The Linux Development Platform

Configuring, Using, and Maintaining a Complete Programming Environment

ISBN 013009115-4

94999

9 780130 091154

SERIES PAGE INSERTS HERE

The Linux Development Platform

Configuring, Using, and Maintaining a Complete Programming Environment

Rafeeq Ur Rehman

Christopher Paul

Prentice Hall PTR
Upper Saddle River, New Jersey 07458
www.phptr.com

Library of Congress Cataloging-in-Publication Data

A CIP catalog record for this book can be obtained from the Library of Congress.

Editorial/production supervision: *Mary Sudul*
Cover design director: *Jerry Votta*
Cover design: *DesignSource*
Manufacturing manager: *Alexis Heydt-Long*
Acquisitions editor: *Jill Harry*
Editorial assistant: *Kate Wolf*
Marketing manager: *Dan DePasquale*

© 2003 Pearson Education, Inc.
Publishing as Prentice Hall PTR
Upper Saddle River, New Jersey 07458

This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale.
The publisher offers discounts on this book when ordered in bulk quantities. For more information, contact Corporate Sales Department, Phone: 800-382-3419; FAX: 201-236-7141;
E-mail: corpsales@prenhall.com
Or write: Prentice Hall PTR, Corporate Sales Dept., One Lake Street, Upper Saddle River, NJ 07458.

Other product or company names mentioned herein are the trademarks or registered trademarks of their respective owners.

All rights reserved. No part of this book may be
reproduced, in any form or by any means,
without permission in writing from the publisher.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-009115-4

Pearson Education LTD.
Pearson Education Australia PTY, Limited
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.
Pearson Education Canada, Ltd.
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education — Japan
Pearson Education Malaysia, Pte. Ltd.

To Asia, Afnan, and Faris for their love and support.

—Rafeeq Ur Rehman

To Cheryl, Rachel, and Sarah for the moral support and unending encouragement to complete this project. I'd be lost without you.

—Christopher Paul

CONTENTS

PREFACE

Setting up a complete development environment using open source tools has always been a challenging task. Although all of the development tools are available in the open source, no comprehensive development environment exists as of today. This book is an effort to enable the reader to set up and use open source to create such an environment. Each chapter of the book is dedicated to a particular component of the development environment.

Chapter 1 provides an introduction to the practical software development life cycle and stages. The chapter also provides information about the documentation required for all serious software development projects. Guidelines are provided about criteria for selecting hardware and software platforms.

Chapter 2 is about using editors. Editors are essential components of any software development system. Selection of a good editor saves time and money in the development life cycle. This chapter provides information about commonly used editors like Emacs, Jed and vim (vi Improved).

Chapter 3 is about the GNU set of compilers commonly known as GCC. The procedure for installation and use of gcc with different languages is presented here.

Larger software projects contain hundreds or thousands of files. Compiling these files in an orderly fashion and then building the final executable product is a challenging task. GNU `make` is a tool used to build a project by compiling and linking source code files. Chapter 4 provides information on how to install and use this important tool.

Chapter 5 discusses debuggers. An introduction to commonly used debuggers is provided in this chapter with an emphasis on the GNU debugger gdb.

Chapter 6 introduces CVS, which is the open source revision control system and is most widely used in open source development. Setting up a CVS server is detailed in this chapter. You will learn how to use remote the CVS server in a secure way.

There are tools other than compilers, debuggers and editors. These tools are discussed in Chapter 7. These tools help in building good products.

Open source tools are also widely used in embedded and cross-platform development. Chapter 8 provides information using open source tools in such environments. Remote debugging is an important concept and it is explained in this chapter.

Chapter 9 is the last chapter of the book and it provides a basic introduction to open source Java development.

There is one important thing that you must keep in mind while reading this book. It is not a tutorial on any language or programming techniques. It is about development tools and how to use these. You need other books to learn programming languages and techniques.

The book explains the installation procedures of different tools. By the time you read this book, new versions of these tools may be available. The installation procedures will not vary drastically in these versions and you can use the steps explained in this book. In fact, most of the open source tools employ the same compiling and installation procedure that you will note in this book. This process has been consistent and is expected to remain the same in future as well.

After reading this book, we are very much hopeful that the reader will be able to understand different components of a development system.You will also be able to create such a system from scratch using open source tools.

Rafeeq Ur Rehman
Christopher Paul

About the CD

This book comes with a CD-ROM. The CD-ROM contains source code of all software and utilities used in this book. You can compile and install these tools as explained in this book. If you need latest versions of these tools, you can download these from the links provided in the book.

ACKNOWLEDGMENTS

This is my third book and I have been very fortunate to get help from many people around me in all of these projects. Professor Shahid Bokhari at the University of Engineering and Technology Lahore, Pakistan, provided valuable suggestions while I was creating table of contents for this book. In fact he proposed a mini table of contents about what should be included in the book to make it useful both for the developers and students of computer science and engineering. I am grateful for his continued support.

Mike Schoenborn, Amgad Fahmy, and Greg Ratcliff at Peco II Inc. continuously encouraged me during the time I was writing the manuscript and I am thankful to all of them. I am also thankful to Victor Kean for providing his life experiences both in real social life and software development.

I am also thankful to Jill Harry and Mary Sudul at Prentice Hall PTR for bearing with me and pushing me to meet deadlines which really helped bring this book to the market in time.

Drew Streib did a great job in reviewing the manuscript and giving very useful suggestions to improve it. Thanks Drew.

Jim Shappell at Arcom Control Systems provided x86 based board for testing embedded development examples and remote debugging. Cole Creighton at Artesyn Communication Products provided PowerPC based board for cross-platform development testing purpose. I am thankful to both of them for their help in developing the manuscript of this book.

Bruce Parens gave valuable suggestions about what to include in the book. He also agreed to print the book under his Open Source Series. I was excited to get his approval and I am thankful to him.

And I am thankful to the open source community all over the world for putting such a huge effort to build these open source tools. This book exists only because of the open source products and tools.

Above all, I am thankful to my father, who taught me how to read and write and work hard.

Rafeeq Ur Rehman
September 25, 2002

CHAPTER 1

Introduction to Software Development

Software development is a complicated process. It requires careful planning and execution to meet the goals. Sometimes a developer must react quickly and aggressively to meet everchanging market demands. Maintaining software quality hinders fast-paced software development, as many testing cycles are necessary to ensure quality products.

This chapter provides an introduction to the software development process. As you will learn, there are many stages of any software development project. A commercial software product is usually derived from market demands. Sales and marketing people have first-hand knowledge of their customers' requirements. Based upon these market requirements, senior software developers create an architecture for the products along with functional and design specifications. Then the development process starts. After the initial development phase, software testing begins, and many times it is done in parallel with the development process. Documentation is also part of the development process because a product cannot be brought to market without manuals. Once development and testing are done, the software is released and the support cycle begins. This phase may include bug fixes and new releases.

After reading this chapter, you should understand how software development is done and the components of a software development system. At

the end of the chapter, you will find an introduction to Linux Standard Base. This chapter is not specific to a particular hardware platform or tools. You will start learning about components of an actual software development platform in the next chapter.

1.1 Life Cycle of a Software Development Project

Software development is a complicated process comprising many stages. Each stage requires a lot of paperwork and documentation in addition to the development and planning process. This is in contrast to the common thinking of newcomers to the software industry who believe that software development is just "writing code." Each software development project has to go through at least the following stages:

- Requirement gathering
- Writing functional specifications
- Creating architecture and design documents
- Implementation and coding
- Testing and quality assurance
- Software release
- Documentation
- Support and new features

Figure 1-1 shows a typical development process for a new product.

There may be many additional steps and stages depending upon the nature of the software product. You may have to go through multiple cycles during the testing phase as software testers find problems and bugs and developers fix them before a software product is officially released. Let us go into some detail of these stages.

1.1.1 Requirement Gathering

Requirement gathering is usually the first part of any software product. This stage starts when you are thinking about developing software. In this phase, you meet customers or prospective customers, analyzing market requirements and features that are in demand. You also find out if there is a real need in the market for the software product you are trying to develop.

In this stage, marketing and sales people or people who have direct contact with the customers do most of the work. These people talk to these customers and try to understand what they need. A comprehensive understanding of the customers' needs and writing down features of the proposed software product are the keys to success in this phase. This phase is actually a base for the whole development effort. If the base is not laid correctly, the product will not find a place in the market. If you develop a very good software product which is not required in the market, it does not matter how well you build it.You can find many stories about software products that failed in the market because the customers did not require them. The marketing people

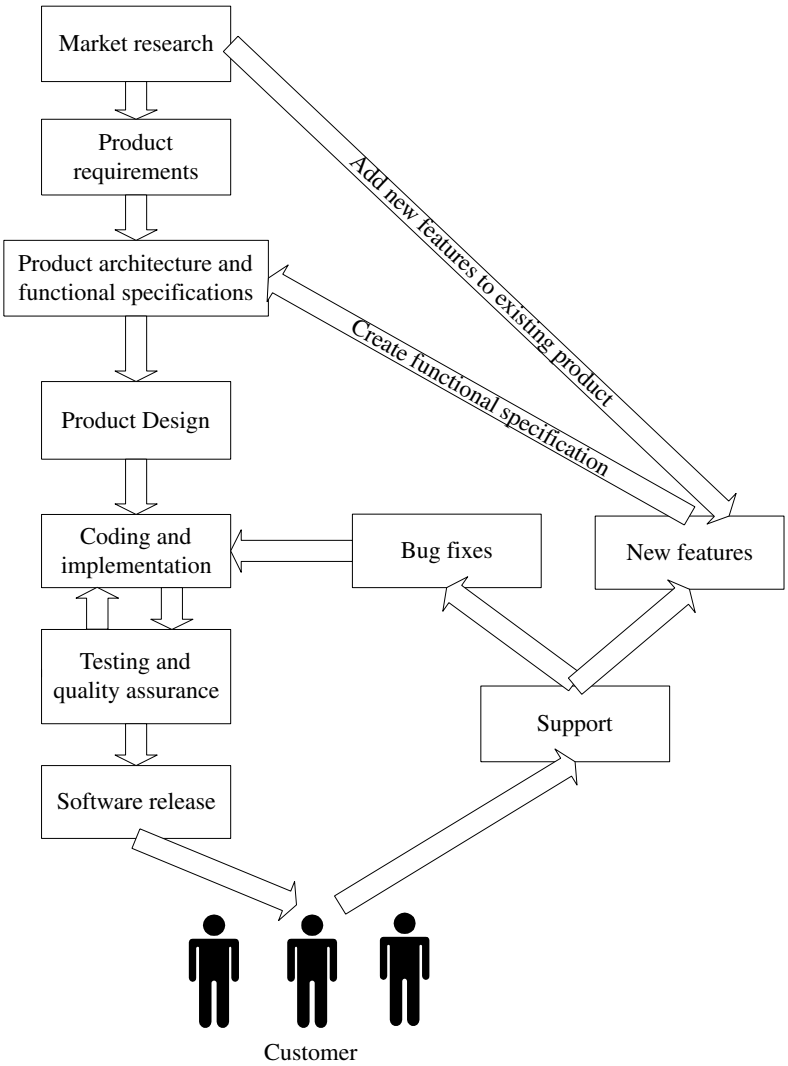

Figure 1-1 Typical processes for software development projects.

usually create a *Marketing Requirement Document* or MRD that contains formal data representation of market data gathered.

Spend some time doing market research and analysis. Consider your competitors' products (if any), a process called competitive analysis. List the features required by the product. You should also think about the economics of software creation at this point. Is there a market? Can I make money? Will the revenue justify the cost of development?

1.1.2 Writing Functional Specifications

Functional specifications may consist of one or more documents. Functional specification documents show the behavior or functionality of a software product on an abstract level. Assuming the product is a black box, the functional specifications define its input/output behavior. Functional specifications are based upon the product requirements documentation put forward by people who have contact with the enduser of the product or the customers.

In larger products, functional specifications may consist of separate documents for each feature of the product. For example, in a router product, you may have a functional specification document for RIP (Routing Information Protocol), another for security features, and so on.

Functional specifications are important because developers use them to create design documents. The documentation people also use them when they create manuals for end users. If different groups are working in different physical places, functional specifications and architecture documents (discussed next) are also a means to communicate among them. Keep in mind that sometimes during the product development phase you may need to amend functional specifications keeping in view new marketing requirements.

1.1.3 Creating Architecture and Design Documents

When you have all of the requirements collected and arranged, it is the turn of the technical architecture team, consisting of highly qualified technical specialists, to create the architecture of the product. The architecture defines different components of the product and how they interact with each other. In many cases the architecture also defines the technologies used to build the product. While creating the architecture documents of the project, the team also needs to consider the timelines of the project. This refers to the target date when the product is required to be on the market. Many excellent products fail because they are either too early or late to market. The marketing and sales people usually decide a suitable time frame to bring the product to market. Based on the timeline, the architecture team may drop some features of the product if it is not possible to bring the full-featured product to market within the required time limits.

Once components of the product have been decided and their functionality defined, interfaces are designed for these components to work together. In most cases, no component works in isolation; each one has to coordinate with other components of the product. Interfaces are the rules and regulations that define how these components will interact with each other. There may be major problems down the road if these interfaces are not designed properly and in a detailed way. Different people will work on different components of any large software development project and if they don't fully understand how a particular component will communicate with others, integration becomes a major problem.

For some products, new hardware is also required to make use of technology advancements. The architects of the product also need to consider this aspect of the product.

After defining architecture, software components and their interfaces, the next phase of development is the creation of design documents. At the architecture level, a component is defined as a black box that provides certain functionality. At the design documents stage, you

have to define what is in that black box. Senior software developers usually create design documents and these documents define individual software components to the level of functions and procedures. The design document is the last document completed before development of the software begins. These design documents are passed on to software developers and they start coding. Architecture documents and MRDs typically need to stay in sync, as sales and marketing will work from MRDs while engineering works from engineering documents.

1.1.4 Implementation and Coding

The software developers take the design documents and development tools (editors, compilers, debuggers etc.) and start writing software. This is usually the longest phase in the product life cycle. Each developer has to write his/her own code and collaborate with other developers to make sure that different components can interoperate with each other. A revision control system such as CVS (Concurrent Versions System) is needed in this phase. There are a few other open source revision control systems as well as commercial options. The version control system provides a central repository to store individual files. A typical software project may contain anywhere from hundreds to thousands of files. In large and complex projects, someone also needs to decide directory hierarchy so that files are stored in appropriate locations. During the development cycle, multiple persons may modify files. If everyone is not following the rules, this may easily break the whole compilation and building process. For example, duplicate definitions of the same variables may cause problems. Similarly, if included files are not written properly, you can easily cause the creation of loops. Other problems pop up when multiple files are included in a single file with conflicting definitions of variables and functions.

Coding guidelines should also be defined by architects or senior software developers. For example, if software is intended to be ported to some other platform as well, it should be written on a standard like ANSI.

During the implementation process, developers must write enough comments inside the code so that if anybody starts working on the code later on, he/she is able to understand what has already been written. Writing good comments is very important as all other documents, no matter how good they are, will be lost eventually. Ten years after the initial work, you may find only that information which is present inside the code in the form of comments.

Development tools also play an important role in this phase of the project. Good development tools save a lot of time for the developers, as well as saving money in terms of improved productivity. The most important development tools for time saving are editors and debuggers. A good editor helps a developer to write code quickly. A good debugger helps make the written code operational in a short period of time. Before starting the coding process, you should spend some time choosing good development tools.

Review meetings during the development phase also prove helpful. Potential problems are caught earlier in the development. These meeting are also helpful to keep track of whether the product is on time or if more effort is needed complete it in the required time frame. Sometimes you may also need to make some changes in the design of some components because of new

requirements from the marketing and sales people. Review meetings are a great tool to convey these new requirements. Again, architecture documents and MRDs are kept in sync with any changes/problems encountered during development.

1.1.5 Testing

Testing is probably the most important phase for long-term support as well as for the reputation of the company. If you don't control the quality of the software, it will not be able to compete with other products on the market. If software crashes at the customer site, your customer loses productivity as well money and you lose credibility. Sometimes these losses are huge. Unhappy customers will not buy your other products and will not refer other customers to you. You can avoid this situation by doing extensive testing. This testing is referred to as Quality Assurance, or QA, in most of the software world.

Usually testing starts as soon as the initial parts of the software are available. There are multiple types of testing and these are explained in this section. Each of these has its own importance.

1.1.5.1 Unit Testing

Unit testing is testing one part or one component of the product. The developer usually does this when he/she has completed writing code for that part of the product. This makes sure that the component is doing what it is intended to do. This also saves a lot of time for software testers as well as developers by eliminating many cycles of software being passed back and forth between the developer and the tester.

When a developer is confident that a particular part of the software is ready, he/she can write test cases to test functionality of this part of the software. The component is then forwarded to the software testing people who run test cases to make sure that the unit is working properly.

1.1.5.2 Sanity Testing

Sanity testing is a very basic check to see if all software components compile with each other without a problem. This is just to make sure that developers have not defined conflicting or multiple functions or global variable definitions.

1.1.5.3 Regression or Stress Testing

Regression or stress testing is a process done in some projects to carry out a test for a longer period of time. This type of testing is used to determine behavior of a product when used continuously over a period of time. It can reveal some bugs in software related to memory leakage. In some cases developers allocate memory but forget to release it. This problem is known as memory leakage. When a test is conducted for many days or weeks, this problem results in allocation of all of the available memory until no memory is left. This is the point where your software starts showing abnormal behavior.

Another potential problem in long-term operation is counter overflow. This occurs when you increment a counter but forget to decrement, it resulting in an overflow when the product is used for longer periods.

The regression testing may be started as soon as some components are ready. This testing process requires a very long period of time by its very nature. The process should be continued as more components of the product are integrated. The integration process and communication through interfaces may create new bugs in the code.

1.1.5.4 Functional Testing

Functional testing is carried out to make sure that the software is doing exactly what it is supposed to do. This type of testing is a must before any software is released to customers. Functional testing is done by people whose primary job is software testing, not the developers themselves. In small software projects where a company can't afford dedicated testers, other developers may do functional testing also. The key point to keep in mind is that the person who wrote a software component should not be the person who tested it. A developer will tend to test the software the way he/she has written it. He/she may easily miss any problems in the software.

The software testers need to prepare a testing plan for each component of the software. A testing plan consists of test cases that are run on the software. The software tester can prepare these test cases using functional specifications documents. The tester may also get help from the developer to create test cases. Each test case should include methodology used for testing and expected results.

In addition to test cases, the tester may also need to create a certain infrastructure or environment to test the functionality of a piece of code. For example, you may simulate a network to test routing algorithms that may be part of a routing product.

The next important job of the tester is to create a service request if an anomaly is found. The tester should include as much information in the service request as possible. Typical information included in reporting bugs includes:

- Test case description
- How the test was carried out
- Expected results
- Results obtained
- If a particular environment was created for testing, a description of that environment

The service request should be forwarded to the developer so that the developer may correct the problem. Many software packages are available in the market to track bugs and fix problems in software. There are many web-based tools as well. For a list of freely available open source projects, go to http://www.osdn.org or http://www.sourceforge.net and search for "bug track". OSDN (Open Source Developers Network) hosts many open source software development projects. You can find software packages that work with CVS also. CVS is explained in Chapter 6 of this book.

1.1.6 Software Releases

Before you start selling any software product, it is officially released. This means that you create a state of the software in your repository, make sure that it has been tested for functionality and freeze the code. A version number is assigned to released software. After releasing the software, development may continue but it will not make any change in the released software. The development is usually carried on in a new branch and it may contain new features of the product. The released software is updated only if a bug fixed version is released.

Usually companies assign incremental version numbers following some scheme when the next release of the software is sent to market. The change in version number depends on whether the new software contains a major change to the previous version or it contains bug fixes and enhancement to existing functionality. Releases are also important because they are typically compiled versions of a particular version of the code, and thus provide a stable set of binaries for testing.

1.1.6.1 Branches

In almost all serious software development projects, a revision or version control system is used. This version control system keeps a record of changes in source code files and is usually built in a tree-like structure. When software is released, the state of each file that is part of the release should be recorded. Future developments are made by creating branches to this state. Sometimes special branches may also be created that are solely used for bug fixing. CVS is discussed in detail in Chapter 6.

1.1.6.2 Release Notes

Every software version contains release notes. These release notes are prepared by people releasing the software version with the help of developers. Release notes show what happened in this software version. Typically the information includes:

- Bug fixes
- New functionality
- Detail of new features added to the software
- Any bugs that are not yet fixed
- Future enhancements
- If a user needs a change in the configuration process, it is also mentioned in the release notes

Typically a user should be given enough information to understand the new release enhancements and decide whether an upgrade is required or not.

1.1.7 Documentation

There are three broad categories of documentation related to software development processes.

1. Technical documentation developed during the development process. This includes architecture, functional and design documents.
2. Technical documentation prepared for technical support staff. This includes technical manuals that support staff use to provide customer support.
3. End-user manuals and guides. This is the documentation for the end user to assist the user getting started with the product and using it.

All three types of documents are necessary for different aspects of product support. Technical documents are necessary for future development, bug fixes, and adding new features. Technical documentation for technical support staff contains information that is too complicated for the end user to understand and use. The support staff needs this information in addition to user manuals to better support customers. Finally each product must contain user manuals.

Technical writers usually develop user manuals which are based on functional specifications. In the timelines of most software development projects, functional specifications are prepared before code development starts. So the technical writers can start writing user manuals while developers are writing code. By the time the product is ready, most of the work on user manuals has already been completed.

1.1.8 Support and New Features

Your customers need support when you start selling a product. This is true regardless of the size of the product, and even for products that are not software related. Most common support requests from customers are related to one of the following:

- The customer needs help in installation and getting started.
- The customer finds a bug and you need to release a patch or update to the whole product.
- The product does not fulfill customer requirements and a new feature is required by the customer.

In addition to that, you may also want to add new features to the product for the next release because competitor products have other features. Better support will increase your customer loyalty and will create referral business for you.

You may adopt two strategies to add new features.You may provide an upgrade to the current release as a patch, or wait until you have compiled and developed a list of new features and make a new version. Both of these strategies are useful depending how urgent the requirement for new features is.

1.2 Components of a Development System

Like any other system, a development system is composed of many components that work together to provide services to the developer for the software development task. Depending upon the requirements of a project, different types of components can be chosen. Many commercial companies also sell comprehensive development tools. On Linux systems, all of the development tools are available and you can choose some of these depending upon your level of expertise with these tools and your requirements. Typically each development platform consists of the following components:

- Hardware platform
- Operating system
- Editors
- Compilers and assemblers
- Debuggers
- Version control system
- Collaboration and bug tracking

Let us take a closer look on these components and what role they play in the development cycle.

1.2.1 Hardware Platform

This is the tangible part of the development system. A hardware platform is the choice of your hardware, PC or workstation, for the development system. You can choose a particular hardware platform depending upon different factors as listed below:

Cost	Depending upon budget, you may chose different types of hardware. Usually UNIX workstations are costly to buy and maintain. On the other hand, PC-based workstations are cheap and the maintenance cost is also low.
Performance	Usually UNIX workstations have high performance and stability as compared to PC-based solutions.
Tools	You also need to keep in mind availability of development tools on a particular platform.
Development Type	If the target system is the same as the host system on which development is done, the development is relatively easy and native tools are cheap as well, compared to cross-platform development tools.

Depending upon these factors, you may make a choice from the available hardware platforms for development.

If hardware is part of the final product, selection of hardware platform also depends upon customer/market requirement.

1.2.2 Operating System

Choice of a particular operating system may be made depending upon:

- Cost
- Availability of development tools
- Hardware platform
- Native or cross compiling

Some operating systems are cheaper than others. Linux is an excellent choice, as far as cost is concerned. Linux is also a very good operating system as it has all of the development tools available. Now you can install Linux on high-end workstations from Sun Microsystems, HP, and IBM as well as commodity PC hardware available everywhere. It provides stability and most of the people are familiar with development tools. You can also use the operating system for cross-platform development using GNU tools.

1.2.3 Editors

Editors play an important role in the development work. Easy-to-use and feature rich editors, like Emacs, increase developers' productivity. You should look at a few things while selecting editors. These features include:

- Understanding syntax of language
- Collapsing of context
- Support of tags
- Opening multiple files
- Easy editing for generally used editing functions like cut, copy, paste, search, replace and so on
- Multiple windows
- Support of user defined functions and macros

If you look at the open source community, you can find a lot of good editors available to developers. The most commonly used editors are Jed, Emacs and Xemacs. However, many other variants of these editors are also available. You can also use X-Windows-based editors available on Linux platform. A lot of people also edit in vi or vim, both of these have been very popular historically.

1.2.4 Compilers and Assemblers

Compilers and assemblers are the core development tools because they convert source code to executable form. Quality of compilers does affect the output code. For example, some compilers can do much better code optimization compared to others. If you are doing some cross-platform development, then your compiler should support code generation for the target machine as well.

GNU compilers collection, more commonly called GCC, is a comprehensive set of compilers for commonly used languages including the following:

- C
- C++
- Java
- Fortran

In addition to GCC, you can find a lot of other open source compilers available for Linux. Some of these are introduced in Chapter 3.

GNU utilities set, also known as binutils, includes GNU assembler and other utilities that can be used for many tasks. GNU assembler is used whenever you compile a program using GNU compiler.

1.2.5 Debuggers

Debuggers are the also an important part of all development systems. You can't write a program that is free of bugs. Debugging is a continuous part of software development and you need good tools for this purpose. GNU debugger, more commonly known as GDB, is a common debugger. Many other debuggers are also built on this debugger. The GNU debugger and some other debuggers will be introduced later in this book.

1.2.6 Version Control Systems

The revision control system is a must for any serious development effort where multiple developers work on a software product. The most popular version control system on Linux is known as Concurrent Versions System or CVS. CVS allows many people to work on files at the same time and provides a central repository to store files. Developers can check out files from this repository, make changes and check the files back into the repository. CVS also works with editors like GNU Emacs.

When multiple developers are modifying the same file at the same time, conflict may occur between different changes made by multiple developers. When a conflict is detected in the files being checked in, CVS provides a mechanism to merge the files appropriately.

CVS can be used over secure links as well. This is required when developers are not physically located at the same place. A server on the Internet can be used to provide secure access to the central software repository.

There are other version control systems as well which are popular in the software development community. Examples are Aegis, PRCS, RCS and SCCS.

1.2.7 E-mail and Collaboration

In any software development project, collaboration among developers, designers and architects as well as marketing people is a must. The objective can be achieved in many ways. Probably e-mail is the most efficient and cheapest way. Some collaboration tools provide more functionality than just e-mailing.

1.2.8 X-Windows

X-Windows is much more than just a GUI interface on Linux, but for development purposes, it provides a very good user interface. This is especially useful for editors like Emacs.

1.2.9 Miscellaneous Tools

Many miscellaneous tools are also important during the development process. Some of these tools are listed below:

- The `make` utility
- The `ar` program
- The `ranlib` utility
- The `hexdump` utility

Information about these tools is provided later in this book.

1.3 Selection Criteria for Hardware Platform

The development process needs computers, networks, storage, printing and other hardware components. However the important hardware decision is the selection of PCs and workstations for developers. There is no hard and fast rule about how to select a particular hardware platform. It depends upon the requirements of a development project. Some factors that you may keep in mind are as follows:

- Cost of the hardware.
- Availability of desired operating system on the hardware. For example, you can't run HP-UX on PCs.
- Availability of development tools.
- Maintenance cost.

There may be other factors as well and you are the best person to judge what you need. However, keep in mind that reliability of hardware is one major factor that people usually overlook. If you buy cheap systems that decrease productivity of developers, you lose a lot of money.

1.4 Selection Criteria for Software Development Tools

After selecting the hardware, software development tools are the next major initial expense in terms of money and time to set these up. Selection of software development tools depends upon the choice of hardware and operating system. In many cases GNU tools are very well suited. Selection of development tools also has a major effect on the productivity of the whole development team.

1.5 Managing Development Process

In large software development projects, management of the development process is a big task and a dedicated person may be needed to take care of this aspect of the project. A development manager usually acts as a binding and coordinating force among different parties with conflicting interests. These parties include:

- Marketing and sales people who put forward requirements, change requirements and come up with new requirements, usually when much of the work is already done!
- Architecture and design people.
- Software developers who always think that they always have less amount of time.
- Release management people.
- Software testers.
- Documentation writers.
- Senior managers who often push to complete the project earlier than the deadline.

Coordinating all of these parties is not an easy task. The manager has to convince senior management that a new feature needs that much time for development. At the same time he has to push developers to meet the deadlines. Some of the important tasks of software management in a real-life project are as follows.

1.5.1 Creating Deadlines

The manager usually coordinates with the software developers to set reasonable deadlines for certain features. These deadlines must conform to the product delivery time lines. The manager may have to arrange additional resources to complete feature development in the allotted time.

Project management software can help a manager to set and meet deadlines and track completion of different components.

1.5.2 Managing the Development Team

The manager has to keep track of how development among different parts of the software is going on. If part of the product is behind schedule, she has to re-arrange resources to get it back on track.. She may also need to hire new people to finish the development of a particular component on schedule.

1.5.3 Resolving Dependencies

Usually software components are dependent on one another. If the development of one component is lagging behind, it may affect development of other components. The development manager needs to keep an eye on these dependencies and how these may affect the overall progress of the project. Well-known project management methods are usually helpful for this task.

1.6 Linux Development Platform Specifications (LDPS) and Linux Standard Base (LSB)

Linux Development Platform Specifications or LDPS was an effort to design a common specification so that programs developed on one Linux distribution could be easily ported to other distributions. The specifications define components and packages that must be present on Linux development workstations. The latest version of the specifications at the time of writing this book is available at http://www.freestandards.org/ldps/1.1/ldps-1.1.html web site.

Linux Standard Base or LSB (http://www.linuxbase.org) is the new forum to standardize Linux distributions. LSB specifications 1.1.0 is available at the time of writing this book. LSB compliant applications can run on any LSB compliant distribution without any modification or recompilation process. Specifications are detailed and the latest version can be found at http://www.linuxbase.org/spec/.

1.6.1 Libraries

The following libraries will be available on LSB compliant systems. While developing applications for Linux, the developer can assume presence of these libraries on target machines, provided the target is LSB compliant.

- libX11
- libXt
- libGL
- libXext
- libICE
- libSM
- libdl
- libcrypt

- libz
- libncurses

1.6.2 Current Contributors to LSB

Major Linux vendors include:

- Caldera Inc.
- MandrakeSoft
- Red Hat Software
- The Debian Project
- TurboLinux Inc.
- SuSE
- VA Linux

References

1. LDPS web site at http://www.freestandards.org/ldps/
2. CVS web site at http://www.gnu.org/software/cvs/
3. Aegis at web site http://aegis.sourceforge.net/index.html
4. PRCS at its web site http://prcs.sourceforge.net/
5. GNU Software at http://www.gnu.org
6. Linux Standard Base at http://www.linuxbase.org
7. Open Source Developers Network at http://www.osdn.org