Open Source Software System

Unit-I :
Essence of OSS Development Methodology

- definition: what is open source software?
- examples of open source software
- history of free software and open source
- open source business models
- open source software development model
- open source licensing models & beyond
  - copyleft and other legal means
- open source in general
  - what, other than software, might be open source?

what is open source software?

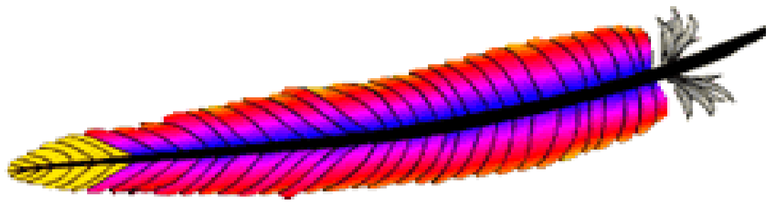
- Open Source software is distributed with its source code. The Open Source Definition has three essential features:
  - It allows free re-distribution of the software without royalties or licensing fees to the author
  - It requires that source code be distributed with the software or otherwise made available for no more than the cost of distribution
  - It allows anyone to modify the software or derive other software from it, and to redistribute the modified software under the same terms.

examples of open source software

- Operating Systems
  - Linux
  - FreeBSD, OpenBSD, and NetBSD: The BSDs are all based on the Berkeley Systems Distribution of Unix, developed at the University of California, Berkeley. Another BSD based open source project is Darwin, which is the base of Apple's Mac OS X.

examples of open source software

- Internet
  - Apache, which runs over 50% of the world's web servers.
  - BIND, the software that provides the DNS (domain name service) for the entire Internet.
  - sendmail, the most important and widely used email transport software on the Internet.
  - Mozilla, the open source redesign of the Netscape Browser
  - OpenSSL, is the standard for secure communication (strong encryption) over the



examples of open source software

- Programming Tools
  - Zope, and PHP, are popular engines behind the "live content" on the World Wide Web.
- Languages:
  - Perl
  - Python
  - Ruby
  - Tcl/Tk
- GNU compilers and tools
  - GCC
  - Make
  - Autoconf
  - Automake,etc..

open source software sites

- Free Software Foundation www.fsf.org
- Open Source Initiative www.opensource.org
- Freshmeat.net
- SourceForge.net
- OSDir.com
- developer.BerliOS.de
- Bioinformatics.org
- see also individual project sites; .org; etc.

History of Oss

- Richard Stallman, the GNU operating System, the Free Software Foundation, and the General Public License (GPL)
- Bill Joy, UNIX and the Berkeley Software Distribution License (BSD)
- Open source comes of age – Linux, Mozilla, Apache et al., and the corporate licenses
- The Open Source Initiative

History of Oss

- 1984 – The Free Software Foundation is Formed - Goal: to develop a free version of a UNIX-like OS, thuis was called the GNU project
- 1989 – FSF releases the GPL v1.0
- 1991 – the first code for Linux is released
- 1994 – Linux 1.0 is released
- 1994 – RedHat and Slackware versions released (C++, TCP/IP, Server)
- 1995 – Work starts on Apache
- 1996 – Work starts on KDE
- 1997 – “The Cathedral and the Bazaar” is published
- 1998 – the term “Open Source” is coined
- 2001 – Linux 2.4 is released

Main characteristics of F/OSS

- Free Software - Open Source Software: used interchangeably [F/OSS]
- Software development paradigm: collaborative and distributed development
- Licensing models: proliferation of “open source licenses (GPL, LGPL, BSD, etc..)
- F/OSS: potential benefits for interoperability and standards

ISSUES RELATING TO THE USE OF OSS

- Competition: adding competition to the market
- Lock-in: strong customer dependence on vendor
- Cost: total cost of ownership
- Reliability: software errors
- Maintenance: update cycle
- Sustainability: F/OSS as a lasting mechanism
- Capacity building: more S/W producers, small market segments better served

ISSUES RELATING TO THE USE OF OSS

- Innovation: large base of developers
- Product liability: a general problem; “good governance”
- Security and trust: unwanted functions; branding
- Education: educational tools
- Empowerment: customer has more decisive power and access to tools
- Equity: wider access to software

Software Development in Practice

- In the real world, software development is totally different and is more chaotic
  - Software professionals make mistakes
  - The client's requirements change while the software product is being developed
  - A software product is a model of the real world, and the real world is continually changing.

conventional models of software development

- waterfall
  - from requirements to code without a backward turn
  - historically used for large military and corporate software productions; originally used because computing time was expensive

- spiral
  - iterative cycles of requirements, development, testing, redrafting of requirements, etc.

conventional models of software development

- Rapid prototyping
  - is a working model that is functionally equivalent to a subset of the product.
  - The first step is to build a rapid prototype and let the client and future users interact and experiment with the rapid prototype.

open source software development

- Open Source” is a software development and distribution methodology
  - Source is provided
  - Source can be distributed
  - Source can be modified

- The basic idea behind open source is very simple: When programmers can read, redistribute, and modify the source code for a piece of software, the software evolves.

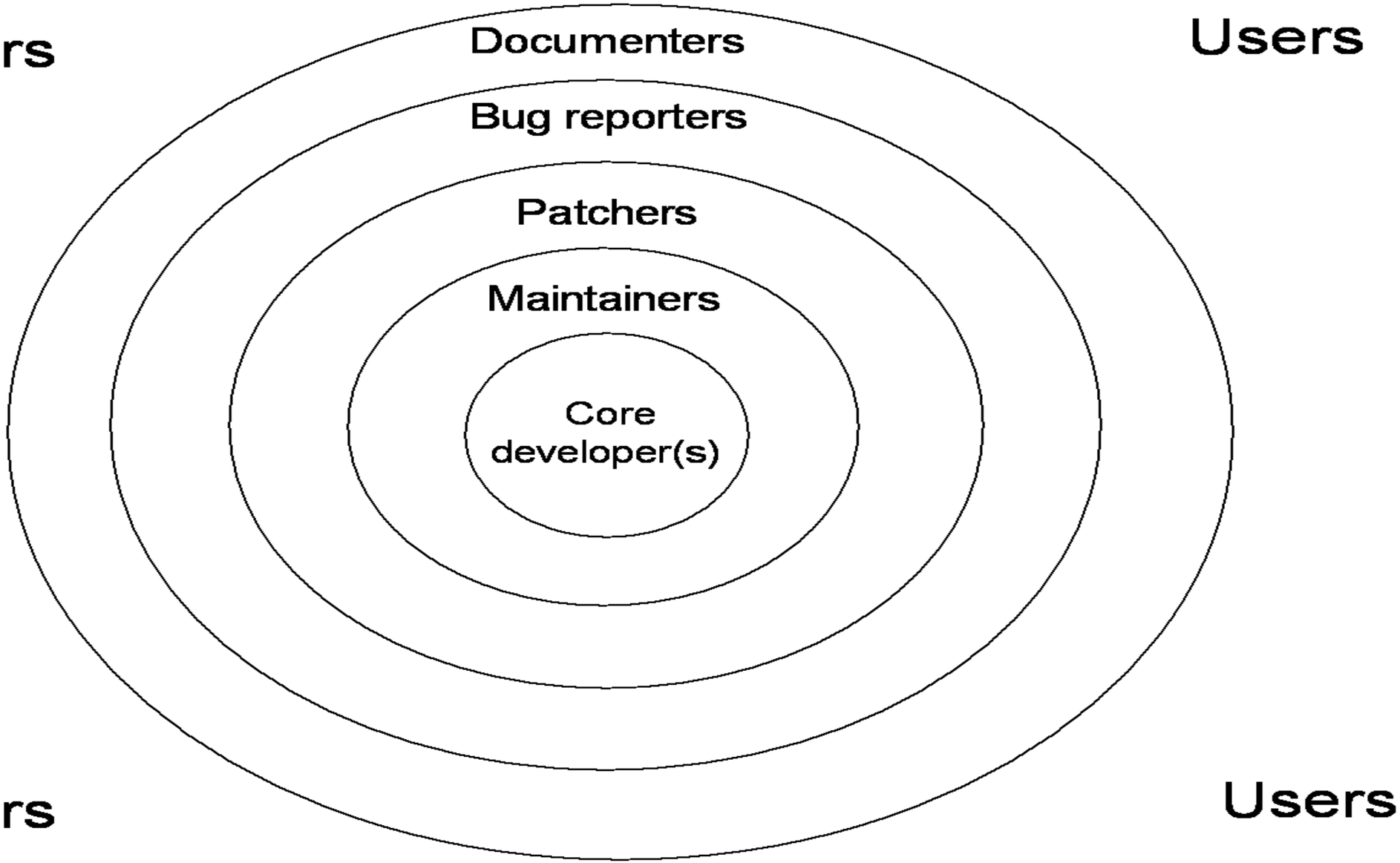
- People improve it, people adapt it, people fix bugs.

open source software development

- This can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing.

- Free Software - Open Source Software: used interchangeably [F/OSS]

open source software development



Open Source Vs. Closed Source Software

CSS

- Developed by Companies and developers work for economic purposes.
- Centralized, single site development
- Users may suggest requirements but they may or may not be implemented
- Release is not too often. There may be only yearly releases.

OSS

- Developed By Volunteers work for peer recognition. People know that recognition as a good developer have great advantage
- Decentralized, distributed, multi-site development
- User suggests additional features that often get implemented.
- Software is released on a daily or weekly basis

Open Source Vs. Closed Source Software

CSS

- Market believes commercial CSS is highly secure because it is developed by a group of professionals confined to one geographical area under a strict time schedule. But quite often this is not the case, hiding information does not make it secure, it only hides its weaknesses

- Security cannot be enhanced by modifying the source code

OSS

- OSSD is not market driven;

  it is quality driven. Community reaction to bug reports is much faster compared to CSSD which makes it easier to fix bugs and make the component highly secure

- The ability to modify the source code could be a great advantage if you want to deploy a highly secure system

open source companies

- IBM
  - uses and develops Apache and Linux; created Secure Mailer and created other software on AlphaWorks
- Apple
  - released core layers of Mac OS X Server as an open source BSD operating system called Darwin; open sourcing the QuickTime Streaming Server and the OpenPlay network gaming toolkit
- HP
  - uses and releases products running Linux

open source companies

- Sun
  - uses Linux; supports some open source development efforts(Forte IDE for Java and the Mozilla web browser)
- Red Hat Software
  - Linux vendor
- ActiveState
  - develops and sells professional tools for Perl, Python, and Tcl/tk developers.

Licensing

- A license is basically an agreement between the user and the developer on how that software can be acquired and used.

- “Open Source” has specific meaning, tied to licenses endorsed by the Open Source Initiative (OSI)

- software licensed to users with these freedoms:
  - to run the program for any purpose,
  - to study and modify the program, and
  - to freely redistribute copies of either the original or modified program (without royalties, etc.)

Licensing

- Grants permission to use a copyrighted work
- Can grant any or all of the rights associated with copyright
- Can impose other restrictions, such as type or place or usage, or duration of the license
- Does not transfer ownership of the copyright

- An open source licensor must give the licensee certain rights to be considered open source
- Basically, the licensee has the right to use, modify or distribute the software, and the right to access the source code.
- Open source software is software that is subject to an open source license

What are the OSI and the OSD?

- The Open Source Initiative (OSI) is the de facto standards body for open source software. It determines what open source means, and approves licenses as being open source

- The Open Source Definition (OSD) is a set of criteria that a license must conform to to be considered open source. The OSI maintains the definition and changes it from time to time.

Licensing

- The distribution terms of open-Source Software must comply with the following criteria:
  - Free Redistribution
  - Source code
  - Derived Works
  - Integrity of the Author's Source code
  - No Discrimination Against Persons or Groups
  - No Discrimination Against Fields of Endeavor
  - Distribution of License
  - License Must Not be Specific to a Product
  - License Must Not Restrict Other Software
  - License Must Be technology-Neutral

The distribution terms of open-Source Software

- 1. Free Redistribution. “The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources, The license shall not require royalty or other fee for such sale.”

The distribution terms of open-Source Software

- 2. Source Code. “The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.”

The distribution terms of open-Source Software

- 3. Derived Works. “The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.”

- 4. Integrity of the Author’s Source Code. “The license may restrict source-code from being distributed in modified form only if the license allows the distribution of ‘patch files’ with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.”

The distribution terms of open-Source Software

- 5. No Discrimination Against Persons or Groups. “The license must not discriminate against any person or group of persons.”

- 6. No Discrimination Against Fields of Endeavor. “The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used in genetic research”

The distribution terms of open-Source Software

- 7. Distribution of License. “The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.”

- 8. License Must Not Be Specific to a Product. “The rights attached to the program must not depend on the program’s being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program’s license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.”

The distribution terms of open-Source Software

- 9. License Must Not Contaminate Other Software. “The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.”

- 10. License Must Be Technology Neutral. “No provision of the license may be predicated on any individual technology or style of interface.”

Open Standards

- Availability
  - Open Standards are available for all to read and implement.
- Maximize End-User Choice
  - Open Standards create a fair, competitive market for implementations of the standard.
  - They do not lock customer into a particular vendor or group
- No Royalty
  - Open Standards are free for all to implement,with no royalty or fee.
  - Certification of compliance by the standards and the organization may involve a fee.

Open Standards

- No Discrimination
  - Open Standards and the organizations that administer them do not favor one implementer over another for any reason other than the technical standards compliance of vendor's implementation.
  - Certification organizations must provide a path for low and zero-cost implementations to be validated, but may also provide enhanced certification services.

- Extension or Subset
  - Implementations of Open Standards may be extended, or offered in subset form
  - However, certification organizations may decline to certify subset implementations, and may place requirements upon extensions.

Different licenses

- Different licenses exist for OSS. The following four rights are typically enshrined in all of the licenses:
  - The right to use the software freely. The user has the right to install and use the software on any and as many computers as he/she likes and use it for professional and/or private purposes.
  - The right to modify the software to suit his/her needs. The user has the right to change the software and extend its functionality, fix any bugs or combine it with other software applications.
  - The right of access to the source code. This is an important prerequisite in order to exercise the right to modify the software.
  - The right to redistribute the original or modified software. This can be done at a cost, normally to cover duplication and distribution costs.

Most Popular OSS Licenses

- The GNU “General Public License” (GPL)
  - No standard open source license, but GPL most widely used (roughly 85% of open source software);
  - Terms include:
    - User freedom to distribute and/or modify.
    - Requirement that original and modified source code be always made available to the world under the terms of the original license.
    - Must retain copyright notices and warranty disclaimers.
    - Does not include grant of patent licenses.

Most Popular OSS Licenses

- The Mozilla Public License
  - Developed by Netscape for the Mozilla browser
  - Terms include:
    - Very similar to the GPL but,
    - Can charge royalties for modified versions;
    - Can include source code within larger works licensed under different license types, thus license does not ‘infect’ all downstream projects;
    - Must retain copyright notices and warranty disclaimers;
    - May provide additional warranties to downstream users but may have to indemnify original developer for any claims arising as a result;
    - Includes grant patent licenses;

Most Popular OSS Licenses

- The IBM Public License
  - Terms include:
    - User freedom to distribute and/or modify;
    - No requirement for source code availability in downstream distribution;
    - The program can be distributed in executable form thus allowing downstream users to develop, sell, and install customized software packages without having to make all customizations available to the world;
    - Must retain all copyright notices and warranty disclaimers;
    - Includes grant of patent licenses.

Most Popular OSS Licenses

- Open Software License
  - Terms include:
    - User freedom to distribute and/or modify;
    - Viral license, source code is always made available to the world;
    - Must retain copyright notices and warranty disclaimers;
    - Requires indemnification for attorney's fees incurred as a result of potential claims or litigation.

Most Popular OSS Licenses

- The Apache Software License
  - Governs the Apache web-server software.
  - Terms include:
    - User freedom to distribute and/or modify;
    - No requirement for source code to be made available to the world in downstream distribution;
    - Must retain all copyright notices and warranty disclaimers.

- The FreeBSD License
  - Unrestrictive license:
  - Only requires preservation of copyright notices and warranty disclaimers.

Community

- What is a community and why do open source projects want to build them?
  - Communities are simply groups of individuals sharing common interests.
  - Both closed and open source projects have communities of users, most of whom will be relatively passive in terms of their interactions with other community members.
  - On the other hand, either type of community may have members who decide to take on more active roles

Community

- To participating in an open source software community
  - Prepare
  - Get to know your community
  - Engage and give back

Prepare

- Play to your strengths
  - Not everyone has the same skills, and there are many roles outside writing code in open source projects. These include:
    - documentation writers
    - translators
    - website designers
    - GUI designers, and
    - communications people
  - Projects also need people to maintain the build and test machines, people to support new users, and beta-testers. See our separate document, Roles in open source projects, for more details.

Prepare

- Estimate your time commitment
  - Decide whether you want (and are able) to commit a small amount of time every day or a more substantial amount of time periodically.

- Check your employment contract
  - If you intend to work on a project as part of your day job, it is vital to check your employment contract for intellectual property rights (IPR) clauses to ensure that you are permitted to participate in an open source project.

Get to know your community

- Understand how the community communicates
  - Most communities have an accepted way of engaging with the community. This can be as simple as joining a developers' email list, and/or adopting a particular code of practice. Most important, perhaps, is to learn how questions are asked and answered in this particular development community.
- Understand how the community is governed
  - Some communities, for example, that of the Linux kernel, are hierarchies with clear chains of command
  - Understanding how decisions are made and conflicts resolved will help you understand how to best engage with the community.

Get to know your community

- Understand the role of constructive critics
  - The culture in a particular open source community may vary from that in other communities.
  - Just as different organizations operate with different cultures, there are differences in how these communities communicate. Discussions can be lively, in which a range of individuals frankly express their opinions.
  - The end goal is to further the development of the software and community as well as possible.
  - Criticism is in general always directed towards that goal, and not to the person as an individual.

Get to know your community

- Get to know the people
  - Large open source projects have a wide range of participants and the first or loudest answer to a query is not always the correct answer.
  - Not all open source communities are wholly virtual. Many of the big projects have face-to-face get together to discuss future plans, or run code fests, and these events can be used to establish personal relationships which will aid online discussion.

- Learn about 'Poisonous People’
  - Some behaviors in open source communities are seen as anti-patterns1 in that they obstruct constructive activity. Most people will

Get to know your community

- Understand the communications channels
  - Open source projects typically use chat sessions (typically IRC or jabber), mailing lists, websites, blogs, wikis, and version control repositories as their primary means of communication. Get to know your project's communication style and preferred tools.

Engage and give back

- Communicate what you are working on
  - If you work completely on your own to re-develop part of a project, by the time you finish, someone will almost certainly have either duplicated your effort or the project may have made other changes that render your work obsolete.

- Acknowledge resources you use and their creators
  - It is important to acknowledge the resources you use.

Engage and give back

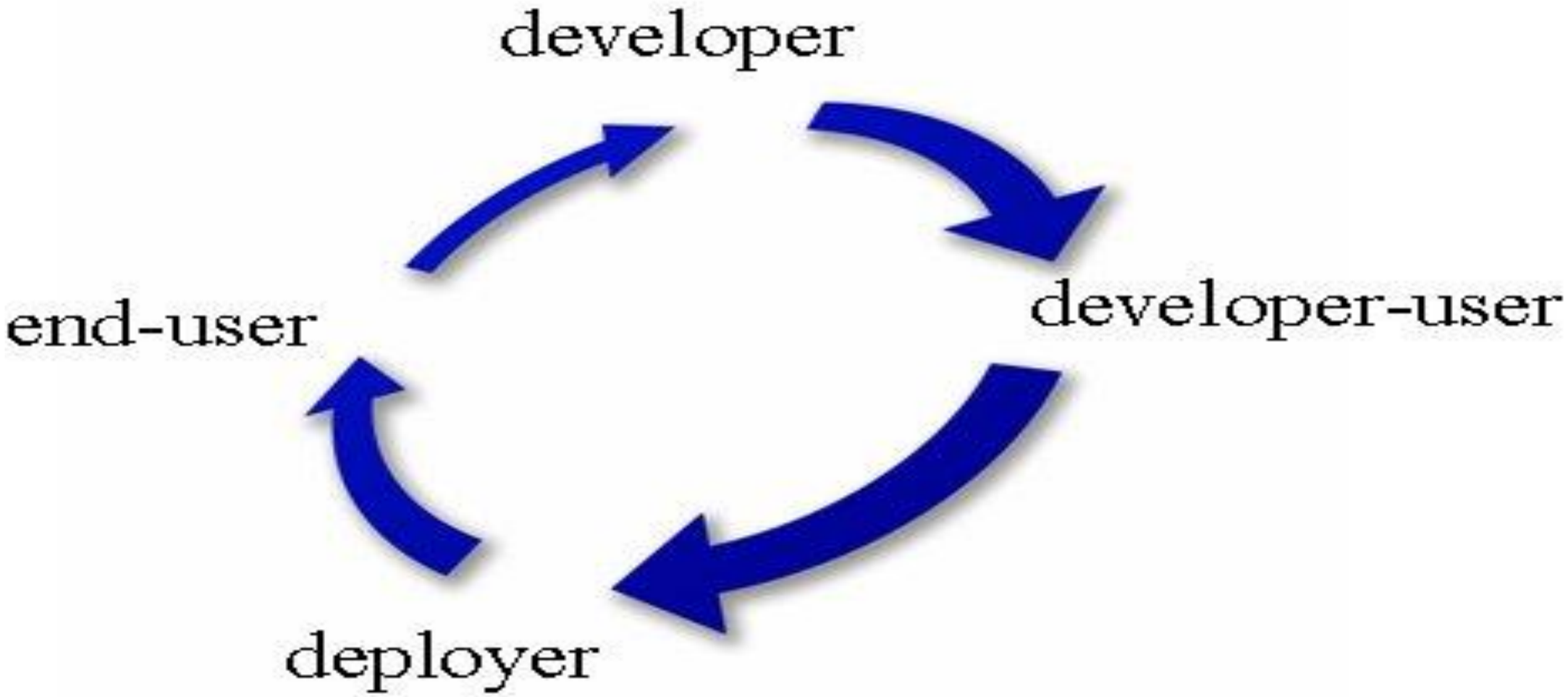
- Give back
  - A healthy community depends on the principle of individuals giving back to that community.
- Plan an exit strategy
  - Have a plan for what happens to your contributions to the open source project when your commitment to the project changes.
- Retire gracefully
  - if your open source participation is waning due to work, family or other commitments, inform other community members of the problem, so that they can take up the slack and/or take steps to lower your workload.

Motivation to Participate

- Understanding what drives open source developers to participate in open source projects is crucial for assessing the impact of open source software.
- what drives open source developers to participate in open source projects is crucial for assessing the impact of open source software.
- The two broad types of motivations that account for their participation in open source projects.
  - The first category includes internal factors such as intrinsic motivation and altruism,
  - the second category focuses on external rewards such as expected future returns and personal needs.

Open Source Development Methodology

- Virtual community of programmers, leveraging the Internet for communication, who create / debug / maintain / evolve a source code base
- OSS projects often self-organizing:
  - Someone determines a need and communicates that need to others on the Internet
  - If the project generates interest, one or more programmers begin writing code
  - Someone takes a leadership role and begins to map out a project road map
  - Interested programmers join the project to contribute new code or fine-tune existing code
  - A network of participants, linked via the Internet, forms
  - Tiered participation levels emerge



OSS Policies and Guidelines

- Purpose:
  - The purpose of the policy is to encourage departments and agencies to consider the use of open source software and proprietary software that incorporates or utilizes open standards when making decisions about procurement of software solutions.
  - A consistent set of policies and guidelines is needed to govern FOSS within an organization.

General guidelines

A few general guidelines that should be considered and addressed when developing a FOSS policy include:

- Determine if the use of FOSS is beneficial.
  - This can be examined by defining the business need, the role it plays in infrastructure and development projects, and cost saving trade-offs.
- Develop a process for FOSS governance.
  - This includes policies, procedures, and tools to manage the acquisition, use, licensing, deployment, and distribution of FOSS.
  - Within the context of this process, employee and management responsibilities should be defined, specifying their roles and responsibilities in support of the FOSS governance strategy.

General guidelines

- Define the extent to which an employee can contribute to open source.
  - Provide specifics for how, when, and where an employee should bring FOSS and its related projects into the organization and how much they should participate in the related community.

General guidelines

- Determine relationships with the open source community.
  - Consideration must be given to how your organization works with the open source community, in general and in relation to specific FOSS projects.
  - Develop best practices for managing the relationship. This should include organizational policies for licensing FOSS projects and protecting proprietary assets.
- Develop a documentation plan in support of communication and awareness of the organization's FOSS governance strategy.
  - In addition to traditional documentation, this may include training, internal public relations campaigns, and other educational opportunities.

Components of FOSS Policies

- There are many things FOSS policies should address, and questions that need to be answered.
- The policies needs to go beyond just understanding the license requirements and answer some of the following questions:
  - How is FOSS chosen?
  - How is FOSS acquired?
  - How and where is FOSS used?
  - How is FOSS supported?
  - How is FOSS tracked and how are FOSS projects tracked?

Determining Potential Business Use Cases for FOSS

- FOSS can be advantageous for an organization and can have cost-savings benefits.
- The following is a list of use cases to consider when determining whether your organization should contribute to the FOSS community.
  - Establishing your FOSS implementation as an industry standard.
  - Increasing sales of other products including hardware and software.
  - Distributing the expense of FOSS maintenance among other collaborators.
  - Gaining cooperation from the open source community.
  - Providing a strategy for a product’s end-of-life plan.
  - Enhancing an organization's image in the marketplace.

FOSS Management Issues

- FOSS management issues and recommendations for effective management strategies include:
  - developing a FOSS program office;
  - creating an open source review board;
  - maintaining a FOSS database;
  - and tracking compliance with FOSS licenses.
- The specific steps required to manage FOSS in an organization differ from management strategies used for traditional software governance.
- FOSS and associated licensing require a management program that is designed to handle the complexities inherent in the use of open source

Open source management strategies

- The following is a list of key open source management strategies that can be used to address and mitigate the potential risks of using open source:
  - Create a set of policies that regulate how an organization deals with FOSS.
  - Establish a clear set of procedures that an employee must adhere to in order to acquire FOSS.
  - Establish a formal FOSS governance process with governing authorities such as an open source review board (OSRB).
  - Compile and track inventory of all FOSS and related projects including: deployment, code use and reuse, community contributions, and shipments of products containing FOSS.

Open source management strategies

- Establish an internal open source community to provide organizational guidance and leadership and to manage utilization and propagation of FOSS technology.
- Develop FOSS legal expertise within your organization.
- Define and communicate to all employees the organization's FOSS policies and guidelines through cross-organizational training and awareness programs.
- Establish open source solutions that are approved by management.
- Leverage the experience of other organizations,.

Unit-II : OSS Development Phases

OSS Development Phases

- Software development
  - Software development is a complex process involving many phases from inception to deployment of the product and the success of the product is mainly dependant on the maturity of the process used to develop it.
  - OSSD differs in many aspects from the traditional closed source development in a company and it has changed the view of software development from an in-house development to global collaborative development.
  - The OSSD model delivers highly successful products as Linux, Apache, etc. as mentioned before and now many companies are trying to incorporate open source practices into their in-house development .

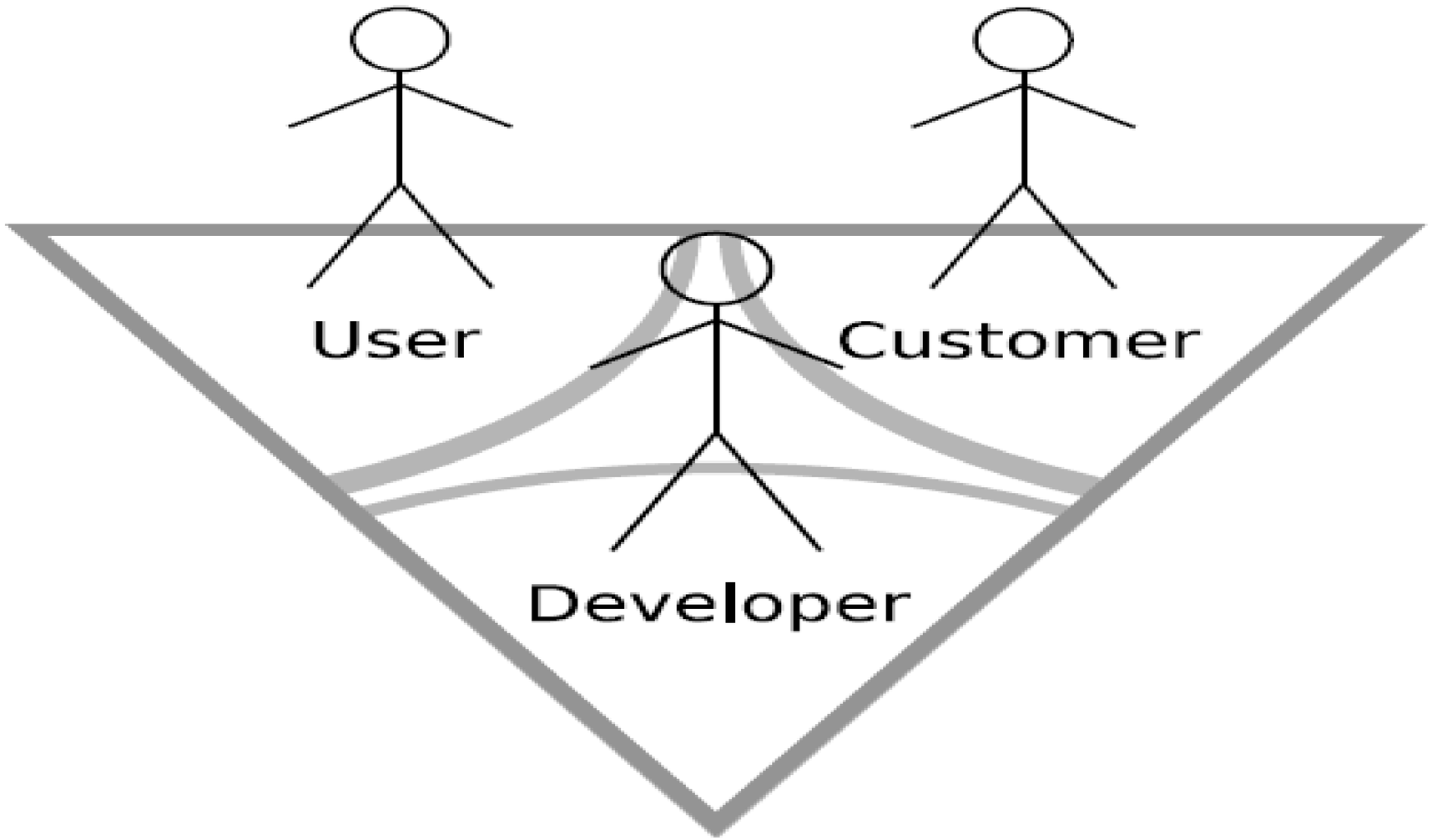
OSS Development Phases

- In closed source development in a company, there are defined teams with assigned roles and tasks, however, in OSSD the team structure is quite diverse.
- In contrast to closed source, OSSD has large community of developers, distributed worldwide participating freely and work in parallel collaboratively, thus the development process is iterative and is driven by large number of users community as well.
- The relationship between users and developers is very different in these two forms of development.
- Typically, OSSD enjoys effective users testing as the product is used by large number of users which are more intimately involved.
- Companies doing in-house development also takes care of its own interests as well, but in open source, users (many of them are experts) voluntarily interact and helps in development process

Roles- Roles in Commercial Projects

- Commercial project at least two groups of stakeholders can be identified.
  - The customer financing the project and users having to work with the product.
  - As a third role the service provider or short the developer can be identified.
- Between each of these groups exist clashing interests.
- Here some examples:
  - Users wish for comfortable functionality, but the customer wants to keep expenses as low as possible.
  - The customer wants to change business processes but the users are well used to old processes.
  - Users or Customers have “crazy wishes” conflicting with the functional understanding of the developers.

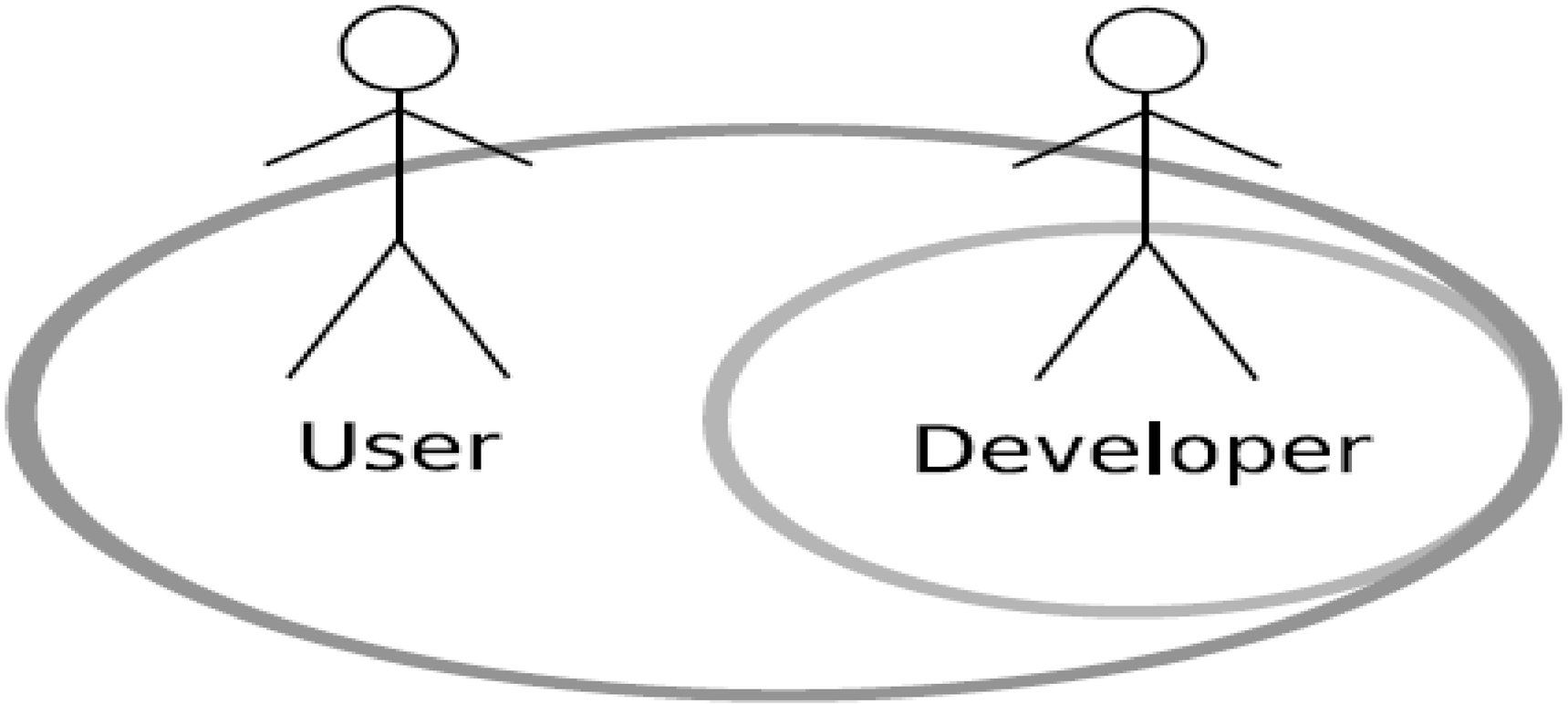
Roles- Roles in Commercial Projects



Roles- Roles in Open Source Projects

- In open-source development users and developers are present as well, but the customer role is eliminated and it is rather split up and distributed to the user and developer role.
- Some of the customer requirements are shift to the user requirements while the top level projects decisions are made by the core developer team.
- It is infeasible to see developers and users as separated groups.
- The developer is a user, this is where his personal interest for the project originated from in the first place.
- This results in a large intersection between the two groups and thus in far less clashing interests.

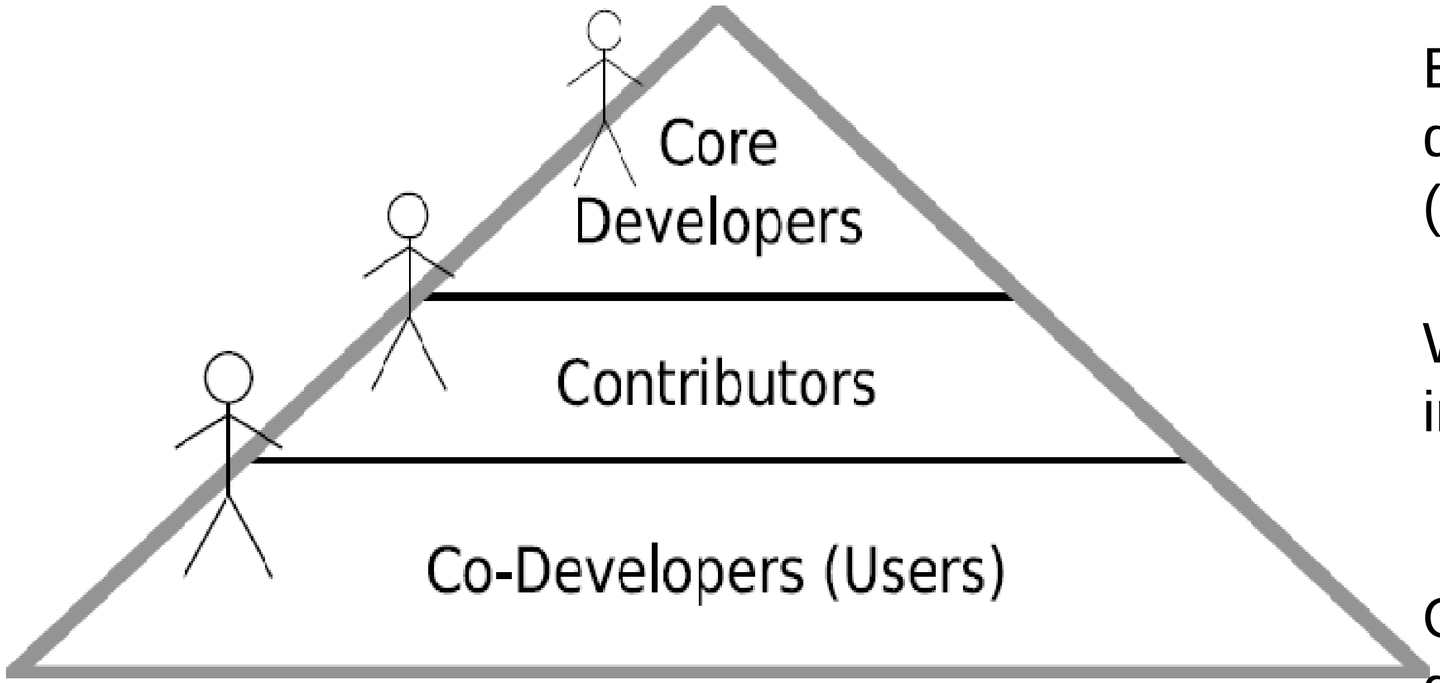
Roles- Roles in Open Source Projects



Roles- Roles in Open Source Projects

- it is worth taking a closer look on the developer role in open-source development.
- Here we usually find a flat hierarchy in order to coordinate the project.
- The developer role is usually split up into two levels of sub roles: core developers and contributors.
- The core developer teams includes the project leader(s) and often times its members are designated as being the only ones able to commit code-changes to the project’s code base.
- Major decisions are made within this team.

Roles- Roles in Open Source Projects



Evolution is almost entirely user-driven
(contributors use the software).

Workflow is iterative and incremental
("release early, release often")

Quality rules enforced by top tier of developers who have ultimate authority to
accept or reject changes.

Phases of Development -Starting OSS Project

- The first informal phase
- One individual builds an initial version and makes it available via the Internet (e.g., SourceForge.net)
- If there is sufficient interest in the project, the initial version is widely downloaded;
  - users become co-developers;
  - the product is extended.

- Key point: Individuals generally work voluntarily on an open-source project in their spare time

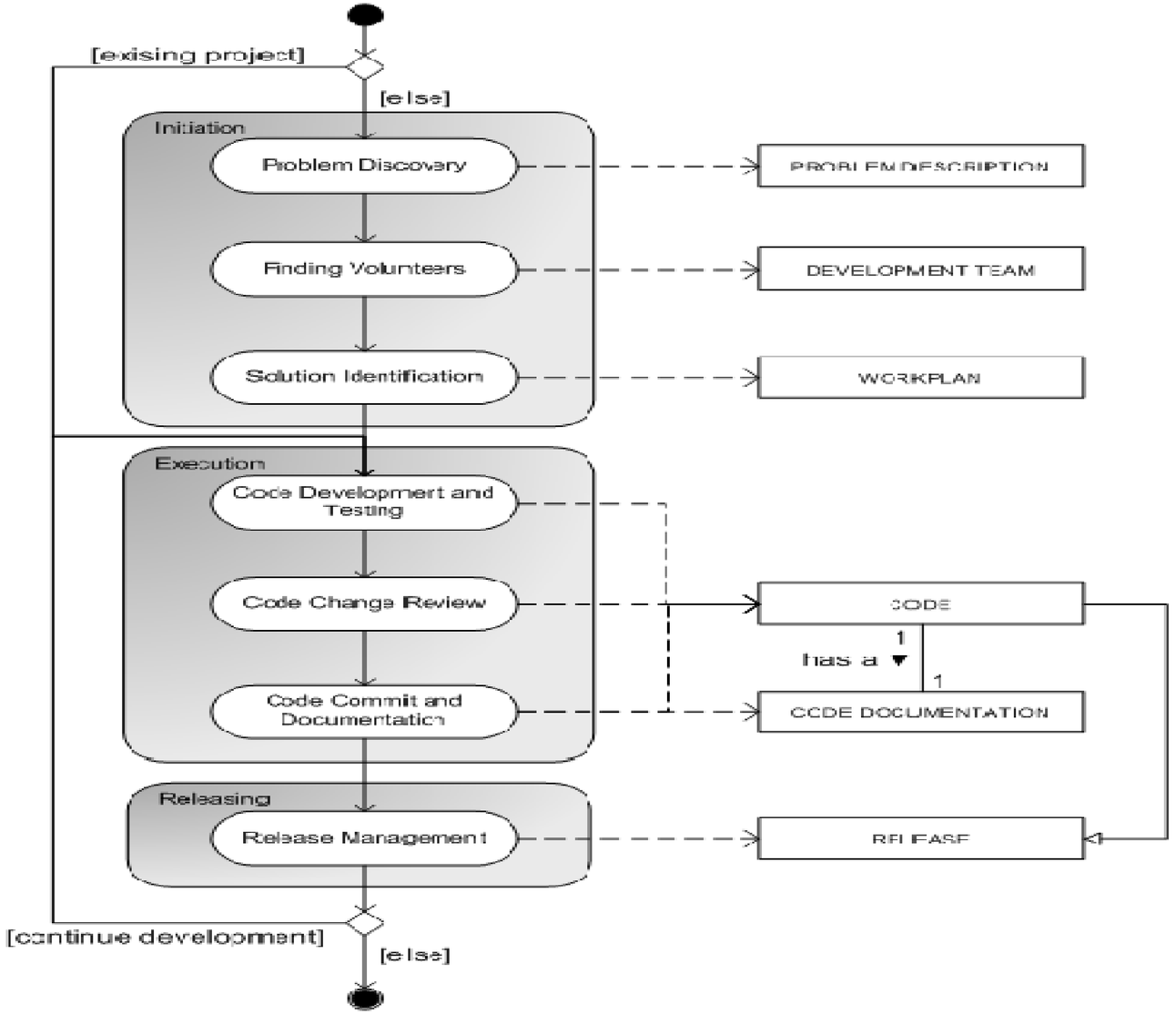
Phases of Development

- The second Informal Phase
  - Reporting and correcting defects
  - Corrective maintenance
  - Adding additional functionality
  - Perfective maintenance
  - Porting the program to a new environment
  - Adaptive maintenance

- The second informal phase consists solely of postdelivery maintenance.

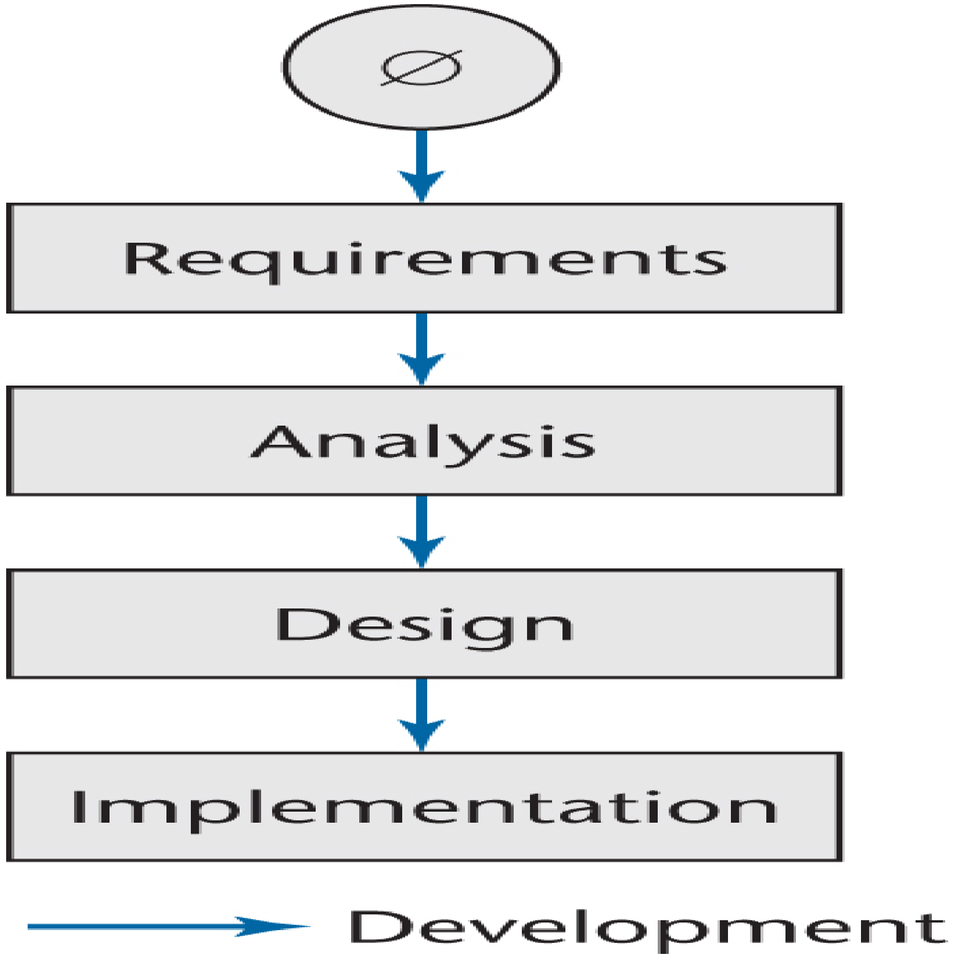
Phases of Development

- An initial working version is produced using the rapid-prototyping model, the code-and-fix model, and the open-source life-cycle model.
- The initial version of the rapid-prototyping model is then discarded. The initial versions of Code-and-fix model and open-source life-cycle model become the target product
- There are generally no specifications and no design. However, open-source software production has attracted some of the world's finest software experts. They can function effectively without specifications or designs

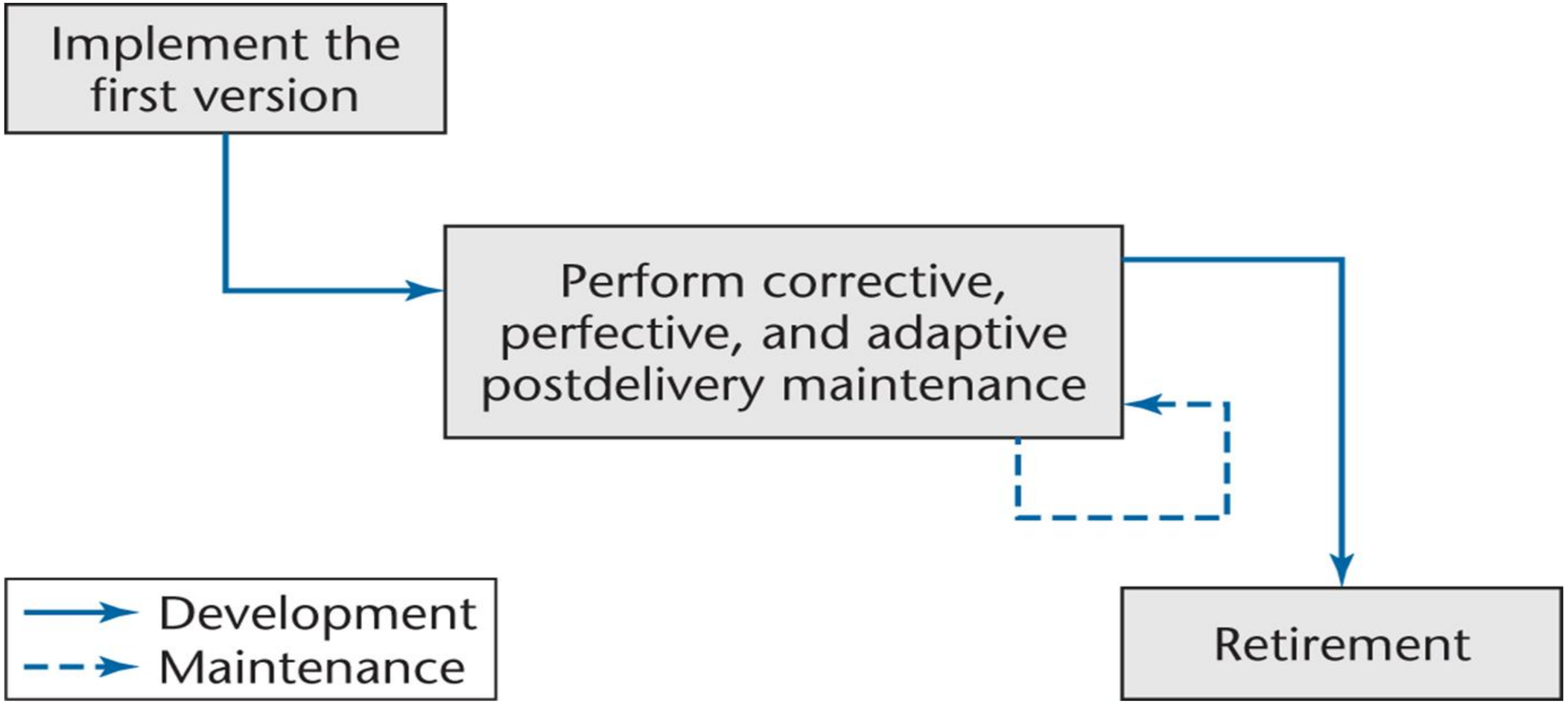


Software Development in Theory

- Ideally, software is developed as described in the last lecture
  - Linear
  - Starting from scratch



Starting-Adopting-Exit



Open-Source vs. Closed-Source

- **Closed-source software** is maintained and tested by employees
- Users can submit failure reports but never fault reports

- **Open-source software** is generally maintained by unpaid volunteers
- Users are strongly encouraged to submit defect reports, both failure reports and fault reports
- Core group: Small number of dedicated maintainers with the inclination, the time, and the necessary skills to submit fault reports (“fixes”); They take responsibility for managing the project; They have the authority to install fixes
- Peripheral group: Users who choose to submit defect reports from time to time

Open-Source vs. Closed-Source

- New versions of **closed-source software** are typically released roughly once a year
- After careful testing by the SQA group

- The core group releases a new version of an **open-source product** as soon as it is ready
- Perhaps a month or even a day after the previous version was released
- The core group performs minimal testing
- Extensive testing is performed by the members of the peripheral group in the course of utilizing the software

Types of Open Source Software

- There are 3 broad types of Open Source software:
  - Community, Vendor (or Commercial) and Hybrid.

- Community Based Open Source Software
  - Community based Open Source Software is software that is developed and managed by a community of people.
  - Often times, there are key developers in the community, but the community is open for anyone to join.
  - Examples of community-based Open Source Software include: Apache, Netscape and GNU Compiler Collection.

Types of Open Source Software

- Vendor Based Open Source Software

  - Vendor based, or Commercial, Open Source Software is software that is primarily developed and released by a company.
  - Typically, the vendor makes a version of the product available in a community edition that is freely downloadable.
  - The community edition does not come with support and typically has less functionality than their commercial version

    (often called a Professional or Enterprise version).
  - Examples of this type of Open Source software include Talend, SugarCRM, Jaspersoft, Infobright and BonitaSoft.

Types of Open Source Software

- Hybrid Open Source Software
  - The Hybrid model of Open Source software often evolves from a community based system.
  - This occurs when a company forms around the open source project. The company typically sells pre-packaged releases of the software and offers training, support and other services.
  - In addition, the company often develops and offers proprietary add-ons or customizations to the to Open Source product.
  - Examples of this are Linux (sold by Red Hat, Canonical, and many others) and postgreSQL (sold by EnterpriseDB).

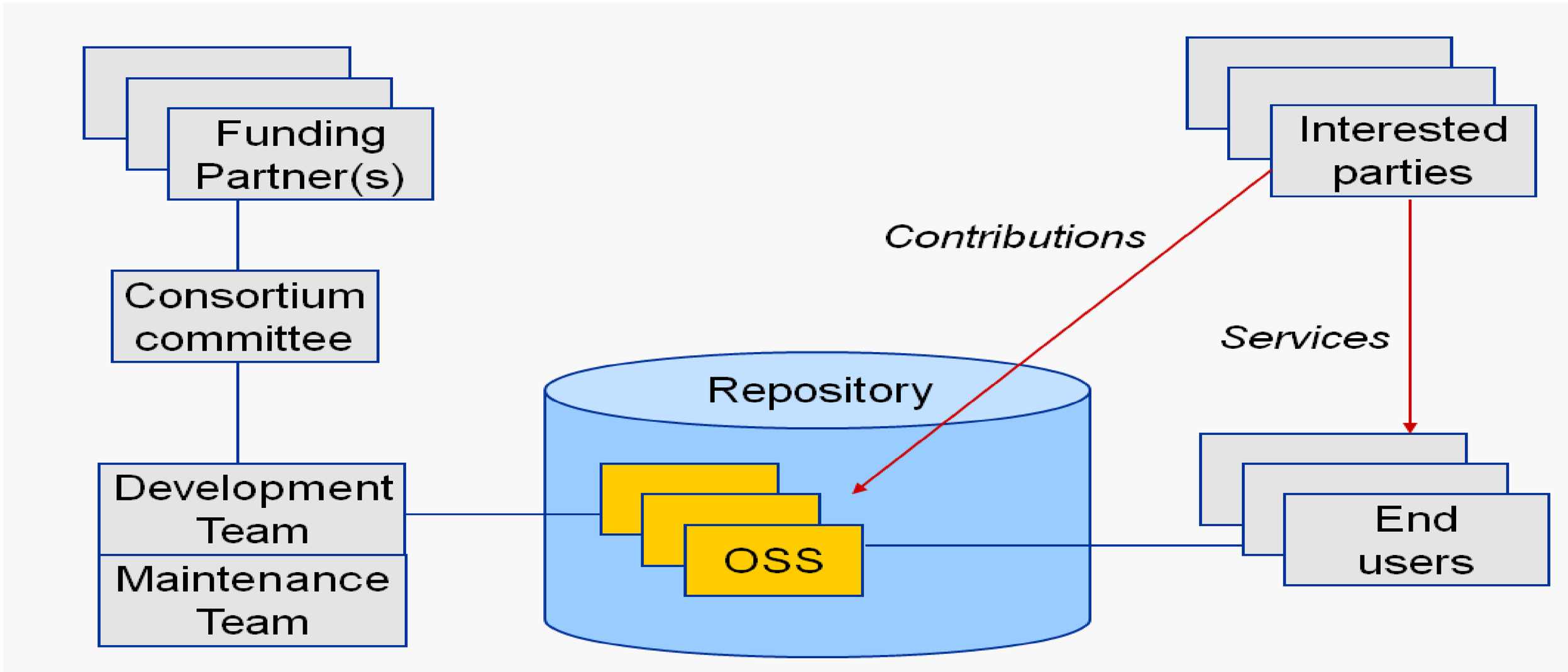
Starting an Open Source Project

- Decide upon software development strategy and cooperation
- Establish organisation
- Decide upon type of F/OSS Licence
- Involvement of commercial companies
- End user administration
- Marketing
- Funding

Open Source Community - responsibility

- Establish ownership/ partnership, sufficiently anchored within the participating organisations
- Establish organisation
- Development team (roles)
- Maintenance team (roles)
- Ownership and commitment
- Repository (Eurostat OSOR…)

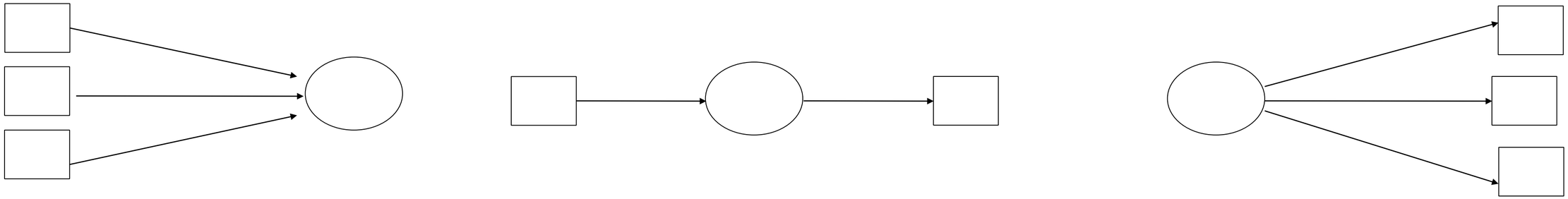
Open Source Community - responsibility

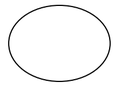


Co-ordination:

- defined as managing dependencies

Shared Input Resource Shared Output Resource Producer Consumer





Task

Resource

Task using or creating a resource

Co-ordination mechanisms

- Self-assignment
- Ask someone to do something
- Ask an unspecified person

Decision Making

- Decision-making practices emerge from the interactions of the team members rather than from organizational context.

  -Decision Trigger Type
  -Decision Process Complexity
  -Decision Announcement
  -Decision Type

Leadership

- Micro-level analysis of OSSD governance
- Accounting for Individual Actions and Resources
- Resources and Artifacts as Objects of Interaction

Modularity and division of labor

- An innovative manufacturing paradigm for the design and the production of complex artifacts and a key element in explaining the development and the success of many F/OSS projects.
- **Modularity in F/OSS development**

  -Imitating a previously existing design

  -Horizontal division of labor, task interdependencies and Brooks' Law

Development Process

- Universal, immediate access to project artifacts.
- Volunteer effort.
- Standards-based.
- Diversity of usage leads to plurality of authorship.
- Release early, release often.
- Peer review.

Development Process

- Despite the many differences from proprietary software, it is important to note that OSS usually goes through the same stages as a proprietary product.
- Some key differences are:
- In open source project development, this process may happen much more organically – starting with a single developer doing a relatively small project, then having the project involve more developers, and attract institutional support, as it develops.

- The pace of open source development can be slower,due to the voluntary nature of many development projects.

Development Process

- The quality of open source software can be much better than proprietary software, because programmers learn from each other, the additional "sets of eyeballs" viewing the code tend to catch potential bugs, and there is less commercial deadline pressure to rush the software out the door in an unfinished state.

- The version numbering of open source software tends to be more conservative.

Development Process

- The quality of open source software can be much better than proprietary software, because programmers learn from each other, the additional "sets of eyeballs" viewing the code tend to catch potential bugs, and there is less commercial deadline pressure to rush the software out the door in an unfinished state.

- The version numbering of open source software tends to be more conservative.

Unit-III: Tools Used for OSS Development

Oss Tools

- Minimum prerequisites for launching an OSS project:
  - Mailinglist
    - primary communication channel for the people working on the project
    - public discussion
  - Version control
    - Preferably Subversion these days but CVS remains popular too.
  - Bug tracking
    - Bugzilla or similarly capable system

Oss Tools

- Optional
  - Build, integration and testing facilities
  - Website
    - A place where you list the access points to the tools above + maybe documentation and marketing information
  - WIKI
    - A place for developers/users to collaboratively work on non development artifacts

Version Control

- A Version Control System (or revision control system) is a combination of technologies and practices for tracking and controlling changes to a project's files, in particular to source code, documentation, and web pages

- The reason version control is so universal is that it helps with virtually every aspect of running a project:
  - inter-developer communications,
  - release management,
  - bug management,
  - code stability and experimental development efforts,
  - and attribution and authorization of changes by particular developers

Version Controlling OSS

- A version control system maintains an organized set of all the versions of files that are made over time.
- Version control systems allow people to go back to previous revisions of individual files, and to compare any two revisions to view the changes between them.
- In this way, version control keeps a historically accurate and retrievable log of a file's revisions.

Version Controlling OSS

- More importantly, version control systems help several people (even in geographically disparate locations) work together on a development project over the Internet or private network by merging their changes into the same source repository which is the base of open source software.
- There are various version control tools available such as concurrent version system (cvs), subversion (svn), bazaar, Git

Version Control Basics

- There are some basic terms used in VCS:-

- **commit**: To make a change to the project; more formally, to store a change in the version control.
- **log message**: A bit of commentary attached to each commit, describing the nature and purpose of the commit.
- **update**: To ask that others' changes (commits) be incorporated into your local copy of the project; that is, to bring your copy "up-to-date".
- **repository**: A database in which changes are stored.

Version Control Basics

- **checkout**: The process of obtaining a copy of the project from a repository
- **working copy**: A developer's private directory tree containing the project's source code files, and possibly web pages or other documents.
- **revision, change, changeset**: A "revision" is usually one specific incarnation of a particular file or directory.
- **diff**: A textual representation of a change.
- **tag**: A label for a particular collection of files at specified revisions.

Version Control Basics

- **branch**: A copy of the project, under version control but isolated, so that changes made to the branch don't affect the rest of the project, and vice versa, except when changes are deliberately "merged" from one side to the other.
- **merge**: To move a change from one branch to another.
- **conflict**: All version control systems automatically detect conflicts, and notify at least one of the humans involved that their changes conflict with someone else's.
- **lock:** A way to declare an exclusive intent to change a particular file or directory.

Subversion:

- Subversion is a free/open source version control system (VCS).
- That is, Subversion manages files and directories, and the changes made to them, over time.
- This allows you to recover older versions of your data or examine the history of how your data changed.

Subversion:

- In this regard, many people think of a version control system as a sort of “time machine.”

- Subversion can operate across networks, which allows it to be used by people on different computers.

- At some level, the ability for various people to modify and manage the same set of data from their respective locations fosters collaboration

Bug Tracking Tools:

- Most large-scale projects require a bug tracking system (usually web or otherwise Internet based) to keep track of the status of various issues in the development of the project.

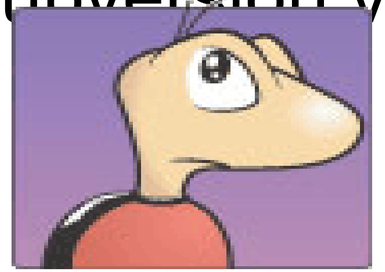
- A simple text file is not sufficient, because they have many such bugs, and because they wish to facilitate reporting and maintenance of bugs by users and secondary developers

Some popular bug trackers include:

- Bugzilla - a sophisticated web-based bug tracker from the Mozilla house.

- Mantis Bug Tracker - a web-based PHP/MySQL bug tracker.

- Trac - integrating a bug tracker with a wiki, and an interface to the Subversion version control system.



trac
Integrated SCM & Project Management

Some popular bug trackers include:

- Request tracker - written in Perl. Given as a default to CPAN modules - see rt.cpan.org.

- Fossil -written in C, and uses SQLite database. Apart from bug tracking, it also provides Wiki.

- EventNum -This was developed by the n in PHP.

Test Management Process

- Specify Requirement:

  Analyze your application and determine your testing requirements.

- Plan Tests:

  Create Test plan based on the Testing Requirements

- Execute Tests

  Create Test sets and perform test runs

- Track Defects

  Reporting the defects detected in your application and track how reports are progressed.

Test Management Process in OSS

- In proprietary software development, it is normal to have teams of people dedicated solely to quality assurance: bug hunting, performance and scalability testing, interface and documentation checking, etc.

- As a rule, these activities are not pursued as vigorously by the volunteer community on a free software project. This is partly because it's hard to get volunteer labor for testing.

- partly because people tend to assume that having a large user community gives the project good testing coverage.

Communication

- In OSS projects development teams are
  - Large
  - Geographically distributed
  - Crossing organizational boundaries
  - Composed of developers only
- Communication infrastructure consists primarily of
  - **Email (private and mailinglists)**
  - Newsgroups
  - Project website
  - Development tools (next theme)
- Face 2 face meetings not so common

Package

- In software, a package management system, also called package manager, is a collection of software tools to automate the process of installing, upgrading, configuring, and removing software packages for a computer's operating system in a consistent manner.

- Packages are distributions of software, applications and data.

- Packages also contain metadata, such as the software's name, description of its purpose, version number, vendor, checksum, and a list of dependencies necessary for the software to run properly.

Package

- Name and Layout
  - The name of the package should consist of the software's name plus the release number, plus the format suffixes appropriate for the archive type.
  - For example, Scanley 2.5.0, packaged for Unix using GNU Zip (gzip) compression, would look like this:
    - scanley-2.5.0.tar.gz
  - or for Windows using zip compression:
    - scanley-2.5.0.zip

Package

- The canonical form for distribution of free software is as source code. This is true regardless of whether the software normally runs in source form (i.e., can be interpreted, like Perl, Python, PHP, etc.) or needs to be compiled first (like C, C++, Java, etc.).
- With compiled software, most users will probably not compile the sources themselves, but will instead install from pre-built binary packages.

Package

- However, those binary packages are still derived from a master source distribution. The point of the source package is to unambiguously define the release.

- There is a fairly strict standard for how source releases should look. One will occasionally see deviations from this standard, but they are the exception, not the rule.

- Unless there is a compelling reason to do otherwise, your project should follow this standard too.

Format

- The source code should be shipped in the standard formats for transporting directory trees.
- For Unix and Unix-like operating systems, the convention is to use TAR format, compressed by compress, gzip, bzip or bzip2.
- For MS Windows, the standard method for distributing directory trees is zip format, which happens to do compression as well, so there is no need to compress the archive after creating it.

Package-TAR files

- TAR Files
  - TAR stands for "Tape ARchive", because tar format represents a directory tree as a linear data stream, which makes it ideal for saving directory trees to tape.
  - The same property also makes it the standard for distributing directory trees as a single file.
  - Producing compressed tar files (or tarballs) is pretty easy. On some systems, the tar command can produce a compressed archive itself; on others, a separate compression program is used.

Binary Packages

- Although the formal release is a source code package, most users will install from binary packages, either provided by their operating system's software distribution mechanism, or obtained manually from the project web site or from some third party.
- Here "binary" doesn't necessarily mean "compiled"; it just means any pre-configured form of the package that allows a user to install it on his computer without going through the usual source-based build and install procedures.
- On RedHat GNU/Linux, it is the RPM system; on Debian GNU/Linux, it is the APT (.deb) system; on MS Windows, it's usually .MSI files or self-installing .exe files.

Documentation

- Documentation is essential.
- There needs to be something for people to read, even if it's rudimentary and incomplete.
- Coming up with a mission statement and feature list, choosing a license, summarizing development status—these are all relatively small tasks, which can be definitively completed and usually need not be returned to once done.
- Documentation, on the other hand, is never really finished, which may be one reason people sometimes delay starting at al

Documentation

- One way to ensure basic initial documentation gets done is to limit its scope in advance. That way, writing it at least won't feel like an open-ended task.
- A good rule of thumb is that it should meet the following minimal criteria:
  - Tell the reader clearly how much technical expertise they're expected to have.
  - Describe clearly and thoroughly how to set up the software, and somewhere near the beginning of the documentation, tell the user how to run some sort of diagnostic test or simple command to confirm that they've set things up correctly.

Documentation

- One way to ensure basic initial documentation gets done is to limit its scope in advance. That way, writing it at least won't feel like an open-ended task.
- A good rule of thumb is that it should meet the following minimal criteria:
  - Tell the reader clearly how much technical expertise they're expected to have.
  - Describe clearly and thoroughly how to set up the software, and somewhere near the beginning of the documentation, tell the user how to run some sort of diagnostic test or simple command to confirm that they've set things up correctly.

Releasing

- Once the source tarball is produced from the stabilized release branch, the public part of the release process begins. But before the tarball is made available to the world at large, it should be tested and approved by some minimum number of developers, usually three or more.
- Approval is not simply a matter of inspecting the release for obvious flaws; ideally, the developers download the tarball, build and install it onto a clean system, run the test suite, Managing Volunteers, and do some manual testing.
- Assuming it passes these checks, as well as any other release checklist criteria the project may have, the developers then digitally sign the tarball using GnuPG (http://www.gnupg.org/), PGP (http://www.pgpi.org/), or some other program capable of producing PGP-compatible signatures.

Releasing

- Creating a release at an early stage of the project will ensure that, at a time when it is easy to do so.

- You build the tools and infrastructure for doing quick releases. If you leave this until later, the codebase will have grown in size and complexity. This makes it harder to set up the build process.

- While the source code is freely available and developed in an open manner, potential developers and users will generally want to be able to evaluate it quickly and easily.

- You should therefore release your project's software frequently, and in a form that is easily accessible and can be used by developers in any environment.

- By doing so, you will make it easier for people to contribute. If you tackle it early enough, you will find the creation of a semi-automated build-and-release system reasonably simple.

The reality of tools

- OSS projects are ***tool centric***
  - Tangible results of the project ***live inside the tools***
    - Source code in the SCM
    - Bug reports in the bug tracking system
    - Documentation in the WIKI
  - People communicate through tools + mail
    - All other forms of communication optional

The reality of tools

- Interesting concequences
  - Anything outside the tools is irrelevant.
    - software_architecture.ppt on somebody's laptop is not accessible to anyone and disconnected from the information in the SCM and bugtracking db. It might as well not exist and it is probably out of date/inacurate/obsolete/misleading/.... !
  - Tools are the ***only interface*** to the project
  - Tools are ***not compatible with many of the traditional waterfall*** model phases:
    - You won't find a requirements specification for the linux kernel
    - Nor is there a detailed design document for the Firefox browser
    - While there are some open source UML tools, they are rarely used in open source projects!

Unit- IV:
Common Development Methodologies

Development Methodologies

- Open source software development methodology in recent years is emerging as an alternative approach for the development of software projects despite that fact that no mature development methodology exists.
- There are many theoretical approaches that try to explain the phenomenon of open source development.

The Cathedral and the Bazaar

- Eric Steven Raymond
  - December 4, 1957
  - Fetchmail, gpsd, emacs editing modes
  - "The Cathedral and the Bazaar", published in 1997 .
  - Became a prominent voice in the open source movement
  - Co-founded the Open Source Initiative in 1998

Origins

- First version of the paper written in 1997.
- Several revisions appeared up to 2000.
- The author discovers a \development model" through the history of the Linux kernel and an own tool.
- This model is presented as revolutionary, since it is useful to build large systems without apparently any or few organization at all.

Models

- The Cathedral: The “classic” model.
  - Closed environment.
  - Small group of leaders/developers.
  - Only \stable" releases on, in some cases, \betas".
  - Used both in classic development models, such as waterfall,
  - spiral etc; and in classic OSS projects.
  - Examples: GCC, GNU Emacs.

Models

- The Bazaar: The model introduced by Linus Torvalds.
  - Open environment, almost any person can participate.
  - There are no clear leaders, undefined number of developers.
  - However, there is a benevolent-dictator figure.
  - “Release early, Release often”.
  - Examples: Linux.

Models

- The Bazaar style of development:
  - with a community seemed to resemble a large babbling bazaar of diverse agendas and approaches
  - with archive repositories where anyone can propose a modication
  - but from this, a stable and coherent large system emerges.

- This was surprising:
  - Why Linux world did not fly apart in confusion?
  - and why Linux seemed to go from strength to strength at a speed barely imaginable to cathedral-builders?

Eric Raymond principles

- Every good work of software starts by scratching a developer's personal itch.
  - Most successful free software projects have been started by developers with needs addressed by their \pet" project.
  - In the world of proprietary software, programmers spend their time building programs that they neither need nor want.
  - This motivation could explain the high quality of results given by Linux.

Eric Raymond principles

Good programmers know what to write. Great ones know what to rewrite (and reuse).

- Linus Torvalds did not try to write Linux from scratch.
- Instead, he started by reusing Minix code and ideas.
- Although today all reused Minix code has been removed or rewritten, while it was there, it provided scaffolding for the infant that would eventually become Linux.
- The source-sharing tradition of the Unix world has always been friendly to code reuse.

Eric Raymond principles

- If you have the right attitude, interesting problems will find you.
  - Eric's problem was that he needed a POP protocol client to work with.
  - And he found an abandoned one.
  - The problem was the continuation of the abandoned client, and Eric took it over and started to coordinate it.

Eric Raymond principles

- When you lose interest in a program, your last duty to it is to hand it off to a competent successor.
  - Before abandoning the development of a free software, we should find another person to continue its development.
  - Fortunately, in the bazaar world, some other hacker will find your abandoned work soon, and will start to develop it for his own needs.

Eric Raymond principles

- Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.
  - In Linux, many users are hackers too.
  - Because source code availability, these users can be e effective hackers.
  - This can be useful for shortening debugging time.
  - These users will diagnose problems, suggest fixes, and help in improvements.

Eric Raymond principles

- Release early. Release often. And listen to your customers.
  - This is another Linux characteristic: during very active development periods, lots of versions were released.
  - Sometimes, more than one in a day.
  - This maintains the hackers constantly stimulated and rewarded:
  - stimulated by the prospect of having an ego-satisfying piece of the action.
  - and rewarded by the sight of constant (even daily) improvement of their work.

Eric Raymond principles

- Given a large enough beta-tester and co-developer base,

  almost every problem will be characterized quickly and the fix

  obvious to someone.
  - “Given enough eyeballs, all bugs are shallow.” (named “Linus's Law" by Eric S. Raymond).
  - Somebody finds the problem, and somebody else understands (and fixes) it.
  - The differences between the cathedral and the bazaar:
  - In the cathedral-builder view of programming, bugs and development problems are tricky, insidious, deep phenomena.
  - In the bazaar, bugs are generally shallow phenomena (they turn shallow when exposed to a thousand eager co-developers collaborating in next release).

Eric Raymond principles

- Smart data structures and dumb code works a lot better than the other way around.
  - It is dificult to understand the code written by others,
  - but when we understand the data structures, understanding the code is easier.

Eric Raymond principles

- To solve an interesting problem, start by finding a problem that is interesting to you.

- Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

- the next best thing to having good ideas is recognizing good ideas from your users; sometimes the latter is better,

- perfection (in design) is achieved not when there is nothing to add, but rather when there is nothing more to take away.

Eric Raymond principles

- When the code is getting both better and simpler, that is when we know it is right.

- At this moment, the software maintained by Eric was, not only very different, but also simpler and better. It was time to change its name and give it its new identity: “fetchmail” instead of “popclient”.

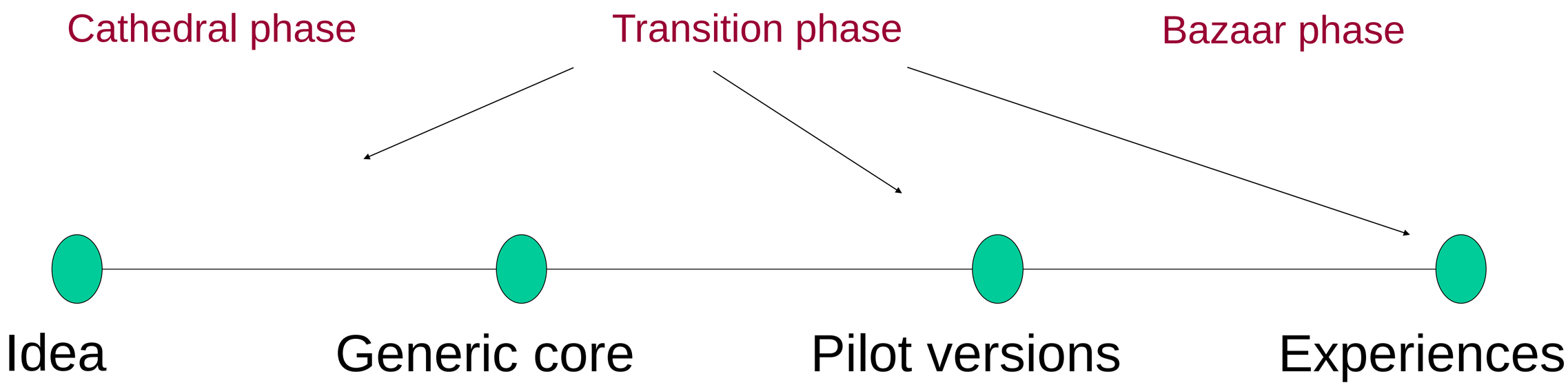
Migration from Cathedral to Bazaar

.

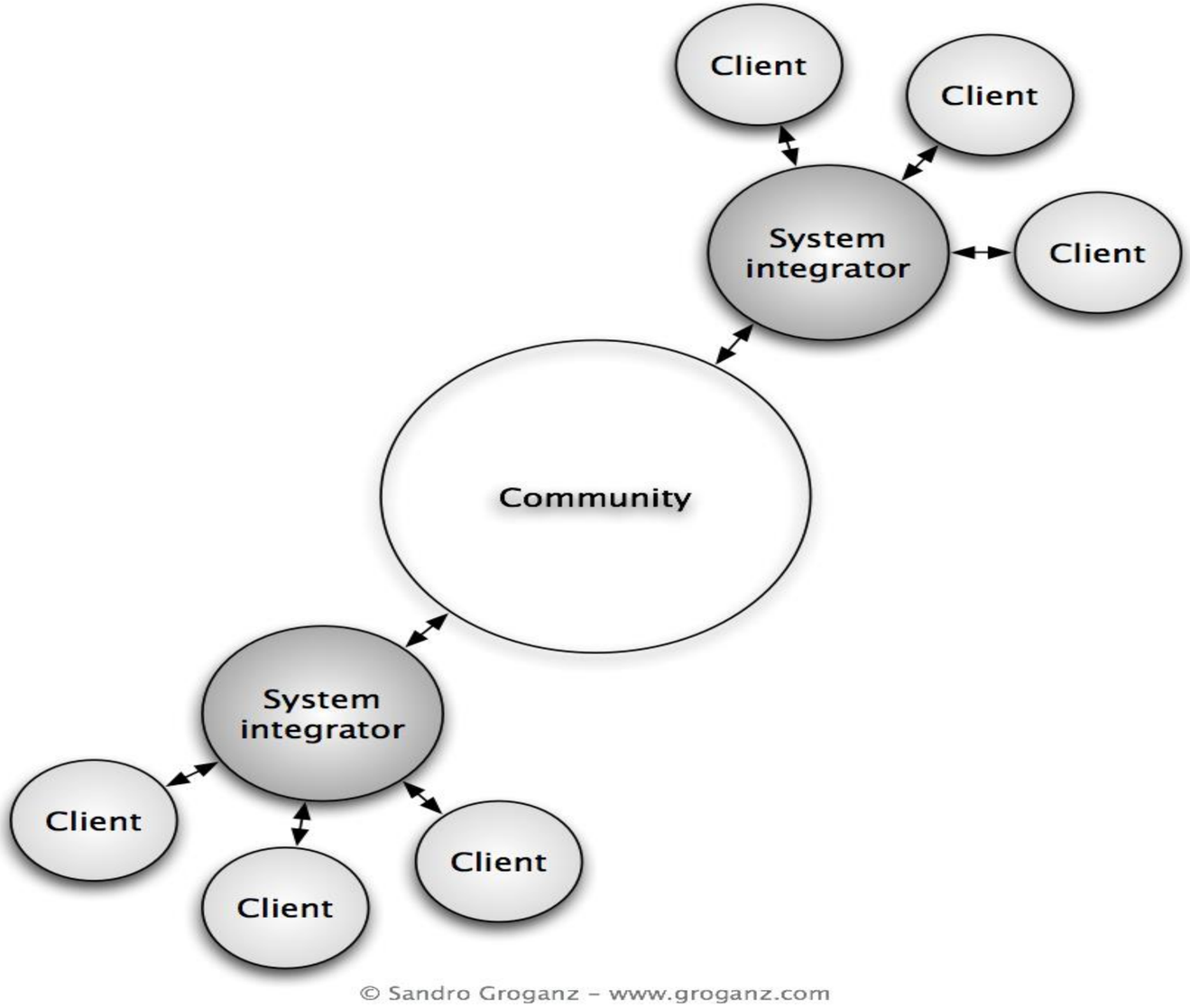
Project idea
Original stakeholders
Core developers

Stable modular version

Peer reviews
Code contributions
Community
Distributed development

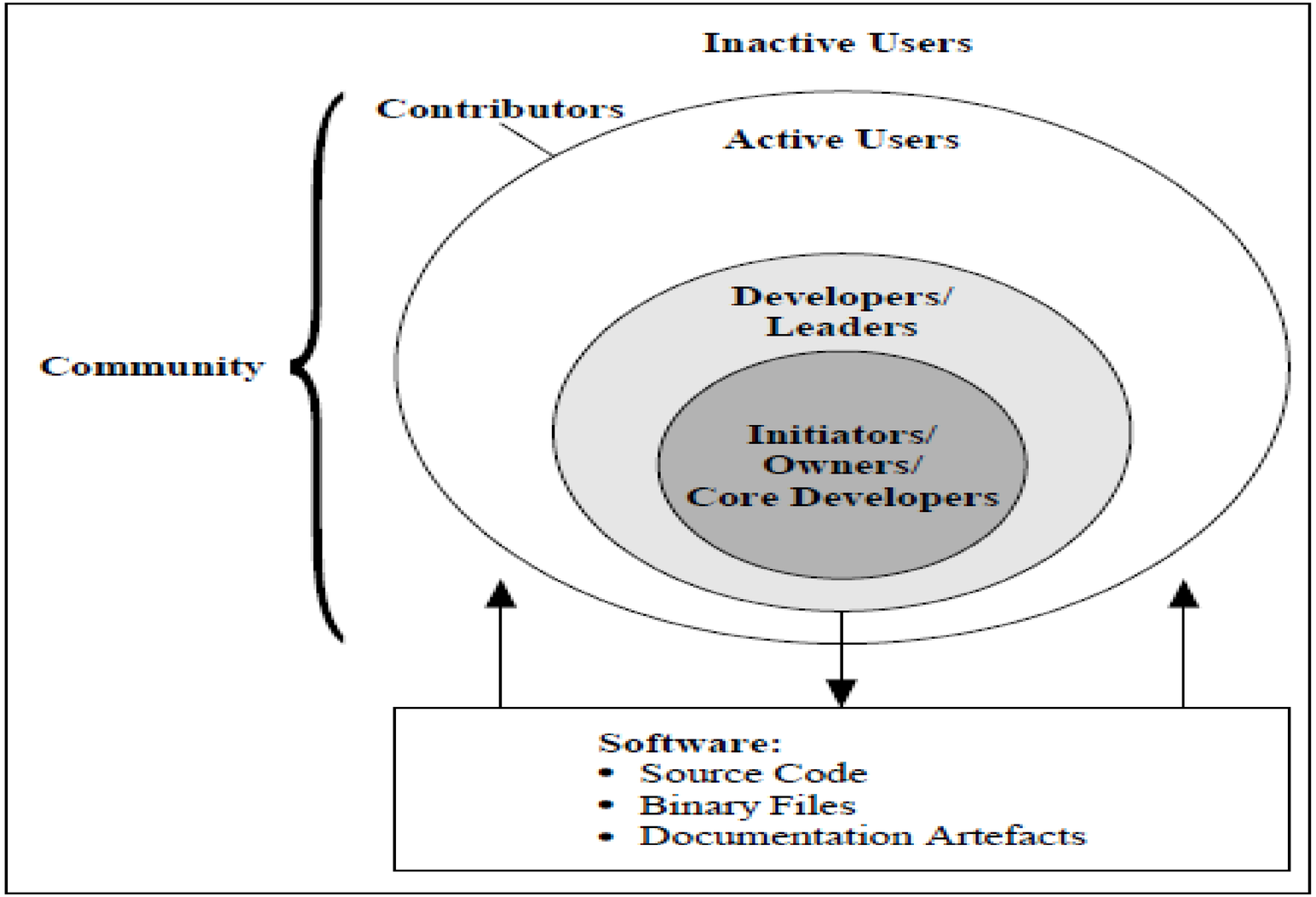


Community Ecosystem



Roles in Community driven ecosystem

- the contributor,
- the developer,
- the leader,
- the core developer,
- the project owner
- and the initiator



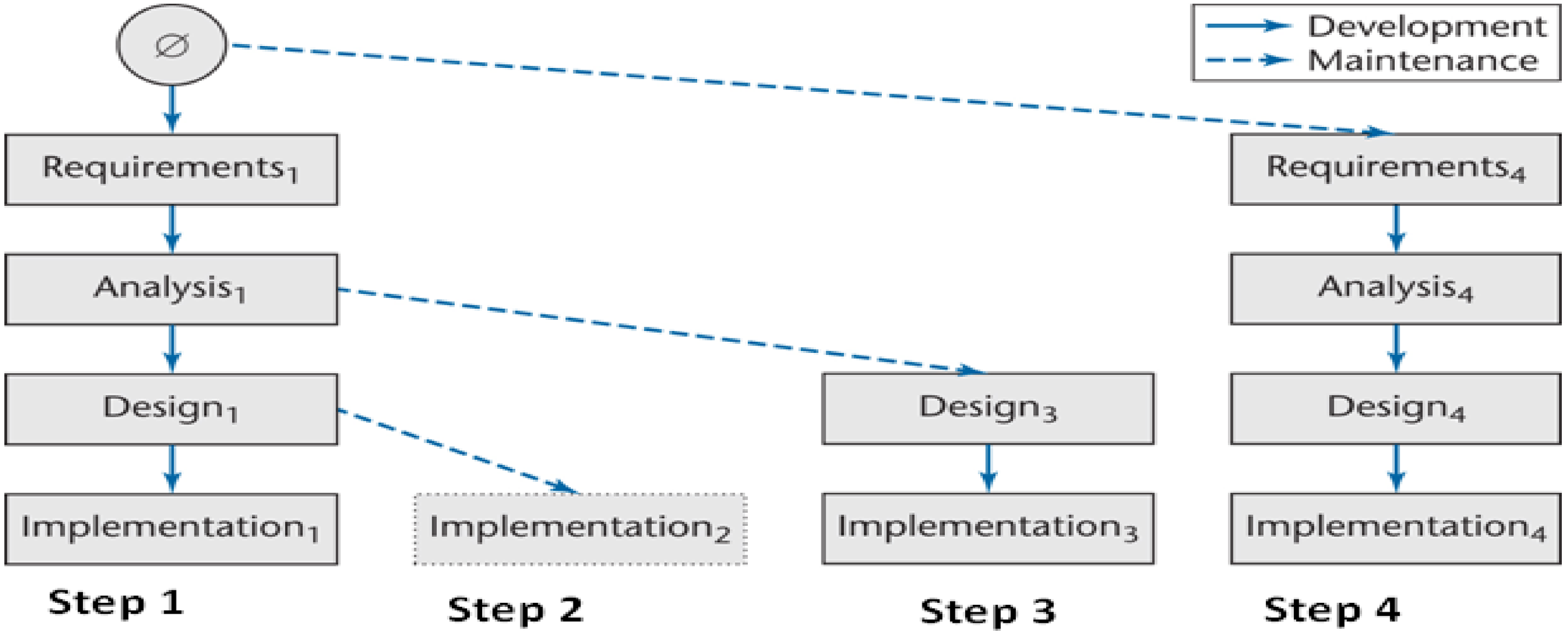
Openness

- Open to Join
- Open to the Choice of Work
- Open to Leave
- Open to Communicate
- Limits of Openness

Mini Case Study

- Step 1: The first version is implemented
- Step 2: A fault is found
  - The product is too slow because of an implementation fault
  - Changes to the implementation are begun
- Step 3: A new design is adopted
  - A faster algorithm is used
- Step 4: The requirements change
  - Accuracy has to be increased
- Epilogue: A few years later, these problems recur

Evolution Tree



Problems to be Avoided with any Model

- Moving Target Problem: A change in the requirements while the software product is being developed
- There is no solution to the moving target problem
- Must mitigate the negative impact to the max extent possible
- Feature Creep: a succession of small, almost trivial, additions to the requirements.

Problems to be Avoided with any Model

- Even if the reasons for the change are good, the software product can be adversely impacted
- Any change made to a software product can potentially cause a regression fault
- A fault in an unrelated part of the software
- If there are too many changes
- The entire product may have to be redesigned and reimplemented

- THE END