.

### Skills required for Project Manager

Peters and Waterman<sup>[iii]</sup> report a high correlation between project success and the leadership qualities and/or authority level of the project manager. In many types of projects, leadership qualities are more important than authority level. It is essential that the project manager operate as a manager/leader rather than just as a coordinator/monitor. He or she must have well-defined, business inter-relationships with the support managers participating in the project. The complex interpersonal relationships require that the project manager be selected more on the basis of behavioral (e.g., negotiating and leadership) skills than on technical skills. However, the project manager should be “conversant” in the project domain and knowledgeable of the system engineering process. In addition to the skills identified, the project manager should exhibit the following capabilities:

- Leadership and team building.
- Entrepreneurial and business acumen.
- Balance between technical and business capabilities (generalist).
- Planning, organizing, and administration abilities.

### Project Essentials

All projects have a cycle. The cycle usually has Periods (such as Plan, Develop, and Production), and Phases within the periods (such as Concept Definition and Verification). Project manager acts under

different capacity in each of these phases by decomposing the higher levels tasks and assigning them to the team. There are proactive elements inherent in a project that requires planning as well as reactive element of the project that requires control and adjustment. A project also depends on these four essential elements that are interwoven during the entire project cycle

- Common vocabulary
- Teamwork.
- Project plan
- Leadership and management elements

Common vocabulary and teamwork are seen as perpetual properties of a project, while the project cycle and management elements embody the sequential and situational properties.

**Common Vocabulary:** Describes the inherent norms and practices in a project. These form the basis of the project work and helps resolve conflict between team members. It also provides a guideline of what is acceptable practice during the course of the project. These are unspoken norms that are conspicuously absent from the actual project plan but are very heavily impacted by the prevailing culture in the organization.

### Project Teamwork

Teamwork is often defined as working together to achieve a common goal and encompasses the following fundamentals as well.

- **Common goals:** Building teamwork begins with clearly defining the group objectives and outlining the various roles and responsibilities required to accomplish those objectives. Gaining consensus at all levels of project is important to reveal and resolve conflicts. A common goal therefore, is essential in planning, measuring and evaluating a project since all team members need to be on the same page to perform effectively in the project environment.
- **Acknowledged interdependency and mutual respect:** In the team environment, mutual respect, relationships, roles, and interdependencies are inextricable and need to be developed in concert with other activities in order to resolve conflicts and remove ambiguity around roles and goals. For interdependencies in the project environment to be recognized, there must be an acceptance of, and respect for, the roles that must be filled by each team member

- A common code of conduct: The most obvious conduct issues are usually well-documented by company or government policies. But they may not be well known to all team members. Adding contractors and customers in the project requires the project manager to ensure there are no potential problems and remove ambiguity around interdependency.
- Shared rewards: Shared recognition for all contributing team members on a successful project is often far more important than a cash bonus. People are motivated to do a good job and to cooperate with one another when they are confident that their individual, as well as team, performance will be publicly recognized and appreciated by their peers and their management.
- Team spirit and energy: Independent thinking alone is not suited to the interdependent project reality. Putting the team ahead of oneself, however, does not mean the elimination of strong “pacesetters.” A project manager’s responsibility is to understand team members and extract the collective energy of the team to achieve a common goal, at the same time channel assertive personalities creatively as to not dominate the team. This sometimes involves subtle leadership techniques.

Effective teams <sup>iv</sup> share several common characteristics. They can articulate their common goal which they are committed to achieve. They acknowledge their interdependency coupled with mutual respect. They have accepted a common set of boundaries on their actions—a common code of conduct for the performance of the task. They have accepted the fact that there is one reward they will all share. Add team spirit and a sense of enjoyment.

### Project Plan

The project plan identifies the requirements, defines scope of work, identifies resources and presents a schedule visually for team members and stakeholders. This is the phase where most of the planning and scheduling work is done by the project manager. The essential items that go into a project plan are

- Requirements: Project requirements start with what the user really needs and end when those needs are satisfied. In the end-to-end chain of specifications, there is an ongoing danger of misunderstanding and ambiguity. The project customers control the definition of requirement, and so, it is crucial for the project manager to correctly assess and educate users of what is technically feasible in the scope of project.
- Risk: The project manager is also responsible for managing expectations of the users upfront and responsible for communicating alternatives to mitigate risk inherent in the project. On an ongoing

basis the project manager's job is to proactively identify risk against the project baseline and communicate the risk for corrective action by the stakeholders.

- Work Breakdown structure: The work breakdown structure depicts the subassemblies and components of the high level requirements. It illustrates the way the project will be integrated, assigned and statused. This forms the basis of project planning work including budgeting, scheduling, cost allocation and reporting.
- Project Baseline: Given the set of requirements, allocated resources and time constrains the project baseline serves as a benchmark for performance of the project. The project manager must ensure that the plan is updated at all times, to communicate status to senior management and also to seek additional resources if things fall behind.
- Project control: While monitoring the cost and project status is control; it is hardly project control without proactive action. Controls can be in the form of funding, personnel conduct or quality. A common technique used in software development used the gated approach to controlling a project. With each successive step the project moves through different stages or gates satisfying the appropriate standards.
- Status or Measurement: Project Statusing is the timely and comprehensive measurement of project progress against the plan to determine the potential seriousness of any variances left uncorrected. The main objective is to identify variances that require corrective action in order to recover to plan. Cost, schedule and accuracy provide one set of discreet measurement. In addition, reviews and checklists measure the non discreet elements of the project.

### Leadership and management elements

Leadership and management elements are embedded in all stages of a project. In the context of project management, leadership represents the ability to inspire—to ensure that project members are motivated—on both the individual and the team level. Three primary aspects of project leadership:

- Situational leadership—the relationship of leadership to management.
- Techniques for inspiring and motivating individual and team performance.
- Style—determining and communicating your leadership style.

Other key elements of managing a project include



- Vision
- Creating a conducive project environment,
- Problem solving,
- Conflict resolution,
- Communication
- Interpersonal skills
- Motivating others
- Delegation
- Coaching

### Communication and Information sharing

Beyond the elements described above, communication in a project team is an essential component to successful implementation of a project. The project manager plays a central role in communicating with all the participants of the project - from internal team members, management and external customers. Communicating above and beyond the required status, the project manager is responsible to set expectations with stakeholders in the form of project baseline, hold regular meetings and keep users abreast with the changes in the project. Increased communication with all parties reduces scope creep and creates one view of the expected outcome. Project manager is also the custodian of information within a team and is responsible for sharing business and technical information with team members. A breakdown in communication is one of key reasons why Brooks<sup>y</sup>'s law applies to project development.

## Applying Fayol’s model to Project Management

Applying Fayol’s model to the project management process gives us a better understanding of the process in a project setting. These elements can be applied to every phase of the cycle and identify those indispensable responsibilities of project management that are too often misunderstood, minimized, or ignored in practice.

<b>Fayol (1916)</b>	<b>Project Environment setting</b>
Organizing	Project Team
	Project Teamwork
Planning	Project Planning
	Requirement
	Opportunity and Risk Management
Controlling	Project Control
Coordinating	Project Status
	Project Communication
Commanding	Corrective Action
	Leadership

Table 2: Applying Fayol's elements to Project management

Project Control embodies those techniques that help ensure that events happen as planned, and that unplanned events do not happen (proactive), whereas the three variance control elements define the means for detecting and correcting unplanned results (reactive).

### Project Management Toolkit

Drawing on the information above and using Fayol’s principles as a guide, for a project to succeed the following elements are necessary in any project management. Creating a toolkit that embodies all these elements will ensure a smoother project management process. The toolkit does not take into account the skill set of the team or the requirements for a project. Given that the requirement process is clearly defined and the team members have the essential skills a project can be implemented with these elements in a toolkit<sup>vi</sup>.

*Project Requirements* covers both the creation and management of requirements. It includes requirement identification, substantiation, concept selection, decomposition, definition, integration, verification, and validation. Techniques and tools include decomposition analysis and resolution, requirements traceability, accountability, modeling, and others. This element is situational rather than sequential since new requirements are apt to be introduced at almost any point in the project to be managed concurrently with the requirements driving development.

*Project Teamwork* considers the strengths and deficiencies of various project structures for example, how each resolves accountabilities, responsibilities, and promotes teamwork and communications. There are many options including matrix, integrated product teams, and integrated project teams. This element is personnel-independent and provides the basis for selecting and changing the structure appropriately as the project progresses through project cycle phases from concept to deactivation.

The *Project Team* element addresses staffing the organization. Selection criteria consider character traits, qualifications, and the specific skills demanded by the challenges of each project phase. Competency models that include necessary attributes and qualifications should form the basis of selection for key positions such as the project manager. The best management approach may require that some key players be changed as the project progresses through the cycle.

*Project Planning* starts with the team's conversion of project requirements into team task authorizations including delivery schedules and resource requirements. Plans must be kept current, reflecting new information and actual progress. The planning process should include both manual and computer tools which support the development of the best tactical approach for accomplishing project objectives consistent with the project cycle constraints.

*Opportunity and Risk* management is an important part of the overall planning process, yet it is often ignored. This element encompasses the identification, evaluation, and management of both opportunities and their associated risks. It includes techniques for determining and managing the planned actions to enhance the opportunities and to mitigate the risks. Opportunities and risks may be identified at any point in the project cycle, so the techniques and tools of this element must be applied perceptively as the project progresses through the cycle

*Project Control* Controlling the project is necessary to ensure that planned events happen as planned and that unplanned events don't happen at all. Controls must be proactive rather than reactive Categories of controlled processes may include security, safety, requirements, manufacturing processes, software development environment, schedule, cost, and so on. Reactive control consists of corrective action initiated in response to unacceptable variances. Many projects fail when control systems are not established or are circumvented.

*Project Communication* encompasses all of the techniques used by the project team and stakeholders to gather data and disseminate information so as to ensure that the project team communicates effectively and is informed as necessary about relevant project activity. It includes manual as well as electronic techniques such as voice mail, e-mail, and video conferencing. The visibility system and associated techniques must be designed to serve the active project phase, the organizational structure, and geographic complexity.

*Project status* is not simply activity, but comprehensive measurements of performance against the plan to detect unacceptable variances and determine the need for corrective action. Status should encompass schedule, cost, technical, and business progress. The evaluation and measurement should also include the rate of change of the variance if not corrected. Earned value and other systems are included in this technique and tool set.

*Corrective Action* is the culmination of variance management and emphasizes that reactive management is necessary and proper for effective project management. Corrective Actions are the actions taken to return the project to plan and usually take place during Project Statusing, or shortly thereafter.

*Project Leadership* is the most important of the 10 project management elements. Leadership is the mortar that holds together all other elements of project management and ensures that all the others are properly implemented and effectively used. It represents the ability to inspire—to ensure that project members are motivated on both the individual and team level to deliver as promised within the desired project management culture. Leadership depends on the skillful application of techniques such as handling different personalities and maturity levels, and team composition and rewards. If the team members are fully trained in the worth of the elements and are believers in the process, then the need for strong leadership is reduced.

Why projects fail?

Given the depth and breadth of research that has been conducted on the topic of project management, it is still common place for projects to fail in an organization. From personal experience and from feedback from project managers at IBM<sup>vii</sup> these are some of the most common reasons why projects fail in an organization.

<b>Project Failure due to</b>	<b>Reality</b>	<b>Applicable Toolkit Element</b>
Fictitious Project Schedule	<ul style="list-style-type: none"> <li>• Customers or executives typically dictate schedules</li> <li>• Adjust scope or resource (\$) to fit the schedule</li> </ul>	Project Planning
Changing Requirements/scope creep	<ul style="list-style-type: none"> <li>• The Scope changed but the schedules did not</li> <li>• Code changes were made but were not communicated to the affected team members</li> <li>• Changes were made on verbal request but were not reflected in documentation or change request. This resulted in code out of compliance with design document</li> <li>• Balance stone walling ( pretending nothing can be changed) and approving all proposed changes which causes major control problems</li> </ul>	Project Requirement Project Communication Project Control Corrective action
Inaccurate Status Reports	<ul style="list-style-type: none"> <li>• Project status reports were written to please customers and executives and do not reflect the actual state of the project</li> <li>• Items submitted by perform resources are changes to meaningless drivel by Project Managers who don't understand what has been written</li> <li>• The Project Managers appears to take credit</li> </ul>	Opportunity and Risk Project Status Project Control Project Leadership Project Teamwork

	for activities completed in status reports when the are actually performed by other team members	
Unrealistic expectations	<ul style="list-style-type: none"> <li>• Customers have unrealistic expectations on what will be delivered and its performance</li> </ul>	Project Planning Project Status Project Leadership
Lack of Communication and teamwork	<ul style="list-style-type: none"> <li>• The PM does not encourage teamwork and cooperation between the team members</li> <li>• PM meets with the customer but doesn't communicate back to the perform team</li> <li>• PM is in touch with what's really going on</li> <li>• Does not call regular meetings for the team to know what's going on</li> </ul>	Project Communication Project Leadership Project Teamwork
Lack of PM leadership skills	<ul style="list-style-type: none"> <li>• The PM understands the WWPMM but does not have the necessary leadership skills</li> <li>• The PM does not have the business or technical understanding needed to lead the team</li> </ul>	Project Leadership
Wrong resources	<ul style="list-style-type: none"> <li>• The PM brought the wrong talent into the project</li> <li>• The PM does not address poorly performing resources or those that are detracting from the effectiveness of the team</li> </ul>	Project Team

Table 3: Some common causes of why projects fail

These common problems suggest that a project is a complex and critical part for any product development process. Ultimately a project manager bears the responsibility, even with limited authority, to manage all aspects of the project to make it a success. Even in project driven industry and organizations, projects fail and suggest that project management is still an art rather than science.

## **PART II**

As discussed in PART I of the thesis, I have attempted to establish that the role of a project manager and the project team is crucial to the project's success. However, recently with adoption of agile development techniques using the concept of Self-Directed work teams has come into the lime light. Understanding how Self Directed teams work and applying some of the best practices to project environment can help organizations achieve the desired results without the overhead of managing. Reducing the overhead that comes with managing project life cycle can greatly reduce cost and improve efficiency in bringing new products to market sooner.

### **Self-Directed Work Teams (SDWT)**

A self-directed work team is a small group of people empowered to manage themselves and their daily work. Such teams are formal permanent organizational units. Team members typically not only handle their current job responsibilities, but also plan and schedule their work, manage production, solve problems, and share leadership responsibilities. A self-directed work team usually performs many traditional support functions along with routine production or service.

SDWTs exhibit the following characteristics:

- The team performs specific tasks.
- Team members are multi-skilled.
- Team members are interdependent.
- Team members are in control of their daily, monthly, and yearly goals.
- The team's focus is on team results, not individual results.
- Team members rotate tasks because extensive cross-training enables them to perform many different jobs.
- Management clearly defines the team's boundaries for task responsibilities and authority (for example, hiring or firing members, or redefining work instructions, may be outside of the team's authority).
- The team monitors and controls both its work quantity and quality.
- Each team understands the need for effective teaming behaviors.

## Communication in Self Directed Work Teams

Information sharing is crucial in breaking paradigms that block effective employee-management relationships and helps establish an open and honest communication environment which forms the foundation for effective teamwork. The goal is a trust-based partnership with employees, and information sharing that reinforces that trust. Sharing information openly with team enables team members to be open to change and allow the team to adopt to change better. This also enhances the two-way communication which is necessary to reap the benefits of the self-directed work team concept. Sharing information openly results in

*Common goals or Vision:* When everyone has the same access to information, everyone in the team understands the common goal.

*Employee Empowerment:* Employees better understand what is expected out of them and reduces ambiguity around role and accountability. Clearly communicating responsibility and assigning accountability, a team member is empowered and more involved in the project

*Rewards and recognition:* Recognizing and rewarding employees openly allows for healthy competition in the team. Public recognition of a job well done goes a long way than simple remuneration, since it builds confidence in the team adds to the reputation of an employee.

## Change Management

Change should be something that people do, not something that is done to them. People are more comfortable with change when they participate in planning for or implementing it. Since sufficient investment in resources is required to implement change, by allowing employees to participate it gives the employees a sense of control and reduces their fear of the coming change. Providing resources for employees to coalesce around the change improves employee-employer relationship and reduces the stress imposed by a limited resource environment. In addition, making training and development available to team members ensures that effective change takes place – since team members now have the necessary tools required to adapt to change.

## Project Scope and control

With a common purpose and a clearly stated goal, self directed work teams can assess requirements better and convert those into major milestones of the project. Open communication allows for better participation from the team which in turn builds trust in the team and eventually reduces the overhead



associated with coordinating project tasks. This reduces the amount of control needed on the project allowing for better project tracking and decision making.

### Leadership in the Change Process

Management commitment and leadership are overwhelmingly the most critical factors in any cultural change. Cultural change begins with the personal commitment and the active involvement of senior management.

For self directed teams to function, management commitment and leadership are critical. Leadership in the team is needed to set the right environment that fosters open dialogue and encourages knowledge sharing. It sets a framework and an atmosphere for learning, open communication, trust building in the team and promoting success through recognition.

# Free and Open Source Software Development Model

The open source model has received increasing attention as an alternative to closed source development. It is characterized by the transparency of development process and artifacts produced, as well as the decentralized organizational structure through which a community of individuals coordinate their activities. The decentralized team structure has all the characteristics of self directed work teams.

## Background: Free and Open Source Software

The term open source as used by the Open Source Initiative (OSI) is defined using the Open Source Definition<sup>viii</sup>, which lists a number of rights which a license has to grant in order to constitute an open source license. These include most notably, free redistribution, inclusion of source code to allow for derived works which can be redistributed under the same license and integrity of author's source code. GNU project, for example, is a copy-lefted software, which is free software whose distribution terms do not let redistributors add any additional restrictions when they redistribute or modify the software. This means that every copy of the software, even if it has been modified, must be free software, a prescription embodied in the most well-known and important license, the GNU General Public License (GPL).

## Introduction

Free and open source software (F/ OSS) is also intrinsically linked with the development of the Internet. The relationship is symbiotic as much of the underlying software that makes up the Internet is F/OSS, and yet F/OSS relies on the Internet for the dissemination of software, and communication between developers and users. This is where these two concepts of virtual community and F/OSS meet. However, F/OSS communities differ from other types of virtual communities because of their emphasis on software. Likewise F/OSS development differs from traditional software development, largely due to the use of the Internet as a development forum. F/OSS community is therefore a unique phenomenon, the details of which can appear undefined and illusive.

The main ideas of this development model are described in the work of Eric Raymond's, *The Cathedral and the Bazaar*<sup>ix</sup>, in which he contrasts the traditional type of software development of a few people planning a cathedral in splendid isolation with the new collaborative bazaar form of open source software development. In this, a large number of developer-turned-users come together without monetary compensation<sup>x</sup> to cooperate under a model of rigorous peer review and take advantage of parallel debugging that leads to innovation and rapid advancement in developing and evolving software products.

The development models that evolved within the F/OSS community proved to be highly effective in managing complex, highly distributed projects and facilitating communication and collaboration among developers in a very diverse geographical and cultural environment<sup>xi</sup>. F/OSS development models were based on the ideas of intensive communication between developers, large dependency on peer reviews, and frequent release of source code. The F/OSS paradigm, based on cooperation and collaboration among developers from all over the world, introduces methodologies and development models different from those usually utilized within the proprietary software industry. A community of developers and users share a common interest in a project and interact regularly with one another to share knowledge collaboratively solve a common problem<sup>xii</sup>. Communities are at the core of what is described in as collaborative innovation networks<sup>xiii</sup> (COINs), highly functional teams characterized by the principles of meritocracy, consistency, and internal transparency. The progress of an open source project is continuously tracked in a number of archives including code repositories, mailing lists, wikis, and bug tracking lists.

### Formation and Growth of Open Source communities

To gain a better understanding of the mechanisms underlying the growth of communities we need to look at how communities form, grow and evolve. Much insight in how developers participate in an open source community can be gained by modeling them as social networks. A community is modeled as a network<sup>xiv</sup>, in which nodes represent developers, and links or edges between nodes indicate that these developers participate in the same project. A characteristic of many of these networks is that they are dominated by a relatively small number of nodes linked to many other nodes. These nodes represent highly prolific developers or “hubs.” Such networks are also known as scale-free networks<sup>xv</sup>. These networks give rise to hubs due to preferential attachment<sup>xvi</sup>. Intuitively, as the network evolves, nodes will be more likely to link to nodes that already have a large number of links, or a high degree in the network. The growth of an open source community is explained in literature by a process of preferential attachment (Madey et al., 2005), or selection through professional attention<sup>xvii</sup> (van Wendel de Joode, 2003). The conclusions drawn about growth of communities are

- Larger communities will attract more new developers.
- Open source communities are dominated by a relatively small number of developers linked to many other developers.

- New developers are more likely to link to well-connected developers. As a result, these well-connected developers become even more connected.
- New communities (which have not yet established links with other communities) are more likely to link to well-connected communities.

Tagging and copying mechanism (van Wendel de Joode, 2003) signal to other developers to join a community. Reputation is an example of a Tag, which signals a certain level of knowledge or skill. Members of an open source community are inclined to copy the behavior of members with a high reputation. Thus, if a developer with a high reputation creates a project, other developers and users will be attracted to participate in this project. The size of a community then itself becomes a tag and signals the popularity of a project, in turn attracting further developers.

### Open source Beliefs, Values and Norms

The Open source culture inherent in the development model is driven by a different set of beliefs and values than traditional development model. The idea of openness and sharing has created a culture that reduces conflict, power asymmetry and encourages participation.

#### **Belief in Free Software :**

The belief in free software appears to be a core motivator of free software developers. This belief is manifested in electronic artifacts such as the Web pages, source code, GPL license, software design diagrams, and accompanying articles on its website and elsewhere.

#### **Belief in Freedom of Choice:**

F/OSS developers are attracted to the occupation of F/OSS development for its freedom of choice in work assignments. Participants to some degree can select the work they prefer. This belief is manifested in the informal methods used to assign or select work in an F/OSS project.

#### **Value in Community:**

The beliefs in free software and freedom of choice foster a value in community. This value is evident, for example, in the IRC archives when newcomers join GNUe offering suggestions or pointing out bugs, and GNUe contributors quickly accept them as part of the community.

### **Value in Cooperative Work:**

Open Source community's beliefs in free software and freedom of choice combined with the value in community foster a value in cooperative work. Contributors work cooperatively to resolve conflicts through the use of IRC and mailing lists.

### **Open Disclosure Norm**

Open disclosure refers to the open content of the open source website including the software source code, documentation, and archived records of IRC, Kernel Cousins, and mailing list interchanges.

Conversations between contributors is recorded on a daily basis and recorded for future reference.

### **Informal Management Norm**

Except for large open source projects such as Linux or Apache foundations most other open source projects have little alliance or sponsoring firms as a networked virtual organization. Most are some forms of emergent organizations where participants work together contribute. Thus the participants self-organize in a manner more like a meritocracy (Apache.org).

### **Immediate Acceptance of Outsider Critiques Norm**

Openness and visibility to all aspects of the code means the community is more open to criticisms of the code or procedures from outsiders.

### **Conflict Resolution**

In most cases, conflicts are resolved without formal management techniques. At the same time, the beliefs in free software are reinforced by people defending their positions and this, in turn, helps to perpetuate the community.

### Motivation to participate

In order to understand the success of Open source development model, it is essential to understand the motivation to participate in an open source project. In general, community members perceive problem solving as their primary reason for participating with F/OSS communities<sup>xviii</sup>. Another way is to view a participant as a prosumer- a user who adapts and refines the software according to his or her needs<sup>xix</sup>. The interaction of prosumers is determined by only low rivalry conditions and, therefore, by low opportunity

costs. Therefore, there must be a selective incentive for contributors—a benefit that only persons who engage can reap. However, in low cost situations only a small selective advantage is needed and the free revealing of the source code is favorable for the user-developer. Some of the key motivating factors are compiled from literature can be split into three categories: technological, economical, and socio-political using the framework<sup>xx</sup> proposed by Feller and Fitzgerald (2000).

- *Use* : To fill a need for an application
- *Reputation and signaling* : A developer's status within the project depends on their performance and, therefore, reputation reflects skills, talent, engagement, and all other important characteristics from an employer 's point of view If a person's reputation is a valid indicator of his or her talent, this reputation can act as a signal in the sense described above
- *Community Identification*: Identification with a group and its goals can explain an individual's actions
- *Learning*: Open source projects have an appeal of programming at the edge of technological innovation; promise to offer extraordinary learning opportunities. In addition, the peer review system specific for the open source area provides timely feedback (e.g., identification of software bugs or suggestions for code improvements) that increases the contributor's learning effect
- *Altruism*: Programmers sometimes engage in an open source project with motivations, for example, because they use open source software and, thus, feel the obligation to reciprocate
- *Fun*: Open source developers program in their spare time because they consume “fun” with this activity and, therefore, open source software is a by-product of this activity

While new participants to the community may sometimes be discouraged from participating due to a feeling of inadequacy, it is the cycle of learning and sharing knowledge that allows them to become experienced. Experience is what counts in these communities and as novices become experts, they can then help others to increase their knowledge and make the same transition. In addition, sharing knowledge and skills was also shown to be the prime expectation of community members. The use of F/OSS communities as knowledge and skill sharing forums has been identified as an important reason for participation<sup>xxi</sup>.

### Communication & Information Flow

In keeping with the Open norms, communication in Open source development model relies on frequent interaction between the community members using mailing lists, forums, wikis and blogs. Automated mailing lists range from development activities of projects and subprojects (Linux has about 81

Majordomo mailing lists for all flavors of open source projects currently undertaken<sup>xxii</sup> ). Using web as the platform for development, by default, allows for easier reach and messaging on part of the development community to attract new participants. This allows for more open and frequent communication between the participants. Based on the research by Lakhani, K.R., & von Hippel, E. (2003), the frequent interaction between the community lowers the opportunity cost and increases the value of individual's participation thereby making development activity less costly than traditional development models. The web provides a common medium to communicate with others in a uniform manner. The community website provides a medium for all sorts of communication, from support activities, documentation to project charter and mission statement. In this manner the website speaks in a consistent manner and reaches out to the outside world with a clear message about the community of developers. More recently the use of wikis and blogs allow for the developer community to reach out to non technical community and raise awareness about the project.

Use of technology tools such as mailing lists, forums, wikis etc allow for information to flow openly and with all participants having equal access to the information. There is no hoarding of information about the development activities or status of projects, which empowers the participants to make decisions about their involvement with the community. As more shared developers join new communities, they increase the interaction between communities and facilitate inter-project communication flow.

### Common Vocabulary

Open source communities consist of technical developers that are well versed in some of the most common programming languages. C, C++, Java, Perl and PHP are widely used programming languages and developers use a set of IDE (Integrated Development Environment) to code in these languages. The significance of the IDE and common programming language is that it provides a common platform and a template for users to understand the coding activity.

### **Coding Style Guides**

Communities like Apache and Linux have *coding style guides* (Egyedi & van Wendel de Joode, 2004; Kroah-Hartman, 2002). The coding style guides prescribe how a piece of source code should be styled. Thus, the coding styles aim to achieve unified definitions and a single style of writing software among all developers in a community. This uniformity or standardization reduces the time needed to understand source code written by other participants and diminishes the need to communicate about a piece of software. It thus increases independency in the communities and allows for a more decentralized development effort.

## **Use of Comments**

Adding comments to the code serves as a means to communicate the intent or functionality for the piece of software code. Some of the IDE also provide boilerplate comments that enhance the code readability and understanding. Adding comments to code works as a signaling function as well, as it enable the author of the code to mark presence of inelegant code.

All these mechanisms provide a way in which participants understand the issues and communicate in a uniform fashion with others. A common vocabulary encourages more involvement from the participant and thereby reduces overhead required in development process.

## Project Control

Large open source projects such as Linux or Apache comprise many subprojects which are associated with an *ecology* of communities<sup>xxiii</sup> However, the communities have a common governance (Apache Software Foundation for the Apache project), and often produce artifacts shared among all projects (such as the Jakarta Commons in Apache).

## **User Participation**

People spend little time to formally inform each other every time they start a new activity or continue on someone else's work. Generally speaking, they ask no permission to start a project or consult others to determine whether they believe a certain project is valuable. Instead, they simply *do*. Anyone with the appropriate skills or the willingness to learn can be a software documentation writer, a developer, a tester, and so forth. Participation requires a person to be technically oriented and a high level of requisite knowledge for access, which renders the low entry boundaries effectively.

## **Contributions' Approval**

Project maintainers within the F/OSS community rely on informal trust mechanisms and on their own network of acquaintances and personal relations with contributors. These practices enable maintainers to place some degree of trust in contributions from people they know. They may also initially start dealing with a contributor's submissions skeptically, until this contributor establishes a reputation based on the quality of his submissions. Afterwards, the project maintainer could increase his or her trust in this



contributor's submissions. Reputation may be transferable to other projects, although some projects might require specialist skills and therefore would render the past reputation of the contributor useless.

### Project Team

The fundamental difference is the role transformation of the people involved in a project. In proprietary software projects, developers and users are clearly defined and strictly separated. In F/OSS projects, there is no clear distinction between developers and users; all users are potential developers. Members of an open source community play different roles, ranging from project leaders (maintainers) and core members (contributors) to active and passive users<sup>xxiv</sup>. Project leaders are often also the initiators of the project. They oversee the direction of the project, and make the major development decisions. Core members (contributors) are members who have made significant contributions to a project over time. Active users comprise occasional developers and users who report bugs, but do not fix them. Passive users are all remaining users who just use the system.

The distinct feature of role transformation in F/OSS projects leads to a different social structure. People involved in a particular F/OSS project create a community around the project, bonded by their shared interest in using and/or developing the system. Members of an F/OSS community assume certain roles by themselves according to their individual ability and personal interest, rather than being assigned roles by someone else. This self selection of task is the distinctive advantage that F/OSS has over traditional development method as claimed by Benkler<sup>xxv</sup> (2002), because it enables the matching of the best available person to a given job.

### **Project Leader**

The project leader is often the person who has initiated the project. The project leader oversees the direction of the whole project and makes most of the decisions about system development. Although all other members in a project are free to contribute and provide feedback, it is up to the project leader to decide which contribution should be included and which feedback should be addressed

### ***Core Member***

Core members are responsible for guiding and coordinating, collectively the development of an F/OSS project. Core members are those people who have been involved with the project for a long time and have made significant contributions to the development of the system. For example, PostgreSQL does not have a single project leader. Instead, it has six core members who collectively decide the direction of the

system, and the inclusion of a new feature must be sponsored by one core member and approved by all other core members.

## **Contributors**

A contributor is simply someone who in some way has contributed to the project. You don't "resign" as contributor, you simply stop contributing. As only committers have access rights to the repositories, a committer must always approve the work of a contributor and perform the actual changes to the repository. A contributor is free to choose the tasks (i.e., bugs) that seem most attractive to work on, and websites and documentation in the two projects actively encourage people to do this.

### ***Active Developer***

Active developers regularly contribute new features and fix bugs; they are one of the major development forces of F/OSS systems and work very closely with the project leader or core members. Therefore, active developers, whose capability is well regarded and trusted by the project leader and core members and whose number is not very large, not only contribute their own code but also play an intermediary role.

### **Peripheral Developer**

Peripheral developers occasionally contribute new functionality or fix bugs. Their contribution is irregular, and the period of involvement is short and sporadic. The vast majority of developers make very small contributions.

## **Committers**

A committer is a developer with the right to add or change code in the repository. In both projects, you have to demonstrate your competence first, typically by adding high-quality contributions for some time, before being given CVS write access.

### ***Bug Reporter***

Bug reporters discover and report bugs. They do not fix the bugs themselves, nor do they necessarily read source code. They assume the same role as testers of the traditional software development model.

### ***Reader***

Readers are active users of the system. They not only use the system, but also try to understand how the system works by reading the source code. At the same time, readers are also acting as peer reviewers or code inspectors who put implicit quality pressure on developers. Readers pay special attention to keeping design simple, writing clear and high-quality code, following strict coding conventions, including

documentation, and adding examples to show how it should be run because they are all aware that programmers worldwide will see their source code

### *Passive User*

Passive users use the system in the same way as the users of proprietary software. They are attracted to F/OSS mainly due to its high quality and its potential to be changed when needed. They are end-users who use computing services whose implementation is based on F/OSS systems. Although they are not members of an F/OSS community as they are not directly involved in using or developing the F/OSS system, they have stakes in the F/OSS system because they depend on it.

### Project Teamwork

Project initialization is done by the project or module owners, projects can also be viewed as modules ( ex Mozilla ), where module owners are responsible for overall functioning of a module. In the project's development phase, the application's core functionality is created. In this phase a core community is built around the need of the project by members who have the same need for a solution. The community members may be utilitarianists as well as elite-contributors. Utilitarianists contribute because they are interested in the result and their engagement helps the project. Elite-contributors join for the fun of contributing or learning. In order for community building to occur, the project owner has to offer a credible project vision and challenging tasks for the developers joining the project. The project establishes, more people join to project to fill a specific user chosen role. These new set of participants are important to the project and build the community's culture and identity. They also do the more tedious work essential to reach project stability, for example, project documentation, usability tests, quality and release management, and so on.

The "break-even" point in the project occurs when the project has gathered enough momentum to attract reputation-motivated contributors. Once the project reaches stabilization project leaders only accept codes that are unobjectionable and of outstanding quality. Thus, such projects indeed provide credible signals for the outsiders and, therefore, are attractive for programmers who play the reputation game.

The ultimate proof that an open source project is both stable and successful is its inclusion into a distribution or distro. A distributor selects an open source application only if it adds value to his distribution on the one hand and if it is easy to install on the other. The first condition implies that the distributor has enough clues that silent users demand this application. The latter condition means that the project concerns not only about coding and architecture, but about documentation and packaging, too. To

avoid stagnation of project, the project continuously needs lead users so that it can evolve even in its stable form. Whereas silent users only work with the stable releases of an application, lead users download and install release candidates. Thus, they act as beta testers and provide helpful feedback to the project if they find bugs or deficiencies. Lead users are elite-cooperators. They have fun using the newest version of a slick tool long before others; at the same time they learn and build up valuable knowledge about the application, its evolution, and hidden goodies and limitations.

In the later phase of the development stage, contributors enter the project who might be motivated rather by fairness norms than by fun. The lead users' feedback drives the project to a considerable amount, whereas a project without lead users will stagnate and decline within a short time. Lead users on the other side are attracted by new features. Therefore, reputation-motivated contributors and lead users have a reciprocal relationship: Reputation-motivated contributors implement the new features of an application, which the lead users demand, whereas the latter provide the feedback and stimulate activity. Therefore, lead users need a low-cost access to the source code or the application's installers as well as a credible signal that the open source project in its actual form will persist. To conclude, as long as an open source project succeeds in accomplishing heterogeneous needs, it can attract the differently motivated contributors building a vibrant community required to make the project successful.

### Coordination

Open source communities bring together a dispersed collection of people, sometimes a large number of them, around the development of open source software. In the absence of enforceable formal structures, like those found in corporate settings, understanding coordination between participants is essential for success of the project. Most decisions are made on an individual basis by the participants and as software becomes complex the interdependencies increases thereby increasing the need for collaboration and coordination. However there are mechanisms in place that make coordination easier- notably modularity and elegance.

### **Elegant Software code**

Source code is either elegant or it is not. The more experienced and skilled software programmers are claimed to be best judges of whether source code is elegant. Thus, although the number of lines of source code is bound to increase when the functionality of software is enriched, an elegantly written piece of software provides some counterforce to complexity and to a certain degree ensures that the code remains relatively easy to understand and to change. This enables participants to make decisions without paying

more attention to reading and understanding source code than is strictly needed. It also allows people who were not previously involved in the community to improve code without spending much time and effort deciphering the source code.

Elegance also relieves the need for coordination and collaboration. Because the code is elegant, it is easy to understand what the effects of a change in one part of the software will be for other parts. Elegance allows developers to either adjust other parts of the software or to ask others to take a look at it.

### **Modular code**

Modular software is divided into smaller pieces, building blocks, which together form a software program. By clearly defining the module and its interface that connect to the module, the need for coordination is reduced in development of project. Modularity reduces interdependency between developers and allows them to work independently, thus reducing the costs of coordination.

### **Names Attached to Improvements**

In every open source community, one or more mechanisms are adopted to relate participants to their contributions. One such mechanism is the credits list, which contains the names of developers who have contributed to the development of software. (Ex. Apache community). Connecting participants to their contributions also fulfills a coordinative function. The contributor of the source code is known and thus feels responsible for changes. He is also the most appropriate person to fix a problem with the code, if a user of the source code discovers a bug.

### **Small and Incremental Patches**

Another coordinative mechanism is the norm that developers should only contribute source code in small and incremental patches. Keeping the patches small is important, as it makes it easier for others to understand what the source code aims to achieve and how it intends to do so.

### Mechanisms to Coordinate Massive Amount of Individual Efforts

F/OSS communities use a large and rather sophisticated infrastructure that supports their activities and, in the process, coordinates their efforts with those being invested by hundreds if not thousands of other developers. As communities start to grow and attract more and more participants, they tend to support their activities with an increasingly sophisticated technical infrastructure. This infrastructure consists of mechanisms that computerize coordination. Next to the technical infrastructure, participants have adopted a number of devices that are related to a specific way of working. Many communities have further adopted methodologies and standards to coordinate their individual efforts.

## Versioning System

Most open source communities support their development and maintenance activities with a software versioning system. Well-known examples of such systems are the concurrent versioning system (CVS), subversion (SVN), and bit keeper. These systems are automated systems that allow remote access to the source code and they enable multiple developers to work on the same version of the source code simultaneously. Older versions of the source code are automatically stored in the system and can be used as a backup. In these restricted communities, only participants with committer status can upload source code. Basically, versioning systems support the decentralized development process<sup>xxvi</sup> in a number of ways. First, participants can access the versioning system simultaneously. They do not have to wait until another developer has finished working on the source code. Second, the presence of logs is important. The log files provide participants with an explanation of how the source code works and what it intends to accomplish. Third, the versioning systems allow participants to move back in the development line and take an older version of the source code. This enables them to take out a commit that at a later stage of development proves to be bad code. This last option effectively reduces the need for participants in the community to monitor and analyze the value of every new commit. Versioning systems are systems that support the development activities of individuals and allow multiple developers to simultaneously improve a certain piece of source code. Adopting a versioning system thus reduces the need for coordination among the participants in a community.

**Automated mailing lists:** One of the basic tools to support coordination in open source communities is mailing lists. Every community has a number of mailing lists on which different issues are discussed. These lists serve different purposes and target different audiences. Certain lists focus on users, providing them with a forum to ask questions and receive answers. In short, the mailing lists provide the developers with a forum to exchange and discuss their ideas, and they also give users a forum to ask questions and receive answers.

## Bug-Tracking Systems

Bugs are basically mistakes or flaws in a software program. Many software bugs are discovered while the software is actually in use. Users of open source software often write a “bug report” when they come across a mistake or when they find that something does not work<sup>xxvii</sup>. More advanced bug-tracking systems, for instance, have a format in which a bug report should be written. A report of the bug is then stored in the system, where it awaits someone to fix the bug. Effectively, the more advanced systems

eliminate the need for people in the communities, first, to contact others and explain about the bug they found and, second, to convince another developer to solve the bug.

### **To-Do Lists**

Participants typically have many ideas about how a software program should work or what new features should be added to a program. The only way these ideas are transformed into actual lines of source code is by someone writing the source code.

The to-do list is a coordinative mechanism because it signals developers as to what others in the community find important. Participants do not have to discuss and explain why they find the ideas important. Instead, they just put the item on the to-do list. Others can take a look at the list and judge for themselves what they find interesting and what they would like to work on. As such, the to-do list serves as a marketplace in which demand, for example, for a certain feature, meets supply, namely participants who have the knowledge, time, and motivation to develop the feature. To-do lists are another example of a mechanism created and adopted with the goal of structuring the efforts of individuals in the communities.

### **Tinderboxes**

In order to be able to discover errors in newly committed code as fast as possible, some projects use Tinderboxes to test build on different environments and hardware. Results from the process are communicated to tinderbox websites automatically. This reduces the communication required to test a system for different configurations.

### **Verification Machines**

Supplementing the automatic and continuous tests performed by the tinderboxes, Mozilla follows a rather strict procedure in which the source code in the trunk is tested on a daily basis. The tests follow this scheme, according to Yeh (1999) and our own observations:

### **Website**

A projects' websites has several important functions. It presents both the projects and the products (the software) to the outside world, and it acts as a portal, making it possible for developers (and everyone else) to locate and down- load all existing information (project documentation, manuals, and news) related to the project.

## Project Leadership

Generally, leadership is “given” to the person who makes the first lines of source code publicly available. Other communities, like Apache, have leadership vested in a board of directors. The community members periodically elect a new board of directors. Although the importance of project leaders is identified by many, due to their limited influence over other participants, there is no universally accepted view of their leadership ability. von Hippel and von Krogh (2003), argue that project leaders are different from most managers in “traditional” companies, because they cannot enforce.

Project leadership is an important part in ensuring coordination among the contributors, as the project lead performs many activities that benefit the joint development of the software. However, project owner’s actual influence over individual contributors is rather limited. In some F/OSS communities there are no project sponsors or leaders but the community itself performs the leadership role collectively. This characteristic—the fact that self-interested and individualistic behavior aggregates into collective processes that benefit all.

Some projects employ several staff and management functions, i.e., people with more permanent positions, including formal authority and obligations. A study<sup>xxviii</sup> conducted on the FreeBSD<sup>xxix</sup> shows that there were 18 “official hats” covering everything from Public Relations to Standards.

### ***Top-Level Management***

In the case of Mozilla and FreeBSD the foundations are headed by groups comparable to the board of directors found in traditional organizations. In Mozilla, this group is the Mozilla.org staff of 11 persons (Mozilla Roles and Responsibilities, 2002)<sup>xxx</sup>:

*“ The Mozilla community is governed by a virtual management team made up of unpaid experts and employees from a range of companies. Leadership roles are granted based on how active an individual is within the community as well as the quality and nature of his or her contributions. This meritocracy is a resilient and effective way to guide our global community. The different community leadership roles include:*

#### *Module Owners and Peers*

*A module owner is someone who is responsible for leading the development of a module of code or a community activity. This role requires a range of tasks, including approving patches to be checked into the module or resolving conflicts among community members.*

#### *Release Drivers*



*Drivers provide project management for milestone releases. The drivers provide guidance to developers as to which bug fixes are important for a given release and also make a range of tree management decisions.*

### *Super-Reviewers*

*Super-reviewers are a designated group of strong hackers who review code for its effects on the overall state of the tree and adherence to Mozilla coding guidelines. Super-review generally follows code review by the module owner, and the approval of a super-reviewer is generally required to check in code.*

### *Bugzilla Component Owners*

*A Bugzilla component owner is the default recipient of bugs filed against that component. Component owners are expected to review bug reports regularly, reassign bugs to correct owners, ensure test cases exist, track the progress toward resolving important fixes, and otherwise manage the bugs in the component. The Bugzilla component owner and the related module owner may be the same person, but in many cases they will be different.”*

## Balancing Anarchy with Control

In keeping with the principle of openness, very little information seems to be kept secret with all participants having equal access to project.

- Everyone can download every version of every file
- Everyone can monitor each file for who made which changes
- Test results are free for everyone to see
- Nearly all newsgroups and mailing lists can be read by everyone.

To address the risk related to the degrees of openness there are mechanisms in place that allow for quality control and reduce risk, such as

- Every change to the repository is logged and reversible; and
- A considerable effort is put into detecting and correcting broken-build situations as quickly as possible.

## Quality

In OSS communities, the source code is treated as open that can be downloaded and modified by anyone. This leads to a highly decentralized software development in which large numbers of people contribute time and effort. Despite the highly decentralized and geographically-dispersed development process, the software that is developed in some of the communities is of a high quality. Based on the research by Lakhani and von Hippel (2003), the Apache and Linux software are said to be of a high quality. The authors argue that the cost of participation in communities is relatively low and that the low level of benefit may be enough for participants to contribute to the project.

In addition, automated test tools like tinderbox and verification machines are used in projects and the results of the test are kept open and transparent as possible. This enables any user to validate the claims of the maintainers and developers, by conducting these tests using the same mechanisms that were used in producing them. Furthermore, users may customize the testing scenarios to match their own specific requirements.

## Comparing Traditional Project Development with Open Source Project Development

Projects developed using traditional or closed source approaches have well defined software processes. Most follow some sort of development cycle such as waterfall model, prototyping model, spiral model and more recently Rational Unified Process. These traditional models for managing the projects differ with the development models in Open source

1. Unlike closed source projects, in F/OSS projects the number of contributors (programmers) greatly varies in time, cannot be directly controlled, and cannot be predetermined by project coordinators.
2. In any F/OSS project, any particular task at any particular instant can be performed either by a new contributor or an old one who decides to contribute again. In addition, it has been shown that almost all F/OSS projects have a dedicated team of programmers (core programmers) that perform most of the contributions, especially in specific tasks (e.g., code writing), while their interest in the project (judged by how often they contribute in the course of time) stays approximately the same<sup>xxxix</sup>.

3. In F/OSS projects, there is also no direct central control over the number of contributions per task type or per project module. Anyone may choose any task (e.g., code writing, defect correction, code testing/defect reporting, functional improving, etc.) and any project module to work on.
4. The allocation of contributions per task type and per project module depends on the following sets of factors:
  - a. Factors pertaining to programmer profile or preference- some programmers may prefer code testing to defect correcting. This also depends on the aptitude or preference of the programmer to write code.
  - b. Project-specific factors - a contributor may wish to write code for a specific module
5. In F/OSS projects, because there is no strict plan or task assignment mechanism, the total number of Lines of Code (LOC) written by each contributor varies significantly per contributor and per time period, again in an uncontrolled manner. Therefore, project outputs such as LOC added, number of defects, or number of reported defects is expected to have a much larger statistical variance than in closed source projects<sup>xxxii</sup>. This fact is not only due to the lack of strict planning, but also to the much larger numbers and diverse profiles of contributors that participate in an F/OSS project.
6. In most F/OSS projects there is no specific time plan or deadlines for project deliverables. Therefore, the number of calendar days for the completion of a task varies greatly.
7. Submissions for changes or new code are sent by technically competent participants that

## Application of an Project Management Toolkit in Open Source

Drawing on the information from Part I and using Fayol's principles as a guide, project management in open source takes on a new meaning to managing project. Internet serves as a visible and accessible platform for project functions during the software development. With no individual i.e. project manager responsible for the entire project, as in the case of traditional development model, but a number of people contributing iteratively to the code, there is no central authority or role player in this model. In case of Mozilla or Apache an organization governing body exists only as an overseer of the project that guides the overall vision of the project. Developers contributing to the project also play different roles in the development of the project and each member has access to the project information equally.

Project reviewer does not have the same level of responsibility as project manager in a traditional development process. The project reviewer in open source model is responsible only for starting a project but that does not guarantee that the work gets done. In addition, he/she has little influence over who contributes to the project. Developers contribute to the project for reasons that are important to them like solving a problem or have a need for the solution. A number of automated tools are used in communicating, gathering requirements or statusing the project. The reliance on automated tool reduces the need of communication and the open access of information reduces the amount of coordination required, as compared to traditional project.. The toolkit presented here addresses the key elements that ensure a smoother project management process in the open source community. In almost all the cases of open source development the skill set of developers is at par or higher for what is needed to be accomplished in the project.

*Requirements* are descriptions of the “desired features of the proposed system.” They may be functional (describing what the system should do) and non-functional (describing how a facility should be provided, or how well, or to what level of quality). F/OSS projects usually lack an explicitly formulated requirements specification (Bartkowiak, 2004), because most of the projects depend upon the evolutionary development of requirements. Determining project requirements does not conflict with this concept, and the recommendation simply suggests specifying the translated expectations in clearly defined requirements that the project should meet. The initial formulation may capture only the essential required functionality, and could develop through the same current evolutionary mechanisms. In order to facilitate requirements approval, the maintainer may define an approval threshold for each document. When the number of users who approved the requirements reaches the defined approval threshold, the document status automatically changes into “Approved.” This method of gathering requirements can be thought of as a market based requirement gathering process. Since requirements that are approved by

members meet a threshold, it also means that the features included in the software are more aligned with the needs of the market collectively rather than driven by individuals. Thus, comparing the requirement gathering process in open source with traditional process, it is far more attuned to removing unwanted functionality from the software since the requirements gets reviewed by members representing different needs. If the need for a feature exists in the market it will be reflected in the number of approvals received for the enhancement. In traditional development process, the needs of one person or group can overshadow the actual need for an enhancement, because the person may have more control or authority over the project.

*Project Teamwork* includes how teams resolve accountabilities, responsibilities, and promotes teamwork and communications. In open source development the person who starts the project is ultimately responsible for the success of the project. He/she however cannot assign developers to the project, developers who contribute to the project do so only to “scratch an itch” or due to the altruistic reasons. A number of signaling mechanisms are used to attract developers to contribute to the project, that include reputation of the project leader, number of users already attached to a project, number of contributions added to the project or recognition of code contributed to the project. Each of these factors contributes to the success of the project. Each developer contributes a piece of the code that eventually gets committed in a distro after it has been thoroughly tested by other community members. Communication between the members is done using mailing lists, wikis and blogs that give users and developers enough information about the project status. The members of the project team are in continuous state of flux, with new members joining the project and making contributions on a small scale.

The *Project Team* in an open source development project is distributed across countries, regions and is more flat in terms of organization structure. The project team itself is fluid over the course of the development cycle, with members joining at different stages of the project. Additionally anyone can become part of the team by contributing and subscribing to mailing lists. The project team consists of members that are technically skilled at doing specific tasks. Team members perform different functions required in release of code like contributor, developer, bug reporter, reviewer and tester. Sometime multiple functions are performed by the same individual but in different capacity. In this setup no one individual plays the role of project manager instead either a group of core contributors or the project initiator influences the decision making. Influence is earned in the project by reputation of a developer, based on his/her contribution in the past. Communication tools like wikis, blogs and mailing lists play an important role between the members in the team. By allowing for more frequent interactions with team

members using these simple tools the cost associated with coordination reduces further and in turn provides significant reward for team members to contribute more.

*Project Planning* The planning of the project revolves around the release management for software build. Keeping with the motto of open source of ‘release often, release early’ the software goes through multiple phases Alpha , Beta and production. Planning associated with release of the software requires planning in distinct phases. Though no one person is responsible for the entire life cycle, the software is released after review from users and core developers. Release planning can be time based driven or feature based driven. A number of projects including Mozilla<sup>xxxiii</sup> , Ubuntu<sup>xxxiv</sup> and OpenOffice<sup>xxxv</sup> follow the time based release model. In the case of Mozilla the typical release consists of six-week period, during which the changes to the source code is allowed leading to an Alpha version of the coming release. A four week period of code stabilization leads to Beta release. After that a three week period in which only “stop-ship” bugs are found and fixed. From the start of this period, all changes must be accepted by drivers. After two weeks, a release branch is made, separating final work on the coming release from work on the trunk, and effectively marking the start of the next release project. If drivers don’t think the release is of satisfactory quality, this branch may be postponed.

It is possible to make changes to a release even after the release date, but this seldom happens. When it does happen, it is most often when a bug-fix from the trunk leads to a similar bug-fix in the release branch (called merging). These changes must still be accepted by drivers.

The project roadmap defines the list of requirements that are included in the release and the set of activities that are needed to be performed for a release. The lists of requirements at the highest level are defined in the project roadmap and the lower level project work was started by individuals looking to contribute. Anyone can add a requirement to the project which is voted on by the members of the community. Lower level planning around tasks such as resource allocation is not required in open source model since no one person has the authority to allocate a task. The task only gets done if there are members willing to contribute. An updated list of project level activity is maintained by members of the project detailing what is included in the release branch.

*Opportunity and Risk* The most important artifact produced during the open source development is the code that is combined into a build for a release. Since any member can participate and contribute to the code, there is a risk in releasing buggy software. The project team minimizes this risk by using code

repository or versioning system and by thorough testing of the code by other members. The versioning system allows for a change to be reverted or even removed from the code, thereby making sure that the code that gets in has all the approval from the reviewers and tested thoroughly by testers. In addition, by releasing patches or bug fixes often, the project ensures that the gap in functionality is covered. To identify possible bugs or even opportunity the project relies on the participation from passive members or users, once a bug is reported it is prioritized by project members.

*Project Control* Project control in open source is limited to guidelines for its members. No one person has the authority or responsibility to control and report status of the project. Members of the community are required to follow a set of guidelines but there is no control over how and when tasks associated with projects get done. Log files are used in development and committing changes that clearly define the change and the contributor. This information is available for all users to see which allows for little misuse of the process of committing changes to a release

*Project Communication* include mailing lists, wikis and blogs associated with the project. The project website is used heavily to communicate to users and project members on the status of the project. All documentation created by the participants , rules and guideline , project roadmap, mission statement etc are all shared openly and can be viewed by all through the project website. The use of mailing lists allows for open communication between members in a cost effective manner. The website is also used as a marketing tool by the project to enlist new members and to gather support for the project in the community. Blogs and wikis allow for communication between the core developers with the other members in the project. Projects also have automated mailing lists that are open for all to join and allow for sharing of information amongst all members. Real time communication is done using Internet Relay Chat IRC or through instant messaging that allows for frequent interactions that builds trust amongst members and enables members to work independently.

*Project status* In open source development project website and color coded status of requirements and bugs are used to communicate the status of the project. The overall release status of the project is communicated by the project drivers and enhancement level status can be found in release document for the current release. This includes the feature list, improvements and the criticality of an issue. Forums are used in the case of Ubuntu development for details related to the status of individual items related to the

release. Measurements include the number of bugs identified, bugs fixed, project activity, number of users participating, features requested etc.

*Corrective Action* Versioning systems such as CVS allow for corrective action to be taken on code that has been committed. Since all changes are logged and reversible, versioning system allows for the code to be corrected if a bug is found in a release. Though all members can contribute to the code, not all members have the access to commit code in the repository. Members gain the privilege of a committer by contributing quality submission to the project. By contributing bug free code to the project a member gains the trust of the project owner and then given the privilege to commit to the repository. This process insures that the right code is added to the repository and also allow for monitoring of the code checked-in to the repository.

*Project Leadership* is often driven collectively by a group of core members and there is no single project leader. Mozilla and Linux both have a governing council whose members are either selected by the group members or have taken on the responsibility to lead the project. In most cases the project leadership committee is not always fixed, membership to the committee changes frequently as new members win support by majority by actively devoting their time and contribute to the project.

By applying the Project toolkit to open source development model it becomes clear that the role of a project manager is greatly diminished. A project initiator or an owner can play the role of a project manager in a traditional development model but the does not have the same amount of responsibility. Internet serves as critical project environment that allows for some of the responsibilities associated with a project manager to be performed by automated tools. The overarching concept of sharing information leads to

- Decentralized development
- Integration with existing working methods
- Minimize administrative overheads
- “Light-touch” management



## PART III

# Open Source Project Management at Enterprise Level

### Introduction and Background

Many companies face a problem in determining how to best adopt and deploy open source capabilities for product development and e-business services. In the past few years, open source software has become a viable solution for organizations, and is being increasingly adopted. This increased popularity has been of Linux operating system and Apache web server has open source vendors (e.g., RedHat and SUSE) and traditional software vendors (e.g., IBM and HP) to provide reliable support for open source solutions. The increased popularity of these open source software can be attributed to impact the software has on the business in terms of cost and its disruption to business users. Open source software has gained prominence in the server-side application and projects in the horizontal domains such as Internet applications, Internet applications developer tools and technical tools<sup>xxxvi</sup>. These server-side applications are best suited for technical community that has accepted the open source initiative with open arms as it provides a low cost alternative to proprietary software. The reduced licensing fee also helps the IT departments in organizations to reduce overall software expense in an organization. The maturity of these open source software and the robustness of software to run enterprise application has enabled open source software to make inroads into other enterprise software applications.

Since most of the cost associated with software development goes towards software development, companies have started to look at open source development model as a sustainable model in the long run. The benefit of collaborative development approach can greatly reduce overhead in large projects and sharing source code through open source provides companies access to knowledge from the development community. Another approach is to allow for collaboration within the company and keep the source code internal. Both options reduce the size and scope of the potential community but can still provide additional value.

A source license can be used to create a gated community: Anyone agreeing to it is in the community, and anyone who does not is left outside the gate, unable to see and use the source code. This can be attractive to the company that wrote the source code because it can still sell the software and retain all of its IP. Whether this is attractive to outside developers depends on the other license terms.

The most restrictive source licenses only allow a licensee to look at the source code--a licensee does not allow modifying or redistributing code. Even this little access can be useful if you are building other software that must interoperate with the licensed software. The source code essentially provides additional documentation and can aid debugging.

It can be to a company's benefit to open up the source code to everyone within the company. Access to a product's source code provides documentation to those developing other products that must interoperate with it. It can help in testing and fixing bugs. It can facilitate code reuse. In short, all of the same benefits that outside companies can get by being able to see the source code are available with internal open source. Even when sharing totally within a company, proposed changes must still be approved by the project's core team or other people who have earned their trust. Sharing source code within a company is much simpler than sharing it with those outside. One of the biggest benefits might be increased communication between different parts of the company.

A deliberate attempt is needed to build an environment similar to open source community internally for developers and users to collaborate. The attempt to participate in open source projects also conflicts with a traditional company's existing work culture and structure. This focus on creating communities is a major difference between enterprise open source and F/OSS. But if the attempt is successful one of the key benefits is software reuse which creates a library of reusable software components maintained by a code librarian. Once created for one project it can be reused in other unrelated projects and that results in less costly development and reduced time to market for software products.

To understand if projects can be managed in companies without project managers, we need to look at the open source development model in established traditional companies. To answer this need to investigate under which circumstances and in which situations are open source development models a viable option? What chances of survival of projects within companies pursuing a specific open source development model? What role does technology play in enabling such project? How to create communities and build products using open source model and so forth?

In order to understand some of these questions I looked at the open source initiative at IBM which has a significant presence in open source projects. The two case studies look at projects on the server side and on the desktop. Both these case studies address the core of the thesis around software development using self directed teams and explain some of the key questions about the role of technology, project management toolkit and the role of project manager in development of software applications. We first look at the common factors that are embedded in both these case studies.

### Analyzing Motivations on the Firm's Level

There are two main reasons for an employer to participate in open source project: The first reason is that the firm needs a certain software solution for its own use. By opening the source code application or joining an open source project, the firm can lower costs and spread risks. The second reason is that the firm has a business model that builds on open source software. IBM made a strategic choice to adopt open source development model internally after learning from Apache and Eclipse projects. Both these projects gave IBM a say in building standards for Internet e-business and development tool for developing other software products.

### **Use Value**

Software developed in-house that has use value for the company and that does not represent any core competency of the company, should not cause any losses if the source code is opened. Besides there is possibility to lower costs, there is also the prospect of risk spreading by open sourcing code. By releasing the application, the firm could spread the application's maintenance over various independent contributors, thus minimizing the risk that the application goes out of date

A company will not succeed in building up a community around the application until the project reaches stability. By committing to a project a company signals to the open source community that it has interests in keeping the project viable. This support, even though with restrictive licensing, can attract reputation seekers and lead users to participate. By implementing a dual licensing scheme for the project a company can achieve its objective and still participate in open source. The commercial license makes the application fit for commercial use whereas the open source license with a copy left clause guarantees the continuous openness of the source code, thus making the contributions visible. The same consideration holds for lead users.

Another strategy for companies planning to build up an open source community in the project's stable stage could be to hand over the code ownership to a foundation. Even this ensures the source code's continuous openness, thus making the project attractive for rent-seekers and lead users.

## IBM's Motivation to Participate

IBM adopted the first approach with Apache web application server. After supporting the Apache foundation, it attracted contributors who were looking to enhance functionality of the web server and used key modules from the Apache web application server in its commercially available Websphere Application server. The alliance benefited IBM and the Open source project mutually; Apache gained over other web application servers by IBM's backing and IBM gained access to community of developers at a relatively low cost. Eclipse is an example of the second approach, where IBM handed over control of the development effort to Eclipse foundation. IBM gained by standardizing its Eclipse based development tool and use it as the standard in developing other commercial products.

## Business Models

The benefit of low cost and reduced overhead presents a viable alternative to proprietary software development. Companies can succeed with the Open Source development approach by adopting some of these prevent business models. These models are based on literature research made by Raymond (1999); Hecker (1999)<sup>xxxvii</sup>; Leiteritz (2004)<sup>xxxviii</sup>; O'Mahony et al. (2005)<sup>xxxix</sup>; Weber (2004):

- **Open source application provider:** Application provider create software that they distribute under the terms of an open source license. A company generates profit by giving away the software for free and enlarges its application's user base thus increasing the market for the complementary product.
- **Loss leader:** In the "Loss Leader" model, an application is given away as open source software to improve the company's position in the software market. According to Hecker, the open source product could increase the sales of the complementary software product "by helping build the overall vendor brand and reputation, by making the traditional products more functional and useful (in essence adding value to them), by increasing the overall base of developers and users familiar with and loyal to the vendor's total product line" (Hecker, 1999).Ex. Netscape's open source strategy with the Netscape/Mozilla Web browser is an example of this business model.
- **Sell it, free it:** The application is sold (i.e., distributed with a commercial license like any commercial product) when it is ready for release. Later in the application's life cycle when a new version is available the application source code of the older version is opened. Customers buying the software pay a premium for the value of using the application earlier rather than later.

- **Dual licensing:** Application is available under commercial and open source license and the two versions of the software address different target groups. The free version is intended for users that get familiar with the software by installing and using it and thus preparing the market for it. The code base of the two versions is the same but the version with the commercial license delivers additional support or product guarantees. Ex. MySQL.
- **Widget frosting** The complementary product is hardware. In a way, Linux represents the open source software to sell Linux computers, that is, computers preconfigured with Linux, specially designed for an optimal support of this operating system.
- **Service enabler:** The complementary product is neither software nor hardware but a service that generates the company's revenue stream. The company sells a service online and needs software so that users can access the service. If the community enhances the client software and makes it more user-friendly or ports it to new platforms, the market of this service will be expanded.
- **Standard creation:** If a company wants to create a technical standard it can safely use as a foundation to build its proprietary applications, open source can play a crucial role. The company that sponsored the code that builds the new standard can level the playing field for potential competitors by open sourcing this code. Moreover, by making the code open source, the initiator signals that contributors can participate in the negotiation about the future evolution of the standard, thus providing incentives for other companies to join it. This strategy seems only possible for big companies having a long-term policy and the perseverance to pursue it. An example of this model from practice is IBM's sponsoring of the Eclipse open source project.

In other business models the company does not create software but profits as a free rider from open source software and the open source movement. However, by selling their services, they popularize open source software in many ways.

- **Support sellers:** Sell support for users of open source software. There are two versions known for this business model: Distributors combine different open source applications to assorted and tested distributions (i.e., media and hard copy documentation) that can be sold. Ex Red Hat. Another variant of this business model, companies sell technical support for users of open source software. This covers teaching, counseling, system integration, and system tuning, and so forth.
- **Mediators:** The strategy of an open source mediator is to operate a hardware and software (e.g., collaborative tools) platform where open source projects can be hosted. Gains can be obtained by

selling advertising space (banners). The more successful a platform is in terms of the amount of participants, the more attractive it is for new participants. Having selected a certain platform, the developer's willingness to change to another platform is very small, especially if the other mediator hosts lesser projects and is frequented by lesser users. Ex. SourceForge.

- **Accessorizing:** Companies pursuing this business model sell accessories associated with and supportive of open source software. Ex. O'Reilly

### IBM's Business Model

IBM's strategy in open source is a multifaceted approach that encompasses focuses on driving adoption of open standards, extend open source products and create new market opportunities using open standards. These broad strategic goals are driven by a desire for IBM to reduce cost in software development, encourage adoption of Linux and create platform to attract developer community to contribute to projects. Additionally, by using open source products in its software portfolio IBM plans to standardize software development across its multiple product line and reduce overhead in developing its portfolio of software product. By adopting open standards IBM hopes to attract larger participation from open source community and drive adoption of its complementary products. IBM's go to market strategy using open source software revolves around the *Open source application provider*, *Dual licensing* and *service enable* business models.

IBM's participation in the Linux project is an example of Open source application provider strategy. Since Linux offers the same robustness and reliability as a proprietary operating system, a large number of IT departments in established companies view it as means to reduce overall IT cost. Some of these large companies are also IBM clients, so it became necessary for IBM to follow its client and address their need to implement Linux for them. By joining the Linux development effort, IBM could serve its clients and address their need better. It also opened up a new avenue for IBM to distinguish itself from other IT vendors, since there were few players in the market who could service this need of customers.

Eclipse is an example of service enable business model that IBM has adopted. By aligning with Eclipse organization and opening its source code to developers, IBM gained by creating a standard Integrated Development Environment (IDE) for its use in software development of its proprietary Lotus and Rational products. Modules of Eclipse software are now used across its entire software product portfolio which has led to reducing in cost developing, supporting and maintaining applications. Through code reuse and modularization, it is much easier for IBM to integrate its different products better than its

competitors. In this manner IBM is not only able to reduce its cost but also offer an integrated solution to its customers who are proactively looking for business solutions rather than piecemeal offerings from vendors.

Apache is an example of Dual licensing business model. Apache is released under open source license whereas websphere application server is released under commercial license. The dual licensing appeal to the customers since it gives them a choice between getting software for free and getting a product with commercial support and maintenance. With free software companies still have the overhead of applying patches to the webserver.

IBM has identified a set of open source projects and standards that it wishes to participate in. These projects have been chosen specifically to advance its growth through implementing the business models mentioned above. Linux, Apache and Eclipse are some of the key external initiatives that it has committed itself to apart from these IBM is involved in only those open source that either fill in a gap in its product portfolio or increases its product adoption ( ex. Sametime, Domino).

Having a business strategy specifically designed to address the open source software sends a clear signal to competitors and developer community that IBM is serious in its effort to adopt open standards and practices. This helps to build consensus internally and externally about IBM focus in open source software development. This is critical for managers and executives that are looking at open source development internally as it removes ambiguity around adoption of open source model.

### Leadership

From a project management standpoint, project leadership is required to guide and sustain the effort through the development life cycle. Commitment from senior executives ensures that open source project will not be abandoned midway. In addition, leadership plays a critical role in getting the necessary buy-in from different stakeholders. IBM's commitment to open source software is evident by creating an open source software community whose members are responsible for driving the initiative internally and also partner with developers externally. The role of the leadership committee is to provide

*Clear goals:* a set of guiding principles that govern the entire development process. These goals clearly explain the use, participation and conduct of IBMers in open source development projects. Unambiguous goals also help in creating policy that covers licensing, employee participation, collaboration with third party and provides an environment of trust within the community of developers.

*Strict guidelines:* The open source software committee has created strict guidelines around sharing source, licensing of source code, approval process for participation and use of open source software. By requiring its employees to adhere to these guidelines, IBM reduces its risk of copyright violation or unauthorized participation of employees.

*Processes:* The leadership committee has created approval process required by employees and reviewer of codes to reduce the risk of exposure to copyright violations. Each project is reviewed by three levels of review committees before any open source code is shared externally or internally. The open source committee has provided automated tools to ensure the controls added to the process are followed and that exceptions are raised and reviewed. Automated workflow tools allow the committee to be proactive rather than reactive to situations of conflict in the project environment. Additionally projects submitted are reviewed for legal implication related to licensing, scanned to remove unauthorized code and patent any new innovative solution. Project owner also has access to a list of experts and authorities including legal to resolve any discrepancy that may arise during or before the project. These processes are designed specifically to reduce any risk of violation and at the same time to seek opportunities to improve IBM's patent portfolio.

### Licensing

The different licensing schemes under which a product is created and released are clearly communicated to participants of the internal open source community. The licensing schemes are color coded to highlight the degree of control over source code. This does not prohibit users from participating in communities but in a way it signals to management the exposure associated with a project. Some of the most critical projects are controlled with restricted licensing to protect IP but at the same time controls are put in place to encourage participation from contributors.

Participation: IBM employees can participate in open source project internally or externally, though each requires a different process to participate. Within each IBM division there are groups that are implementing IBM open source strategy and developers from those teams participate regularly with external developers. Developers working exclusively on Linux and Eclipse projects are required to participate with external developers and their business commitment for the year are aligned with the division's objective to pursue open source code development. Training is provided to these developers to ensure compliance with IBM's open source effort. Developers in this category frequently communicate with outside developers as in the case of any open source project. A developer is assigned to an open source project that meets a goal of the division and the developer is limited to participating only to that project. A secondary e-mail address is provided to the developer to communicate with external developers



and the e-mail is granted access to mailing lists for the project. For all internal communication developer has to use his primary internal e-mail id. These dual ids allow developers to collaborate with external developers and reduce the risk of unauthorized leaks or access for IBM. Each developer's participation internally or externally is approved by his/her manager for each project. The developer can still participate externally in any other project but can do so using his/her personal id. Developers can initiate new projects internally after it has been reviewed by local legal counsel and systems manager.

Employees, not assigned to open source projects mentioned above, can participate in internal projects on their personal time after an approval from their managers. The projects are hosted internally and can have licensing ranging from restrictive (IBM internal) to GNU (open source). This set of developers can collaborate with others only internally through the IBM's Internal Open Source Bazaar (IIO SB). This ensures that the exiting management is aware of employee participation and does not take away time from employees main responsibility.

### Project Visibility

Information about internal projects is available for all employees and does not require any approval. Any employee in IBM can access documentation to learn more about opportunities to participate or read and download code to learn more. The documentation available ensures sharing of knowledge in parts of the organization and exposes employees to open source development process. This approach helps IBM to transfer knowledge between different divisions and geographies and improves the skills of developers across the organization. Documentation also includes information on acceptable norms and behavior in open source for new comers to ease into the open source community.

### Building Community

Communities play the critical role of driving a project in open source, but building communities of developers internally is very different from building communities on internet. To aid in the process of building communities and connect geographically distributed employees, IBM has created a task brokering and volunteering system. The goal of the system is to allow employees to collaborate with others internally similar to an open source community. The system enables users to post a task, volunteer for a task, create a team and volunteer for a team. The main idea behind it is to build a social collaborative space in the company to

- Aggregate volunteer opportunities
- Broker Tasks

Employees can volunteer to participate in tasks that interest him/her or work in teams to solve a problem posed by others. Both these mechanisms address the key motivations for developers to participate in open

source communities i.e. contribute to community and solve problem. The system facilitates grass roots collaboration out side of management hierarchy. The aim of the system is help employees make connections with people, through volunteering, to get work done.

*Signaling:* Similar to open source communities signaling is used in the system to attract users to participate. These include

- Tags that identify project
- Number of people enrolled in a project
- Color coded messages to highlight criticality

*Task Detail:* The website allows users to browse tasks, view projects of interest, enter profile information and post tasks. Depending on individual's interest, he/she the system recommends tasks that match his/her profile or recommend projects that others with similar interests are participating in.

*Team Building:* The system is intended to create relationship through volunteering - to get work done. It is different from other collaborative approaches used internally like Jams -to hold, sort and discuss ideas. The system is centered on the idea of users volunteering for work usually out side of the scope of their day-to-day jobs. In this the system provides a vehicle for people to expand their job scope, network, reputation, and skills.

### *Building Reputation*

Collaborative approach in open source development relies on meritocracy based on reputation. IBM developers participating in open source development are motivated to build their reputation in the online external communities. This reputation translates internally to job role or job title in IBM. I have identified IT architects<sup>x1</sup> and technologists<sup>xli</sup> who participate in IBM open source projects and are playing an important role in the open source community. The IIOSB platform details information about the developer including membership history and past projects.

For non developers or developers not involved with open source projects building reputation among the internal community can be difficult. People need to work together to build trust and building trust then translates into reputation of the person. To solve this issue IBM has created a Web 2.0 social networking internal site that helps employees connect with each other. The system relies on a commonly used chat tool plug-in to understand a user's team member by studying frequency of communication with others.

The web site has a profile page similar to that of any social networking site which lists a user's interest and skills. These skills are identified by the user and can range from technical skills to interpersonal to specific subject matter. The chat plug-in requests team members to rate a person's skill level based for the skills entered by the user on the social website and anonymizes the results to come up with a score. This score is then compared to other developer's score with similar skills and ranked as top 10%, top 20% and so on. The ranked score translates to a developer's reputation. . It also request specific feedback related to user's performance in the team. The reputation and specific feedback is displayed on the user's profile page on the social networking site.

In this manner the chat plug-in carries over a user's reputation in all collaborative effort the user is involved in. This system is also integrated with IIOSB and the task bartering system and carries the developer's reputation to the open source development model. Team members look at this reputation to gauge skill level of a new user.

### *Reward*

All IBM employees are eligible for profit sharing and reward based on performance. At the beginning of the year all employees commit to a set of goals to achieve for the year that align with the business objective of the group or division. Employees participating in the open source development effort (Linux, Eclipse) therefore commit to goals that are based on their performance in the development of open source software. These goals are then approved by the developer's manager. Since participation in open source is under the supervision of the manager and the manager is involved with the developer's project undertaking, developer's performance is judged on the committed goals at the end of the year. This method of appraisal is no different from ant traditional approach in traditional software development.

Employees that participate in open source projects in their personal free time do not gain any monetary benefit in participating in open source project. They can add skills and build reputation to their profile which can in turn help them advance their career, but they are only appraised on the goals that they committed to on traditional projects.

If developer's join open source projects on their personal time they may or may not choose to work on a task even after joining a team. Team members in these projects don't have any authority over what role others play and no control over when the task is completed. This is similar to F/OSS model where no one has any authority over others action, if team members choose to pick a task it gets done otherwise the task does not get done. Therefore unsatisfactory performance by a team member can only result into lower

reputation of the developer but not lesser monetary compensation. The developer gets compensated based on his/her performance on his primary responsibilities in the traditional model.

### Automating Project Management

IBM uses automated tool to manage the project activities that are performed by project manager such as communication, statusing, monitoring of tasks and gathering requirements. IBM's Internal Open Source Software Bazaar (IIO SB) is used to host open source projects internally and is based on GForge<sup>xlii</sup> by GForge Group tool that provides the collaborative development environment. The site supports hundreds of projects and thousands of developers working on open-source projects that IBM doesn't want exposed to outside parties. Case studies used in this thesis look at two examples of how IBM is using the Open Source development process internally and supplement the role of a project manager using this development tool in managing the projects.

The GForge AS is completely rebuilt to make a modern, extensible platform with an intuitive interface that ties together a huge toolset, from Source Code Management (SCM) to extremely customizable Trackers, Task Managers, Document Managers, Forums, and Mailing Lists. All of these are controlled by a centralized permission system and maintained automatically by the system.

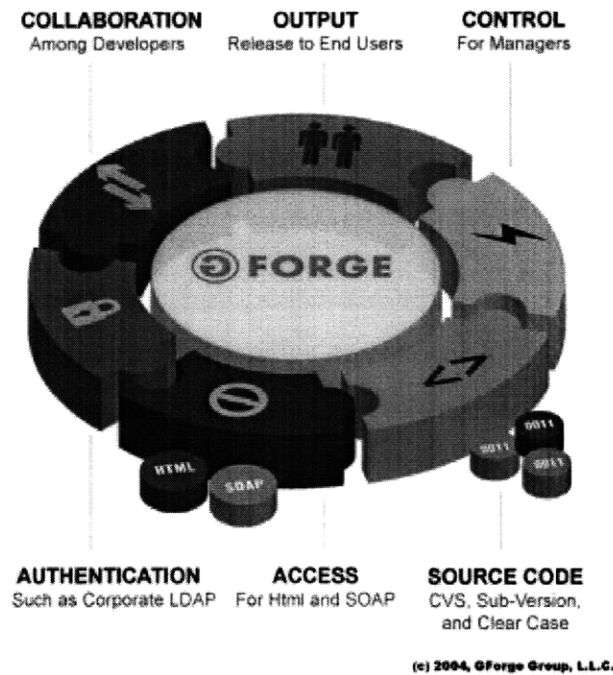


Figure 3: G-Forge collaborative automated tool components

The collaboration tool automates a number of tasks that are performed in a project environment and publishes information on the web for all participants of the community to view. The main features of the tool are

- Create project
- Edit your trackers, or create new trackers, fields, and values in your fields
- Edit your 'roles' and 'Observer' settings to set permissions and access levels
- Add members to your project
- Plan releases
- Create tasks in your Task Manager and use MS Project to estimate ETA on completing them
- Start writing your code and commit code linked to your defined tasks
- Build and release code using the File Release System
- Log defects against the releases
- commit changes and link the changes to the defects

Both case studies attempt to understand automating project management function internally, focusing on projects that are not directly involved with Linux, Apache or Eclipse development. The first case study was chosen since it uses a mixed Open source licensing for developing internal application using code from external source. The code with this licensing remains internal to IBM. The application uses Linux code in the project to build an internal application using open source methodology of collaboration. This is an active project (70%+ activity based on contributions and communication via mailing list) that is driven by time based release of software builds on the client side in a client server architecture model.

The second case study was chosen since it uses an IBM Public License (IPL) or CPL (Common Public License) and the code for it will be released externally to open source. This is a server side application that is driven by functionality. This project has had little activity after the first release.

# Case Study: Analysis of Linux based desktop client

Developing the desktop software occurs through the IIOSB portal that serves as a global information-sharing workplace and collaborative software development environment.

Summary | Home Page | Tracker | Lists | Tasks | Docs | News | SCM | Milestones | Guidelines

Identify/define the requirements for a standard client (Client for e-business) that will meet the needs of the IBM internal Linux development community. Architect an Linux Install service to deliver the Linux C4eb image.

Project Map categorization:

- Development Status: [5 - Production/Stable](#)
- Environment: [Gnome, KDE](#)
- Intended Audience: [Developers, End Users/Desktop, System Administrators](#)
- License: [IBM Internal/Mixed OSS](#)
- Operating System: [Linux](#)
- Topic: [Desktop Environment, Office/Business, Software Distribution](#)

Registered: 2000-05-03 00:30  
Activity Percentile: 71.07%  
View project [Statistics](#) or [Activity](#).  
View list of [RSS feeds](#) available for this project

License Zone  
IBM Internal / Mixed OSS  
[View Zone Definitions](#)

Developer Info  
Project Admins:  
[Thiago Cangussu De Castro Gomes](#)  
[Bob Bovaird](#)  
[Srinivas Kollu](#)  
[John Walicki](#)  
[Greg Fischer](#)  
[Eric Barkie](#)  
[D. Williamson](#)  
[Michael Little](#)

[\[View Members\]](#)  
[\[Request to join\]](#)

Public Areas

[Project Home Page](#)

[Tracker](#)  
- [Patches](#) ( 6 open /26 total )  
A collection of submitted source patches  
- [Feature Requests](#) ( 171 open /875 total )  
A collection of submitted feature requests  
- [c4eb-live](#) ( 1 open /1 total )  
C4eb live (bootable CD-ROM) bugs & feature requests  
- [c4eb-layer](#) ( 0 open /597 total )  
The RPMs that comprise the C4eb layer of applications  
- [Open Client Build System](#) ( 5 open /35 total )  
A TODO Tracker for the Open Client Build System  
- [Bugs](#) ( 574 open /4074 total )  
A collection of submitted bug reports

[DocManager: Project Documentation](#)

[Mailing Lists](#) ( 11 public mailing lists)

[Task Manager](#)  
- [Distros](#)  
- [OAS Image Testing](#)  
- [Redhat errata](#)  
- [NLS](#)

Latest News

**Open Client 1.5 Release**  
[Michael Little](#) - 2008-09-29 13:12 - [Linux Client for e-business](#)  
Open Client is ready for the IBM enterprise!  
(0 Comment) [\[Read More/Comment\]](#)

**Expanding the Open Client for Linux Desktop Offerings**  
[Bob Bovaird](#) - 2008-09-11 13:21 - [Linux Client for e-business](#)  
See this announcement posted off our Linux Home page <https://w3.tap.ibm.com/w3ki/display/Linux/Expanding+the+Open+Client+for+Linux+desktop+offerings>  
(0 Comment) [\[Read More/Comment\]](#)

**Open Client for Linux Fedora Edition available**  
[John Walicki](#) - 2008-09-11 13:20 - [Linux Client for e-business](#)  
In a partnership with the LTC and the internal Linux community, the Open Client Integration Team has started development on an "Open Client for Linux Fedora 9 Edition". Currently they are in its initial phase of the deployment, which provides users with either an option to install either the Open Client Base Layer (Security which meets ITCS 300 requirements) or the Full Open Client Layer (Lotus toolset, Office Productivity, etc.). For installation and more information about these efforts refer to the wiki <http://w3.tap.ibm.com/w3ki/display/Linux/Open+Client+for+Linux+Fedora+9+Edition>  
(0 Comment) [\[Read More/Comment\]](#)

- **Labor Day Weekend Outage (8/29 - 9/2)** 2008-09-04 13:53
- **Open Client for Linux 2.2** 2008-06-16 15:28
- **Open Client 1.4 is released.** 2008-01-23 13:48
- **Open Client for Linux 2.1 announced** 2007-11-15 13:30
- **Open Client for Linux 1.3 Gold Master** 2007-10-04 10:53

[\[News archive\]](#)

Screenshot 1: Linux desktop client Home page on IIOSB

Project initiator can be an individual or a team that are responsible for the project. The project owner decides the licensing for the project which then determines the use of the code in the future. More restrictive licensing means the project contains code from open source but will be used internally for IBM use only. When the project uses a less restrictive licensing it can include other open source code outside IBM and the code will be available externally. Contributors join the project directly by signing up at IIOSB or through the task brokering system. The project team is an emergent one where team members have in a sense discovered each other, and have brought together their individual competencies and contributions. Corporate executive has little administrative authority to determine:

- Project scope

- Project schedule
- Tasks and resource determination
- Task and resource assignment
- Evaluating project team performance

The project follows a time based release with input from project admin, core developers and the IBM Open source Software committee. These release dates are used to create a project roadmap and shared with all members of the project team. The project admin and core developers influence the team members to deliver individual tasks and encourage user participation to meet the release date. Production ready commits to the code on the release date are taken into the build and released to the user. Admin and core developers divide the tasks into smaller tasks that can be picked up by developers, contributors and casual volunteers. Keeping the tasks small and manageable reduces the administrative overhead required to sustain ongoing software development and encourages participants to interact more frequently. Frequent interactions lead to trust amongst the team members and helps build reputation of the participants.

### Applying Fayol's principles and project management toolkit

Applying Fayol's 5 principles to IBM open source initiative and using the project management toolkit as a viewing lens, it becomes clear that IBM's application of open source development approach is similar to the F/OSS in some ways and a hybrid of traditional and F/OSS for some aspects of project management.

#### Planning

According to Fayol's five principles, Planning consists of Requirements and *Opportunities and Risk Management* in IM's open source approach. Both Requirement and Opportunities and risks are proactive components of project management. A project manager is responsible for crating a baseline plan by understanding the requirements and determining scope according to the requirements.

Project manager is required to understand risks in the project and has to take proactive steps to mitigate these risks through out the project lifecycle. In IBM's Open source projects, risks and opportunities are constantly evaluated and acted upon by project team through out the project.

Both of these steps are performed in IBM open source projects similar to F/OSS.

## *Requirements*

Requirements can be added to the project by any participating member of the IIOSB community. Each requirement details the tasks and importance of the requirement to the project. Since members of the project are technical developers, these requirements are generated by users of the code that have a need for a better solution. Project team then votes on the requirement, if the requirement meets the threshold it is then reviewed by group of core members and reviewers. The requirement is then assigned priority based on the recommendation from project admin, core developers and reviewers. For time based release projects the requirement can be added to the current release or postponed for the next release depending on the need and criticality of the requirement. Once the requirement is accepted in the release and participant can contribute to the code for the requirement.

In reviewing the requirements for the Linux desktop, it becomes clear that the requirements are more technical and posted by technical developers since it includes references to the code. These technical developers are interested in improving the functionality of the application to suit their needs. The fact that these developers are technical is demonstrated by looking at their job title and a number of them belong to the IBM global services organization, the group, which is using Linux in its data centers supporting external clients running Linux. To submit the requirement a developer needs to sign in to the IIOSB web site and add the details about the requirement. Information such as submitter's name, date submitted and a brief description of the requirement is displayed in the IIOSB tool for others to vote on. An e-mail message to all the project members is sent with the description of the problem with an action to vote on the requirement. If the number of votes adds up to the threshold level set by the admin at the start of the project, another system generated e-mail is sent to the admin and core developers to assign priority to the requirement. The project admin and core developers then review the requirement and assign priority to the requirement. The priority is another way to highlight what is critical to the project or must have for the application. A color coded priority mechanism developers signals to the developers where to direct their effort and have the maximum impact on the application though it does not limit code contribution to non critical requirements. A priority coded as a 1 is colored red and has the highest priority determined by the admin.

This mechanism of assigning priority by users of the system ensures that the requirements are in line with the project's scope. Voting and prioritizing of the requirement by application users also ensures that there is a need for a particular requirement, a need that is more in line with the project's expectation. This is different from traditional approach where a group of project stakeholders decide on the priority based on factors such as submitter's position or submitter's relationship to funding organization. In my view



requirements that bubble up to the top by voting are based on the market needs rather than internal needs of the project.

Order by: (2) ID Ascending Quick Browse

ID	Summary	Open Date	Priority	Assigned To	Submitted By
32166	<a href="#">GUI for iptables firewall</a>	* 2004-03-21 07:06		Scott Wilson	Thomas D. Dahmann
35453	<a href="#">Life boat default partition space allocation</a>	* 2004-08-26 16:08	4	Eric Barkie	Greg Fischer
42934	<a href="#">Broadband Install Disclaimer</a>	* 2005-04-25 16:28	3	Scott Hill	Michael Quick
48446	<a href="#">remove the need for root</a>	* 2005-10-17 23:22	5	Eric Barkie	Greg Fischer
55144	<a href="#">Integrate Lifeboat Bootable Image into GUI</a>	* 2006-05-17 10:13	4	David Koeller	Tim Milberger
56459	<a href="#">MAke Notes spellcheck more ergonomic.</a>	* 2006-07-04 06:13	3	Derek Burt	Richard John Moore
58078	<a href="#">Add security classification as a part of install process</a>	* 2006-08-28 16:44	3	Eric Barkie	Bill Oswald
59572	<a href="#">xorg EmulateWheelTimeout bug</a>	* 2006-10-07 16:21	3	Nobody	Thomas Rasmussen
59813	<a href="#">Give users the ability to install and uninstall a "layer" on their existing Open Client image.</a>	* 2006-10-13 16:44	3	Eric Barkie	Joanne Icken
66758	<a href="#">PCMCIA : Orange UMTS/3G/EDGE</a>	* 2007-02-19 10:27	3	Gerd Stahl	Jerome Chailloux
66881	<a href="#">Include Fluendo Windows Media and MPEG codec support</a>	* 2007-02-21 10:08	3	John Walicki	Greg Fischer
69665	<a href="#">ipm-print-ibmatt-0.30.1-14 is missing</a>	* 2007-03-30 08:00	3	Nobody	Ralf Kopke
73376	<a href="#">T60p widescreen support - xorg changes</a>	* 2007-05-17 10:56	3	Nobody	John Walicki
73907	<a href="#">Missing Pop-up in the initial Asset Management screen when Next is clicked without Intranet ID.</a>	* 2007-05-23 13:35	3	Nobody	Eddie Dean
75853	<a href="#">WECM uses too much resource when its doing nothing</a>	* 2007-06-13 11:44	3	Michael Little	Eric Barkie
76225	<a href="#">Investigate an enhancement to IBM Easy Update to show multi-download progress bars when using Grnd</a>	* 2007-06-18 11:37	5	Hugo Leonardo De Oliveira Melo	John Walicki
76455	<a href="#">Wireless LED on T60p doesn't blink</a>	* 2007-06-20	3	Nobody	John Watzke

Screenshot 2 : Requirements gathering for Linux based Desktop

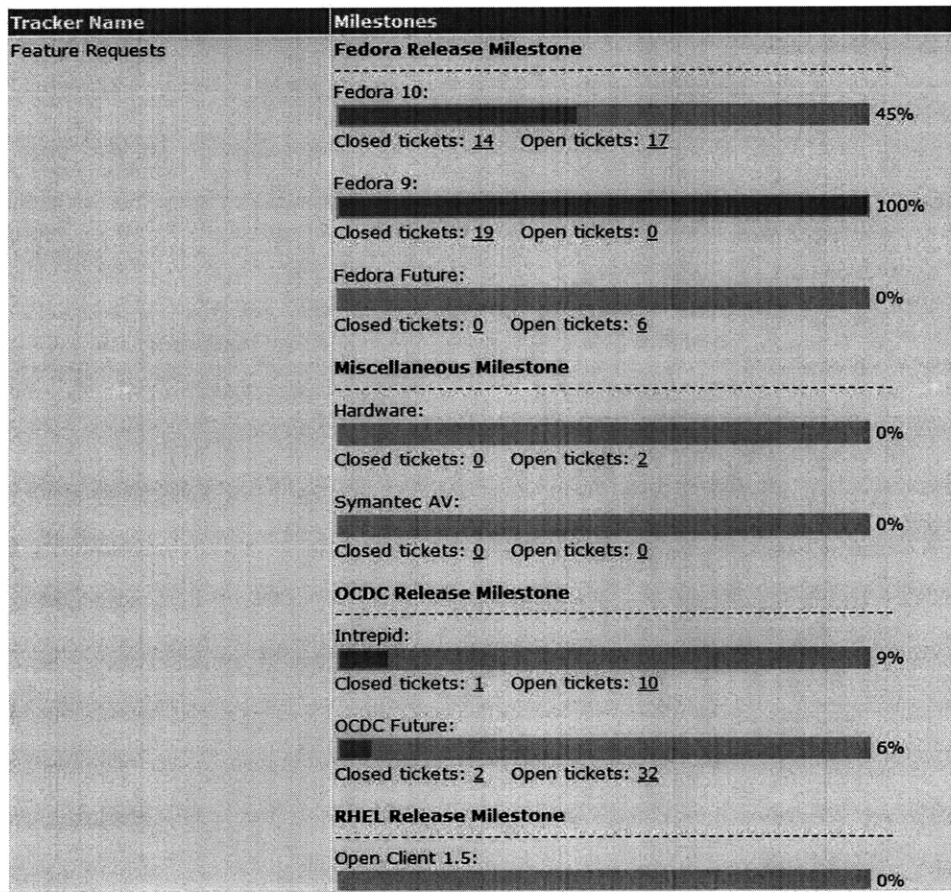
### Project Planning

Linux desktop project follows a project roadmap that has been prepared with project admin and business division executives that are responsible to drive the Linux adoption internal in IBM. This is an executive committee that has the overall responsibility to lead the project to success. The roadmap offers a clear view of how application will be adopted in the organization and reaches technical maturity to be enterprise ready. This roadmap is used by developers to assign priority to the requirements added by users of developers of the system. This approach differs greatly from the F/OSS development approach where there is no to-do list maintained and whatever production ready code is committed is added to the release. The executive committee plays the role of overseer of the project with the responsibility of reaching the strategic goal for the company and also a guide responsible for charting a path for the application internally.

## 2009 Roadmap

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Open Client RHEL 1.x												
Open Client RHEL 2.x		<u>OC 2.3</u> X					OC 2,4 GA					
Open Client Fedora			<u>OC F10</u> Beta	<u>OC F11</u> Beta	<u>OC F11</u> RC	<u>OC F11</u> GA						
Open Client RHEL 3 beta												
OC Debian Community	<u>OCDC-Intrepid</u> Beta	<u>OCDC</u> <u>Intrepid</u> GA		<u>OCDC-Jaunty</u> Beta	<u>OCDC</u> <u>Jaunty</u> GA					<u>OCDC</u> <u>Karmic</u> Beta	<u>OCDC</u> <u>Karmic</u> GA	

Screenshot 3 : Project Roadmap for Linux based Desktop



Screenshot 4 : Project milestone for Linux based Desktop

### *Opportunity and Risk:*

A buggy code present the biggest risk to any software development effort and in internal open source project at IBM this risk is handled automatically using Code versioning system that track the changes made to the code. Similar to F/OSS approach, users have to gain a reputation of producing bug free code before they are granted a status to commit code in the archive. The CVS has the ability to reconcile the code submitted by multiple users for the same module and highlight the differences and apply the patch to the final code, thus limiting the need for co-ordination and at the same time giving users access to information about the code changes. CVS system also allows for code to be reverred back to the last successful version of the code, which gives the admin the ability to undo a change in the software is a bug is detected after a release of the code. Using the automated tool in this manner reduces the overall risk of the project and at the same time provides visibility to the changes to all.

Code released and hosted on the IIOSB website is available for all users to download and is not restricted to the project members. So anyone who wants to contribute to the project or understand the functionality of the code has equal access to the code repository This mechanism allows for non-project members to participate in the project give valuable feedback related to code performance, bugs or improvements to be added. Using the IIOSB website in this manner reduces the risk associated with a buggy code and at the same time opens the project to valuable input from new set of users of the code.

### Organizing

The key component of the project is the project team. In IBM the approach to organizing is based on Self-directed working teams in the project environment, but there are some elements of traditional approach to managing and forming teams.

### *Project Team*

Since the Linux desktop project is critical to IBM's long term open source strategy, a group of executives play an important role in guiding the project. They are responsible for reviewing the project status and creating a project roadmap for development and adoption across IBM. The executive team however does not play a role in assigning a project admin or choosing core developers. It also does not restrict participation of developers to the project. It does, however verify that the code generated is reviewed by IBM's legal team since it uses outside open source code internally to build the application. Since the desktop application is to be used internally it does not need as through a review as something that would be released to open source community.

The project team is consists of project admin and core set of developers who have similar needs and are familiar with Linux environment. As displayed in their position title, most of the project team members are developers or server administrators that should have experience with Linux operating system. Project community is based on skill meritocracy where members of the team follow few explicit rules about what development tasks should be performed, who should perform, when, why, or how. As members contribute to the system they gain the trust of the project admin and are given authority to become part of core developers and have commit access to code repository. The project team is fairly stable throughout the project lifecycle which is different from an open source project where the team is always fluid. Besides the roles mentioned above project team consists of reviewers, testers, document editors and passive users.

Administration of the project also involves the systems manager of the division that intends to use the application and an IBM legal counsel. Legal counsel ensures the licensing use for the project and the systems managers ensure that the project meets the needs of the division.

If you would like to contribute to this project by becoming a developer, contact one of the project admins, designated in bold text below.

Developer	Username	Email	Role/Position
Andre Fernandes De Macedo	afmacedo	afm@br.ibm.com	Senior Developer
Alexei Znamensky	alexreiz	alexreiz@br.ibm.com	Junior Developer
Anton Piatak	antonpiatak	anton.piatak@uk.ibm.com	Senior Developer
<b>Bob Bovaird</b>	bbovaird	bbovaird@us.ibm.com	Admin
Munilo Fernandes Bernardes	bernarde	mfb@br.ibm.com	Senior Developer
Anne Brinkman	brnika	brnika@us.ibm.com	Senior Developer
Colm Malone	colm	colm@us.ibm.com	Senior Developer
Nancy Wei	crab	weinancy@us.ibm.com	Senior Developer
Derek Burt	derekburt	dburt@ca.ibm.com	Senior Developer
DIEGO LALO DE MAURO	diegolo	diegolo@br.ibm.com	Senior Developer
David Koeller	dwk	dwk@us.ibm.com	Senior Developer
<b>Eric Barkie</b>	sbarkie	davnul@linux.vnet.ibm.com	Admin
Frank Bagehorn	fb	fb@zurich.ibm.com	Senior Developer
Frank Heimes	fheimes	frank.heimes@de.ibm.com	Senior Developer
Firas Bouz	fr	fr@us.ibm.com	Senior Developer
Lawrence Futrell	lufrell	lufrell@us.ibm.com	Senior Developer
George Kraft	gk4	gk4@us.ibm.com	Senior Developer
<b>Greg Fischer</b>	gofish	gofish@us.ibm.com	Admin
Gustavo Yokoyama Ribeiro	gyr	gyr@br.ibm.com	Senior Developer
Victor Herrero	harrero	harrero@us.ibm.com	Senior Developer
Scott Hill	hills	hills@us.ibm.com	Junior Developer
Hugo Leonardo Da Oliveira Melo	huogomelo	huogomelo@br.ibm.com	Senior Developer
Ido Levy	idol	idol@il.ibm.com	Senior Developer
Ian Shields	ishields	ishields@us.ibm.com	Junior Developer
JAYAVARDHAN Katta1	jay	jayavardhan.katta@in.ibm.com	Junior Developer
Jane Martinko	jtinko	jtinko@us.ibm.com	Senior Developer
Srinivas Kollli	kollli	kollli@us.ibm.com	Admin
Lou Zapata	lzapata	lzapata@us.ibm.com	Senior Developer
<b>Michael Little</b>	mlittle	mlittle@us.ibm.com	Admin

Screenshot 5 : Project team members for Linux based Desktop

### Project Teamwork

A project initiator or admin starts a project and completes a checklist to identify the suitable licensing type, operating system, intended audience and the environment. The project admin is ultimately responsible for the success of the project but cannot assign tasks to others – similar to open source development. Participants nearer the project core have greater control and discretionary decision-making

authority, compared to those further from the core. However, realizing such authority comes at the price of higher commitment of personal resources such as the ability to convince other participants to the viability of a decision, take position on issues, communicate and create project content. Thus, developers possessing and exercising such skill may be intrinsically motivated to sustain the evolutionary development of their project so long as they are active participants in the project community.

Once the project is initiated the admin relies on core developers to contribute to the project and testers to test for functionality. The contributors of the project have to adhere to the IBM's internal open source guidelines but there are no rules to govern the collective action of the participants within the project. Similar to an open source project, the Linux desktop project participants contribute iteratively to the code and cannot be assigned a task. Depending on the task or based on previous performance of the project admin, contributors join the project team. A number of signaling mechanisms are used to attract developers to contribute to the project, that include reputation of the project leader, number of users already attached to a project, number of contributions added to the project or recognition of code contributed to the project. Project members contribute to the project and earn the right to add to a release and get commit access to CVS repository. The project member has the ability to assign access to commit code to CVS and in most cases only trusted members of the team get the commit access.

A task is accomplished when there are participants interested in coding the requirement and testers willing to test the change. The IIOSB and the internal social networking website provide the project team an assessment of a developer's contribution and a measure of his/her reputation in contributing to open source projects in the past.

Before the release of the application project administrator reviews the high level changes to the code with the systems manager to ensure that the project adheres to the project roadmap. The legal counsel ensures that the code used in development meets the licensing requirements.

The team members resolve conflicts by continuously-negotiating with each other by communicating frequently with others thereby encouraging cooperation and coming to a resolution. The IIOSB portal allows for collaboration with team members using mailing lists, wikis, and blogs and allows for frequent communication with the members of the project.

### Controlling

The rules of governance and control in the IIOSB projects are clearly articulated and recognized by project participants. These rules serve to control the rights and privileges that developers share or delegate

to one another in areas such as who can commit source code to the project's shared repository for release and redistribution.

All projects internally developed using the open source model is controlled to some degree depending on the licensing assigned to the project. Projects with restrictive open source licensing that are intended for internal IBM use only, are controlled to ensure they follow the project path for development internally. Projects with less restrictive open source licensing are controlled and reviewed at different levels by executives and legal team to ensure no IP is given away in releasing the code externally. These less restrictive licensing also require participation from external developers and therefore controls are put in place to reduce IBM's exposure to risk.







### *Project Control*

Rules of control are also expressed and incorporated into the source code itself, in terms of how, where, and when to access system-managed data via application program interfaces, end-user interfaces, or other features or depictions of overall system architecture. Subsequently, project participants self-organize around the expertise, reputation, and accomplishments of core developers, secondary contributors, and tertiary reviewers and other volunteers. This, in turn, serves to help them create a logical basis for their collective action in developing the software

Certain decisions like working on an application of strategic importance may still be taken by senior executives at a very high level of the project but the inner working of the subprojects is handled by core developers. Creating a project road map and requiring the project to adhere to the roadmap also creates a mechanism to control the development of application.

To control the progress of the project and ensure the delivery of the release on time, the IIOSB system uses the project tracking component that details the necessary information for the project team to act on. This includes tracking patches applied, submitted requirements, bugs reports and to-do lists of items to be completed.

Choose a tracker and you can browse/edit/add items to it.

Tracker	Description	open	total
 Patches	A collection of submitted source patches	6	26
 Feature Requests	A collection of submitted feature requests	171	875
 c4eb-live	C4eb live (bootable CD-ROM) bugs & feature requests	1	1
 c4eb-layer	The RPMs that comprise the C4eB layer of applications	0	597
 Open Client Build System	A TODO Tracker for the Open Client Build System	5	35
 Bugs	A collection of submitted bug reports	574	4074

Screenshot 6 : Tracking options for Linux based Desktop

## Coordination

Coordination is key to success of a project, once the project is in the execution stage of the project lifecycle. As in the case of F/OSS development approach, IBM uses automated tools for team members to communication and coordinates project activities within the team. The frequent incremental communication between the team members using automated tools such as mailing lists, wikis and blogs provide a common low cost solution to coordinating activities in the team. This reduces the overall administrative cost for managing the project and at the same time builds trust between team members by requiring frequent communication.

### *Project Communication*

In the Linux desktop application, communication between team members is done primarily through mailing lists. These automated mailing lists cover topics such as commit to archive, contributions by team members to archive, discussion topics, news related to the project etc. Any one the internal open source community can subscribe to a mailing list and get updates as members add code or contribute to the project. This ensures equally sharing of information about the project not just with immediate team but also with anyone who is interested in project.

The use of automated tools to communicate with others is similar to the mechanism used in open source communities. The GForge tool, used internally to host the projects, also provide as hub for all documentation, status and project related material. The open access to information about the project ensures a level playing field for all team members in the team and allows users to work independently in virtual teams at the same time staying connected with the project.

## **Mailing Lists**

Choose a list to browse, search, and post messages.

Mailing list	Description	Subscription
<a href="#">linuxc4eb-commits Archives</a>	CVS commit messages	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-contrib Archives</a>	Linux Client for e-business contributors discussion	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-devel Archives</a>	Linux Client for e-business developers discussion	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-discussion Archives</a>	Linux Client for e-business general discussion	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-managed-dev Archives</a>	Linux Managed Client for e-business developers discussion	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-ocdc-dev Archives</a>	Open Client for Linux Debian Community Developers	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-ocdc-discuss Archives</a>	Open Client for Linux Debian Community Discussion	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-ocdc-news Archives</a>	Open Client for Linux Debian Community Announcements	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-packages Archives</a>	Linux Client for e-business package history	<a href="#">Subscribe/Unsubscribe/Preferences</a>
<a href="#">linuxc4eb-wireless Archives</a>	Linux Client for e-business wireless discussion	<a href="#">Subscribe/Unsubscribe/Preferences</a>

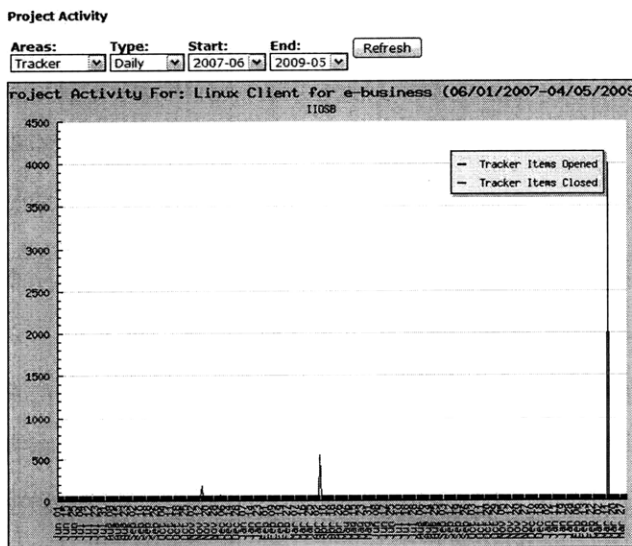
Screenshot 7 : Mailing lists for Linux based Desktop

## Project Status

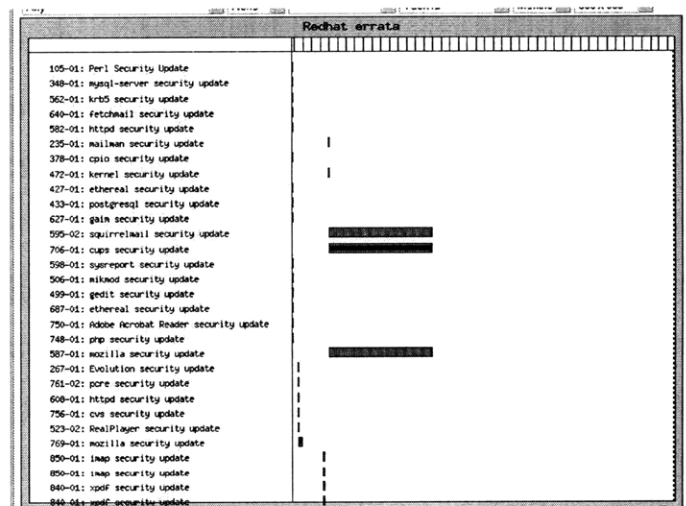
This is an important administrative project management task that is automated using technology. The open source development platform allows users to gather status automatically quickly using pre defined reports in the system.

The project home page shows details related to licensing, the development environment, administrator of the project, activity in the project, the level of release the code is in and so on. This information is needed on a regular basis for people who monitor the project in traditional environments, but in the open source project this information serves to put credibility to the work that is being done in the project.

Transparency in the project motivates the team members to perform at the same time reduces biases in the team and ensures a suitable environment for the project. The IIOSB web site allows anyone in the internal open source community to look at project activity and status of tasks performed by the project team with the ability to create custom reports. The data behind the project activity can also be exported to popular tools like Microsoft project, Excel or OpenWorkbench to do further reporting on the project.



Screenshot 8 : Project activity for Linux desktop



Screenshot 9 : Gantt chart for Linux desktop project

## Commanding

Command over the project requires include corrective actions that can be taken when the project is behind schedule or moves away from the end state desired by the users. In IBM's internal projects the corrective action is applied to the project in two ways (i) to remove bugs (ii) revert changes to code.



## Corrective Action

To report and submit bugs in the code users including non team members can report a bug in IIOSB. The tool provides submission guidelines to users that ensure easy identification of the bug by developers. Bugs can be monitored via mailing lists and any update to the bug fix is reported to all members of the project team instantly. Bugs are also color coded with priority that communicates the urgency of the fix required to developers.

Order by: [L2](#) | [ID](#) | [Ascending](#) | [Quick Browse](#)

ID	Summary	Open Date	Priority	Assigned To	Submitted By
108323	<a href="#">BUG: Notes 8.5 Calendar Advance Issue</a>	* 2009-03-03 12:33	3	Derek Burt	KEN KRAUSE
108325	<a href="#">BUG: "Open" attachment in notes 8.5.1 does nothing</a>	* 2009-03-03 14:37	3	Derek Burt	Brian Olore
108379	<a href="#">BUG: Lotus Notes Java API under Ubuntu</a>	* 2009-03-04 09:47	3	Derek Burt	Jonas Herkommer
108398	<a href="#">Bug: You computer is not equipped with a biometric fingerprint reader</a>	* 2009-03-04 12:11	3	Nobody	JOHN HASTY
108403	<a href="#">Bug: Notes 8.5.1 doesn't archive</a>	* 2009-03-04 12:39	3	Derek Burt	Dave Bachmann
108405	<a href="#">Bug: Add Sender to Address Book broken in Notes 8.5.1 revision 20090218.1657</a>	* 2009-03-04 13:51	3	Derek Burt	Dave Bachmann
108406	<a href="#">BUG: firefox crashes</a>	* 2009-03-04 14:20	3	Nobody	Brian Horton
108408	<a href="#">Bug: File Open dialog not shown in Notes 8.5.1</a>	* 2009-03-04 14:49	3	Derek Burt	Roberto Figueiredo Salomon
108435	<a href="#">BUG: chkconfig in ocde breaks jaunty</a>	* 2009-03-05 03:02	3	Nobody	Anton Piatek
108436	<a href="#">BUG: ocde notes layer needs dependency OR</a>	* 2009-03-05 03:04	3	Derek Burt	Anton Piatek
108440	<a href="#">BUG: Sametime Privacy Preferences page incorrectly sized</a>	* 2009-03-05 05:08	3	Derek Burt	M. Cocker
108496	<a href="#">[BUG] OCDC -- Issue after running beta Lotus Notes 8.5.1</a>	2009-03-06 04:11	3	Derek Burt	MARIUS Bock
108506	<a href="#">Bug: Lotus Notes Menu problems</a>	2009-03-06 10:32	3	Derek Burt	Steffen Waitz
108507	<a href="#">Bug: Errors in provisioning after clean install of 8.5.1 20090218.1657</a>	2009-03-06 10:35	3	Derek Burt	Tim Clark
108511	<a href="#">Bug: Problems with DOS formatted files in an 8.5.1 install package.</a>	2009-03-06	3	Derek Burt	Tim Clark

Screenshot 10 : Bug report for Linux desktop project

## Project Leadership

Project leadership plays an important role in internal IBM open source projects. At the highest level in the company the open source initiative has an executive sponsor for all open source projects in the company. The leadership is carried through from the business executive level to division level and down to the project level. This ensures that open source projects like Linux desktop get the required oversight and guidance from the executive level. The project leadership is needed to remove ambiguity from the project and allow the team to focus on core development activities.

Linux desktop project is supported by a leadership team consisting of systems manager from software and hardware divisions along with legal counsel from IBM corporate. This oversight committee is responsible

for reviewing the licensing agreement, understand IBM's risk exposure, allocate legal resources and align the project initiatives with IBM's goal.

In another example of an open source project that uses open source code and carries the GPL licensing, a project oversight committee guides the project through the rigorous review cycles before releasing the code back to open source community. These reviews are meant to understand the implication of releasing IBM developed code to open source and ensure IP protection.

## Conclusion

The internal Linux development project in IBM using the open source methodology gives a better understanding of limitations and benefits of using the open source approach internally in a company. Project management literature review gives us the key elements that are needed for projects to succeed. Applying the open source development technique to a project can reduce the amount of overhead required in the project. In open source projects there is little or no need of project managers to actively manage the projects but in applying the open source model there are some project management elements that cannot be applied as is from the open source development model.

The role of the organization and the business strategy pursued by the organization is important in applying the open source development models internally since it creates the right environment to collaborate amongst the team members. It is clear that project managers are not needed in managing the project, as demonstrated in the Linux desktop project but an oversight committee of sorts is needed for smoother functioning of the project. This oversight committee is responsible for governing the project including development road map, providing direction and aligning the project to company's goal. Without the oversight committee, project team may not be able to jump through legal hurdles and get backing to continue developing the project.

Technology and the use of automated tools ensure a smoother communication between the distributed virtual teams. The tools also provide a mechanism to coordinate activities and ensure better communication between the team. By using automated mailing lists, all participants of the project can have access to project information to make the right and timely decisions. The project team can function smoothly using the automated tools that provide information about team members including skill set and reputation. Automated tools allow the project team to interact frequently and iteratively over the course of project life cycle that builds trust amongst the members which eventually allows a user to build up his/her reputation amongst the community. Participation by employees in the open source project does not result in better remuneration but does allow the individual to learn new skills and apply these skills to project to gain a reputation. This reputation can translate to a better position for the participating employee in the future or give him the ability to become a subject matter expert in his/her organization.

The technical execution including planning, control, preventing bugs etc of the project life cycle can be done using automated tools that reduce the risk of project overrun and at the same time reduce overhead cost related to quality control.

The table below summarizes the Fayol's principles and applies the project management toolkit along with automated tools that can be performed in managing a project without a project manager.

<b>Fayol (1916)</b>	<b>Project Management Toolkit</b>	<b>Automated tool</b>
Organizing	Project Team	Forming Self Directed team – <ul style="list-style-type: none"> <li>• using social networking,</li> <li>• task brokering,</li> <li>• Individual participation metrics measurement to build reputation</li> </ul>
	Project Teamwork	Forge – ex G-Forge that provide all information about the project in one place
Planning	Project Planning	-None
	Requirement	Requirement gathering using G-Forge, tracking requirements, automated mailing lists
	Opportunity and Risk Management	Code Versioning System Bug Reporting system Subversion system Open access to project via G-Forge platform
Controlling	Project Control	Project tracker Mailing lists Bug tracker
Coordinating	Project Status	Collaborative project website Mailing lists
	Project Communication	Mailing lists Project website Wikis, blogs
Commanding	Corrective Action	Code versioning system Subersion system
	Leadership	-None

Table 3: Using automated tools to Fayol's five elements of management

## References

- <sup>i</sup> Karim Lakhani and Eric von Hippel, No Managers Required: A case study of collaborative innovation using managerial toolkits
- <sup>ii</sup> Henri Fayol, *General and Industrial Management* (New York: IEEE Press, 1984).
- <sup>iii</sup> Tom I. Peters and R.H. Waterman, Jr., *In Search of Excellence* (New York: Harper & Row, 1974).
- <sup>iv</sup> R. Covey, *The Seven Habits of Highly Effective People* (New York: Simon & Schuster, 1989).
- <sup>v</sup> Frederick P. Brooks, Jr. "The Mythical Man Month". 1995. Addison-Wesley.
- <sup>vi</sup> Kevin Forsberg, Ph.D, Hal Mooz, PMP and Howard Cotterman: Visualizing Project Management: A Model for Business and Technical Success, Second Edition
- <sup>vii</sup> Technical Leadership Exchange April 6-9 2008 Orlando FL Copyright IBM Corp
- <sup>viii</sup> Perens, B. Open Source Definition from <http://ldp.dvo.ru/LDP/LGNET/issue26/perens.html>
- <sup>ix</sup> Raymond, Eric S The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary O'Riely Press 1999
- x Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers. *Research Policy*, 32(7) 1159–1177.
- xi O'Reilly, T. (1999, April). Lessons from open source software development *Communications of the ACM*, 42(4), 33–37.
- xii Ye, Y., Nakakoji, K., Yamamoto, Y., & Kishida, K. (2005). The co-evolution of systems and communities in free and open source software development. In S.Koch (Ed.), *Free/open source software development* (pp. 59–82). Hershey, PA: Idea Group Publishing.
- xiii Gloor, P. (2006). *Swarm creativity*. Oxford University Press.
- xiv Madey, G., Freeh, V., & Tynan, R. (2005). Modeling the F/OSS community: A quantitative investigation. In S.Koch (Ed.), *Free/open source software development* (pp. 203–221). Hershey, PA: Idea Group Publishing.
- <sup>xv</sup> Barabasi, A.L., & Bonabeau, E. (2003, May). Scale-free networks. *Scientific American*, 60–69.
- <sup>xvi</sup> Barabasi, A., Jeong, H., Neda, Z., Ravasz, E., Schubert, A., & Vicsek, T. (2002). Evolution of the social network of scientific collaborations. *Physica A* 311, 590–614.
- <sup>xvii</sup> van Wendel de Joode, R., de Bruijn, J., & van Eeten, M. (2003). Protecting the virtual commons. In T.M.C.Asser (Ed.), *Information technology & law series*, 44–50.
- <sup>xviii</sup> Lakhani, K.R., & von Hippel, E. (2003). How open source software works: 'Free' user-to-user assistance. *Research Policy* No. 32. Elsevier Science.
- <sup>xix</sup> von Hippel, E., & von Krogh, G. (2002). Exploring the open source software phenomenon: Issues for organization science
- <sup>xx</sup> Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm, *International Conference of Information Systems*, Brisbane, Australia.
- <sup>xxi</sup> Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K. (2006). Understanding free/open source software development processes.
- <sup>xxii</sup> Based on <http://www.linux.org/docs/lists.html>
- <sup>xxiii</sup> Healy, K., & Schussman, A. (2003). The ecology of open source development. Unpublished. Retrieved from [www.kieranhealy.org/files/drafts/oss-activity.pdf](http://www.kieranhealy.org/files/drafts/oss-activity.pdf)
- <sup>xxiv</sup> van Wendel de Joode, R., de Bruijn, J., & van Eeten, M. (2003). Protecting the virtual commons. In T.M.C.Asser (Ed.), *Information technology & law series*, 44–50.
- <sup>xxv</sup> Benkler, Y. (2002). Coase's penguin, or, Linux and the nature of the firm. *Yale Law Journal*, 112(3), 369–446.
- xxvi Himanen, P. (2001). *The hacker ethic and the spirit of the information age*. New York: Random House.
- xxvii von Hippel, E. (2001). Innovation by user communities: Learning from open-source software. *Sloan Management Review*, 42(4), 82.86.
- <sup>xxviii</sup> Saers, N. (2003). A project model for the FreeBSD project. Retrieved May 5, 2003, from: <http://niklas.saers.com/freebsd-model/freebsd-model.html>.
- <sup>xxix</sup> FreeBSD Foundation Board of Directors, from <http://www.freebsdoundation.org/board.shtml>
- <sup>xxx</sup> Mozilla Roles and Leadership from <http://www.mozilla.org/about/roles.html>
- <sup>xxxi</sup> Mockus, A., Fielding, R., & Herbsleb, J. (2000). A case study of open source software development: The Apache server. *Proceedings of 2000 International Conference on Software Engineering (ICSE2000)*, Limerick, Ireland, pp. 263-272.
- xxxii Koch, S. & Schneider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal*, 12(1), 27-42.
- <sup>xxxiii</sup> Sourced from <https://wiki.mozilla.org/ReleaseRoadmap>
- <sup>xxxiv</sup> Sourced from <https://wiki.ubuntu.com/IntrepidReleaseSchedule>

---

<sup>xxxv</sup> Sourced from <http://wiki.services.openoffice.org/wiki/OOoRelease31>

<sup>xxxvi</sup> Fitzgerald, B. (2005). Has open source software a future? In J.Feller, B.Fitzgerald, S.Hissam, & K.Lakhani (Eds.), *Perspectives on free and open source software* (pp. 93.106). Cambridge, MA: MIT Press.

<sup>xxxvii</sup> Hecker, F. (1999). Setting up shop: The business of open-source software. *IEEE Software*, 16(1), 45.51.

<sup>xxxviii</sup> Leiteritz, R. (2004). Open Source-Geschäftsmodelle. In R.A.Gehring & B.Lutterbeck (Eds.), *Open Source Jahrbuch 2004* (pp. 139.170). Berlin: Lehmanns Media

<sup>xxxix</sup> O'Mahony, S., Cela Diaz, F., & Mamas, E. (2005). *IBM and Eclipse*. Harvard: Harvard Business School

<sup>xl</sup> From <http://www.ratliff.net/blog/about/>

<sup>xli</sup> <http://gh-linux.blogspot.com/>

<sup>xlii</sup> <http://gforge.org/gf/>