EVALUATING USER EXPERIENCE FOR OPERATING SYSTEM DEVELOPMENT

Raihan Hasnain Rahman

Student ID: 03201013

Department of Computer Science and Engineering

April 2008

BRAC University, Dhaka, Bangladesh

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the research performed by me (Raihan Hasnain Rahman) under the supervision of Dr. Mumit Khan, Associate Professor, Department of Computer Science and Engineering, BRAC University, Dhaka.

I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Signature

_____

(Dr. Mumit Khan)

Signature

_____

(Raihan Hasnain Rahman)

Acknowledgements

At first I would like to thank my thesis supervisor Dr. Mumit Khan for his help and cooperation. His useful hints, comments and recommendations helped me in every steps of this research. Without his continuous guidance I would not be able to complete this thesis.

Thanks to Arefin Jamal for downloading around 10 Linux distribution iso's for me. His helpful comments and discussions about many aspects of UX helped me a lot.

Thanks to Farzana Rahman for giving me her notebook computer on which I tested different Linux distributions and compiled this report.

Thanks to the contributor users of Wikipedia.

Disclaimer

This research is done for study purposes only. I have no affiliation with any company or any group. By this research document I do not advertise or preach to the users to use or leave any product of any company. Some of the articles, interviews and comments are used here in actual form (without any form of spell or grammar editing). This document is a compilation of many web articles and guidelines of many respected companies. On reference section there are links to those actual articles.

Abstract

User experience is not formally defined in software engineering yet; this research paper is an effort to relate user experience with software engineering with proper references. Operating system is the most important software that all level of users has to use all the time.

Linux operating systems are stable and able to fulfill desktop users need, but users seem to avoid Linux desktops for some reasons. The primary reason is usability. To develop Linux operating systems evaluation of users experience is needed.

This research was targeted for Linux but the outcomes are valid for any operating system.

Table of Contents

Chapter 4: OS and UX

Chapter 1: Prologue

1.1 Introduction

The user experience covers all the task a user performs with a computer starting from booting the computer, installing and using some software, using some hardware up to shutting down the computer. User experience includes those things that can be seen, for example – the desktop, icons, windows, messages, notifications and those things that can't be seen for example drivers, time taken to load a program, time taken to process a job etc. User experience and user interface are not essentially the same term. User interface needs actions taken by the user, but the experience may not require the user to do anything, for example the booting process.

Operating System is the largest piece of software and in fact the most important software we use every day. Users experience largely depends on the experience gained by the operating system. Therefore, to gain the maximum user experience it is required to evaluate the experiences – properly.

Unlike other desktop operating systems, Linux is the only OS - which can be developed by any one. For this freedom Linux has over a thousand versions. The Linux is known as powerful and stable operating system with the freedom of using infinite number of applications. For the strength of stability Linux was known as programmer's operating system from the beginning of the history, but recently Linux reached the doors of desktop users. Still, desktop users are not satisfied with Linux operating systems. To make Linux more acceptable for all level of desktop computer users, now it is important to evaluate user experience for the development of the next Linux operating systems.

1.2 Objectives

The term user experience is not defined in software engineering yet. For all software evaluating user experience is important before developers can decide what to do. Operating system is software as well. Objectives of this thesis are:

1. Establish a formal definition of User eXperience in context of Software Engineering,
2. Find out the factors of user experience for existing operating systems,
3. Give recommendations for developing Linux OS considering the user's experience.

1.3 Methodology

Internet is the primary source of information.

Some software engineering theory was taken from textbook.

User's experience cannot be known unless we talk to real users. User's forums, direct informal conversations and surveys helped to gather more information.

1.4 Scope of Research

The focus of this research was only on the software engineering part. User experience depends on the hardware a lot; in this research there is nothing about the hardware.

1.5 Limitations

Because of time limitation, extensive research could not be done. It is important to evaluate hardware integrations as well to max out the user experience. For example: Apple Mac OS X only works on Apple hardware.

Chapter 2: Introduction to UX

2.1 What is User Experience

2.1.1 User Experience in general

Users perceive products in different ways. Depending on the product's quality and service the users experience differently. The experience makes the users decide whether to use the product any further or not. If we do not like the product, we do not use it and we suggest other not to use it anymore. User experience is valid for any sort of product it is not a tech term. A food product, a consumer electronic device, a web site or a music player software everything can give users some experience. Experience can be positive or negative. To make any product successful knowing about user experience is important.

2.1.2 User Experience in computer world

"User experience" covers all aspects of the end users interaction with a product or service. It basically describes the requirements that provide all the needs of the user without any fuss or bother. On previous years, this term was used for all products from a magazine to nuclear reactor. Recently this term is applied for specifically web sites, software and digital devices.

First usage of user experience can be found in Edwards, E. C. and Kasik, D. J., "User Experience with the CYBER Graphics Terminal", Proceedings of VIM-21, pp 284-286, October 1974. The real practice of UX started when Apple's Donald A. Norman (User Experience Architect, Vice President (later) – Apple Computers) used this term in 1993.

In mid 90's many started to focus on this term improving Human-Computer Interaction (HCI). HCI is the key factor of user experience (described later). In recent years, practice of user experience became an integral part of software engineering. Specially, after year 2000 evaluating UX became essential part of web development. Testing user experience factors across all stages of web projects and ensuring that the best outcomes for the users are delivered before going online. Now we need a standard and clear definition of what UX is.

2.1.3 Defining UX

Defining user experience seemed to be the hardest part for me. UX is a term that is being used in many contexts. The definition varies on its context. Actually, there is no well-established formal definition for UX. Everyone seems to rely on Wikipedia's definition of UX. Let's take a look what Wikipedia says.

Wikipedia
(http://en.wikipedia.org/wiki/User_experience)

User experience, often abbreviated UX, is a term used to describe the overall experience and satisfaction a user has when using a product or system. It most commonly refers to a combination of software and business topics, such as selling over the web, but it applies to any result of interaction design. Voice User Interface (VUI) systems, for instance, are a frequently mentioned design that can lead to a poor user experience.

Expressing the definition in software engineering way,

The user experience is a term to describe the overall experience of a user that he had encountered using a product or service or system. It totally depends on few factors. The factors let a user feel good or bad. Good user experience means the software has better usability; everybody else is going to use the software. Bad user experience means the software is a failure; nobody is going to use it.

User experience is a part of Usability Engineering. The goal of UX is to provide maximum usability for the users, so that they can use the software with maximum satisfaction.

User Experience depends on few factors

- Human Computer Interaction
- Information architecture
- Interaction design
- User Interface design
- Usability

2.1.4 Importance of UX

Great user experience can make or break a product. There are good chances that an unhappy user is likely to tell others about his bad experience and might suggest them not to use it at all. On the other hand, good user experience makes the users happy and as they feel comfortable using the product – they are going to pull more users for that product.

2.1.5 Short Notes of the factors

2.1.5.1 Human Computer Interaction (HCI)

HCI is the study of man and machine interaction. It defines the ways of interacting between those two subjects and implements how hardware-software mechanism can be used to support them. The final goal is to improve the way of interaction to make it more usable for the users in an intuitive and adaptive way.

2.1.5.2 Information architecture (IA)

IA is the analysis and design of the data stored by information systems. IA is used in web development, database management and software engineering. IA expresses the model or structure of storing information.

2.1.5.3 Interaction Design (IxD)

IxD is the professional discipline that defines the behavior of interactive products and how products communicate their functionality to the people who use them. It deals with problems related to the usage of the system and relevant to the user or system that is interested in usability.

2.1.5.4 User interface design (UI Design)

UI design is the design of interfaces of software by which users directly interacts with the computer. UI design is a very important factor of UX. GUI falls under this category. Recent practices reveal that a better GUI can gain user experience in huge amount.

2.4.5.5 Usability

Usability is the key factor of user experience. It expresses that how easily a user can use a product. More usability defines the clarity of that product. Usability engineering studies the aspects and improvements of effectiveness, efficiency and satisfaction of users using any particular product.

2.2 Formal Definition

UX, is a term used to describe the overall experience and satisfaction a user has when using a product or system. It is an approach of product development that incorporates direct user feedback throughout the development cycle in order to increase usability, reduce costs and create products and tools that meet user needs.

Chapter 3: SE and UX

3.1 Why UX is important for SE

Why User eXperience is important? Let us go through these two articles. Andreas Pfeiffer wrote both of the articles.

Author Profile:
Andreas Pfeiffer, principal of Pfeiffer Consulting, an independent technology research institute and consulting operation focused on the needs of publishing, digital content production, and new media professionals.

3.1.1 Article 1: Why Features Don't Matter Anymore, The New Laws of Digital Technology

Technology Trends

The iPod was never sold on the grounds of its technical merits: Apple hit a gold-mine by marketing a cool new way of integrating music in your life. Even when Apple announced the iPod with video, it presented it not as the best multi-media player in the universe, but as a cool new way of watching "Desperate Housewives" and other TV shows.

In the seemingly never-ending debate about Apple's successes, announcements, new products and predicted-but-unannounced über-gadgets, features and technical specifications often seem to dominate the debate. Yet if there's one lesson to be learned from the company's recent successes, it is a very simple one: features don't matter anymore.

Welcome to the Age of User Experience.

One key aspect of modern digital devices is that technical specifications are easily copied and replicated: mega-pixel count in cameras, storage capacity in music players or processor speed in personal computers are the same everywhere. As a result, they provide only poor distinguishing factors for consumers when it comes to choosing between different brands.

That's where the overall user experience comes in. As computing and digital devices move more and more into the consumer space, features and functionalities will increasingly take the back-seat as motivators for technology adoption: as the iPod abundantly shows, user experience (along with a strong brand, and clever marketing) is much more important for the success of a device then technical specifications. Web designers have grasped the importance of good user experience a long time ago; now it is time the big technology providers to understand where the industry is headed.

10 fundamental rules for the age of user experience technology:

1) More features aren't better, it's worse.

Feature overload is becoming a real issue. The last thing a customer wants is confusion-and what's more confusing than comparing technical specifications, unless you are en expert? Only nerds get a kick out of reading feature lists. (I know - I'm one of them.)

2) You can't make things easier by adding to them.

Simplicity means getting something done in a minimum number of simple steps. Practically anything could be simpler - but you don't get there by adding features.

3) Confusion is the ultimate deal-breaker.

Confuse a customer, and you lose him. And nothing confuses more easily than complex features and unintuitive functionalities.

4) Style matters

Despite what nerds may think, style isn't fluff. On the grand scale of things, style is as important as features-if not more so. Style and elegance can contribute significantly to a good user experience. But style isn't just looks, it's a global approach. Fancy packaging isn't enough.

5) Only features that provide a good user experience will be used.

Why did the iPod catch on? Because it was so self-explanatory, and it remains the market leader in terms user experience. Sure, it may be excruciatingly difficult to make devices like digital media players or computers easy to use; but if a product is complex, intimidating or confusing, its chances for success are minimal.

6) Any feature that requires learning will only be adopted by a small fraction of users.

Learning new features, even the ones that a user might find interesting or intriguing, is a real issue: nobody has time. Getting consumers to upgrade and adopt new features is one of the biggest problems software publishers face these days.

7) Unused features are not only useless, they can slow you down and diminish ease of use.

Over time products become convoluted and increasingly complex to use. The frustration of not finding the one feature you need among a flurry of stuff you don't need, want or even understand, can be considerable. (Ever heard of program called Word?)

8) Users do not want to think about technology: what really counts is what it does for them.

The best tool is the one you don't notice. Why do you think pen and paper remain vastly popular for brainstorming? Because you don't have to think about them. Pencils don't crash.

9) Forget about the killer feature. Welcome to the age of the killer user-experience.

When technology achieves something desirable without being in your face, when it know how to integrate itself into your wishes and desires without distracting from them, that's when technology lives up to its potential. Unfortunately it's not that simple to get there.

10) Less is difficult, that's why less is more

Let's face it: it's usually harder to do simple things exceedingly well, than to just pile up features. The 80/20 rule applies here too: do well what 80 percent of your users do all the time, and you create a good user experience.

3.1.2 Article 2: User Experience Research

In 2005 and early 2006, Pfeiffer Consulting conducted an extensive research project collecting information about Macintosh and Windows operating systems. During the research interviews, which included users of both platforms, many Macintosh users stated that they found their computer “more fluid”, more productive, easier to use. They were, however, most often at a loss when they were asked to quantify their perceptions.

These recurring statements were intriguing for us: from a purely functional perspective, both operating systems have become increasingly similar, and even in terms of user interface, the basic concepts and user interface paradigms used by Windows and Macintosh are almost identical.

This discrepancy between user perception and technical features led us to have a closer look at user interface differences, usability, and productivity. During this research, we realized that the terms and concepts we use to analyze technology have remained surprisingly simplistic given the importance digital tools and devices play in our life.

How do we compare technology?

In our innovation-driven society, we tend to compare technology almost exclusively by looking at features and performance. More pixels equals better camera (or so we think), more gigabytes of storage means a better music player, and the list goes on, and on. Of course, in emerging fields of technology, this functionality-centric approach is utterly natural. (Even "cut and paste" was a big deal a long time ago, yet you would hardly find feature it in a product description today.) The problem is that as technology matures, features are not that important any more.

So if we don't look at features any more, what DO we look when we try to decide on the comparative merits of two products? Design? Style? Both are difficult to measure, and don't do much on their own. Of course there is always "user experience", an increasingly important aspect in the success of technology-related products, but an equally elusive one.

Which usability are we talking about?

It's not that we lack research on usability and ergonomics either: a quick search on the web reveals thousands of pages on the subject. Unfortunately, most of the available research focuses on web-usability, and somehow tacitly assumes that what applies to a web site surely will apply to other products as well.

Not so fast: what is deemed a good web-experience is profoundly different from a good user experience with computer software or on a digital device.
Web usability has a lot to do with creating a flawless "first-time" experience; using a device or computer program, on the other hand, is about fluidity and efficiency in very frequently repeated operations.

And that's only part of the problem: interacting with information which is generally what using the web is about is completely different from interacting with a device or a computer operating system.

So if we are interested in understanding the real differences between the Macintosh and Windows, say, or between an iPod and another music players, we are out of luck. We even lack the most basic terminology to describe, let alone quantify perceived qualities and problems. If we can quantify the speed of a computer, or it's productivity, why can't we do the same for the efficiency of a user interface or a device?

So maybe we need to sharpen our perception. Maybe we should add some words and concepts to our tools and terminologies. Maybe we need to accept that there is more to technology than features, and that it is time to look at something else than number of features, millions of pixels or hard-drive capacity.

Introducing: User Interface Friction

These considerations led us to come up with a new concept that has proven extremely useful in conducting technology analysis. Since in nature it is somewhat similar to the physical notion of friction, we called it User Interface Friction (UIF). We use it to describe and quantify the differences in fluidity and reactivity that exist between different operating systems, between software applications, even between different digital devices (music players, cameras, cell-

phones, among others).

We can find examples of User Interface Friction in many places. Did you ever notice how menu behavior could slow you down when you are trying to access a command, like selecting a program from the Start menu in Windows? That's User Interface Friction. Did you ever remark how scrolling through long lists of songs on a MP3 player can be annoying? User Interface Friction again. In fact, any user interface has some degree of friction. Some of it we may not notice, despite that fact that it exists, other examples can be severely annoying.

UIF is the resistance imposed upon a user-guided process through the operating system and the way the user interface reacts. In most cases, it has nothing to do with functionality: we use the term User Interface Friction to define the difference in fluidity and productivity that can be observed when running the same program or procedure on different computer systems, or when trying to achieve the goal on two similar digital devices.

User Interface Friction is inherent in any modern, menu-driven computer system and any device that sports a graphical user interface, and depends on a number of aspects, ranging from the speed at which the system displays a menu or sub-menu, to the efficiency of the mouse. Just like the smoothness of the paper or the ink-flow of a pen can impact the speed of handwriting, User Interface Friction affects practically any procedure where the user interacts through the user interface with the computer system. UIF could also be compared to the reactivity of a car when one presses the accelerator or the breaks.

(A detailed discussion of User Interface Friction, including user interface efficiency measures and productivity benchmarks of UIF can be found in the report User Interface Friction Research published by Pfeiffer Consulting.)

Subliminal does NOT mean insignificant

One of the most important aspects of User Interface Friction is that, while it may seem hardly noticeable, it shapes our overall user experience and can make the difference between a compelling product and an uninspiring one. Differences between two devices or programs may appear almost subliminal, but the user who compares both will notice them nonetheless, even if he or she could not explain why. Conversely, the sum of little, seemingly insignificant improvements add up and will make the difference between an excellent device and a lesser one.

Creating a "killer user experience" owes a lot to understanding subtle aspects such as User Interface Friction, and that is why I believe it is a very important notion. In many ways, creating an excellent user interface has become the digital equivalent of first-class manufacturing: we need it as users, and we need to understand what contributes to it if we are developing technology.

3.2 Current Practices in Software Engineering Theories

The term User Experience is a new one. It has been practiced in almost all software development organization for a long time, unofficially. In this chapter, we will try to figure out the practices of UX in software engineering.

Software engineering theories defines the process or road map to develop software. It talks about what is the problem to be solved and how can it be solved. It shows the ways of sort out the problem, analyse the problem, if it is solvable it finds out how can it be solved, what resources it might need and how the output would come out. It also gives the idea to sustain and to maintain the software after release.

Here, only those parts are discussed, which are related to user experience in some way.

3.2.1 Process

Process is the series of predictable steps, which helps building a product or system. Software engineering needs process to finish the development within time with best quality. It is important because it controls the activity of the whole development. Without a predefined process the output may become disorganized.

The steps are not strictly predefined. It can vary depending on the type of software or the organization. The output of the whole process is the desired software.

3.2.2 Process Assessment

The software process cannot guarantee that the development will be finished in time or that it will meet all the customer's requirements. The process itself is a learning process, it gets matured by time. The process should be assessed time to time to ensure that it can fulfil all the requirements with good quality.

The figure below shows the relationship of software process and few ways of assessments. We can define a software process, which will be examined by a generic software process assessment. This assessment will lead to capability determination and software process improvement. Capability determination is something that finds out the capabilities of the process. The software process improvement identifies the modification of the features of the process.

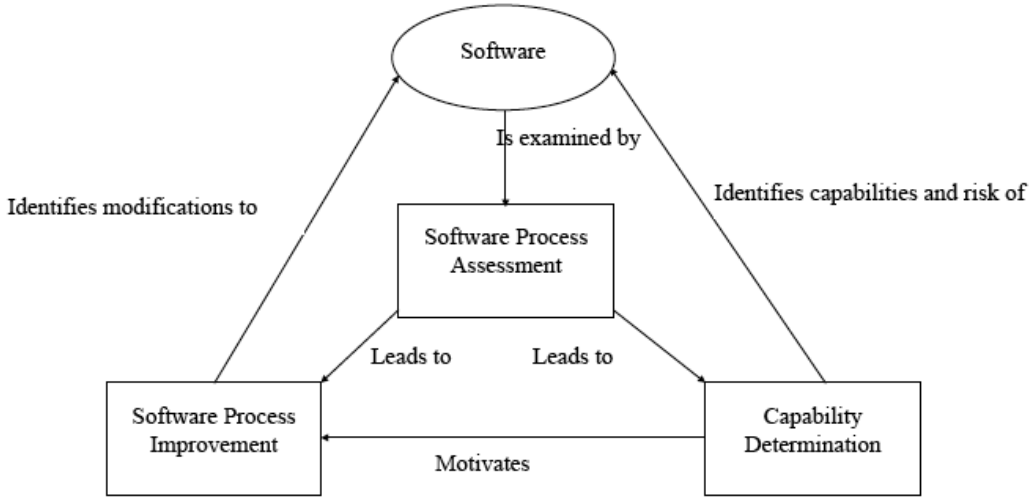


Fig: Process Assessment

This process assessment is important when we consider user experience. We cannot fix a concrete process we want users to suggest what they want, frequently. We need to examine the process to find out if it can satisfy the customer most. According to the users need and expectation we can improve the features of the software time to time. To achieve maximum user satisfaction, quality matters the most.

3.2.3 Agile software engineering

Agile software engineering combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of software. Agile method involved the customers/users with the development. The frequent discussion and/or

reviewing prototype with customers take place on every step of the development. When we care about user's expectation, agile engineering is what we need.

It is more adaptive than predictive. On other approaches of software engineering needs prediction of what is going to be. All the designs, schedules and resources need to be predicted before the development is going to start. As the goal is fixed, it is not possible to add things up while the development is going on. If we care about what users want, we need to communicate with them, show them what we have done and ask them for what they want.

3.2.4 The Adaptive Customer/User

Is agile software engineering for everyone? The answer is 'no'. It depends on the software and the users. Usually the customers tell the developers what they want, when they want and how much money they can pay. They fix a delivery date and they are done.

On the other hand, adaptive customers are something different; in this case customers have more control over the whole software development process. On every iteration of the development, they know what is going on. They can suggest what to do or what not to do after each iteration. This leads to a better relationship of users and developers.

The advantage of agile method is, developers can improve the software on each iteration of software process and it is guaranteed that customer will be satisfied with the output.

The disadvantage is, agile method might not work all the time, if the cooperating users are not responsive enough, or if they are too much choosy, meeting deadline might not be possible.

3.2.5 Agility and how to achieve agility

What is agility? It is the quality of sensing and responding according to it depending on the environment.

The Agile Alliance defined 12 principals for those who want to be agile.

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processed promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and design emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

3.2.6 The Agile Process

The agile process defines three key assumptions for software development.

It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priories will change as a project proceeds.

For many types of software, design and construction are interleaved. It is difficult to predict how much design is necessary before construction is used to prove the design.

Analysis, design, construction and testing are not as predictable as we might like.

3.2.7 Agile Software development (ASD)

ASD is based on collaboration. It has three phases - speculation, collaboration and learning. This life cycle adapts through the speculation and finally releases the software after some required cycles.

Speculation: Adaptive cycle planning uses project initiation information, project statement and user requirements to define the release cycle.

Collaboration: Users and developers work together to get the best out of it. If any change is required, developers can respond immediately.

Learning: As the process goes on, the developers go through a learning process. They learn from previous cycles, users and from the overall process.

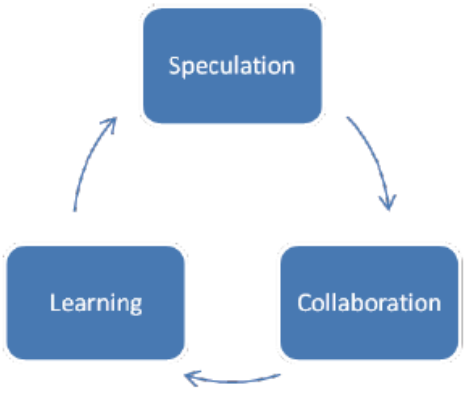


Figure: Showing ASD's 3 continuing steps of adopting through learning and implenting on a development

3.2.8 Agile Modelling (AM)

Agile Modelling (AM) is a practice-based methodology for effective modelling and documentation of software-based systems. AM is a collection of values, principals and practices for modelling software. AM can adopt methodologies from other methods like Extreme Programming (XP), Adaptive Software Development (ASD) and Dynamic Systems Development Method (DSDM).

The requirements change over time, it is necessary to respond on the changes immediately. AM is based on a collection of principles. It is easy to recognize the changes and acting upon by AM. It ensures that the incremental change and rapid feedback is accurately reflecting the needs of the development.

3.2.9 UI design

Interface design for software is something like floor plan. Architects design the floor plan to decide which will be where. User Interface is something by which users interact with the

software. Users do not know how the program works; they do not care about the architecture of the program. All they need to know is how can they use the software and what will they get as output. Interface design decides what components will be where so that the users can use those components to get the output.

There are three golden rules:

- Place users in control
- Reduce user's memory load
- Make the interface consistent

Place users in control: The software is made for users, not the developers. It is important to place users in control. Users like flexible interaction with the computer. Users do not want to be forced to do something. The interface has to be intuitive; users would not like to sit with a 600-page manual to use software. Users do not like to get interrupted; there should be ways to undo things otherwise users might get frustrated. There should be ways to customize things according to user's choice so that the design does not lose them. Users do not want to know technical details, so avoiding them is a good choice.

Reduce user's memory load: User do not like to memorize things when they use some software. The more they need to memorize, the more chances they will do something wrong. The system has to be well designed so that the system itself memorizes for the user and do not ask the same thing again and again. Unnecessary options make users disturbed. Using meaningful defaults is a good practice. The overall interface has to be intuitive and should be based on real world scenario.

Make the interface consistent: The interface should be consistent. Consider inventory program. Where the user has to input through 4 windows to get the work done. Look at the Save button, the user has to click on save on each window. Just because of the button is not in same place on each window, user might get annoyed in no time.

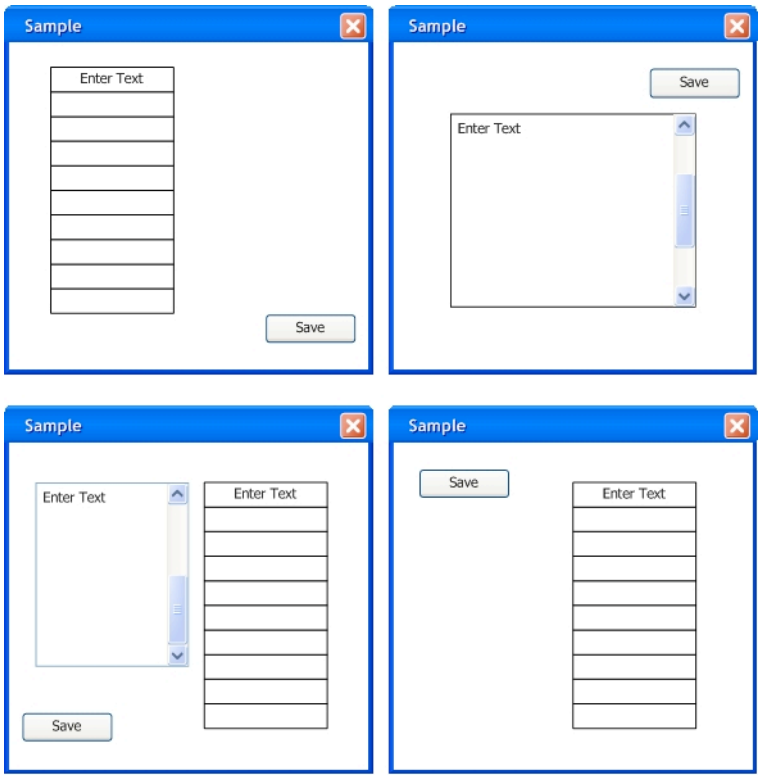


Figure: Sample windows showing inappropriate way of using buttons

3.2.10 Usability

Usability usually refers to software but is relevant to any product. Some ways to improve usability include:

- Shortening the time to accomplish tasks,
- Reducing the number of mistakes made,
- Reducing learning time, and
- Improving people's satisfaction with a system

Usability is often associated with the functionalities of the product, in addition to being solely a characteristic of the user interface. For example, an automobile lacking a reverse gear could be considered unusable according to the former view, and lacking in utility according to the latter view.

When evaluating user interfaces for usability, the definition can be as simple as "the perception of a target user of the effectiveness (fit for purpose) and efficiency (work or time required to use) of the Interface". Each component may be measured subjectively against criteria e.g. Principles of User Interface Design, to provide a metric, often expressed as a percentage.

It is important to distinguish between usability testing and usability engineering. Usability testing is the measurement of ease of use of a product or piece of software. In contrast, usability engineering (UE) is the research and design process that ensures a product with good usability.

Usability is an example of a non-functional requirement. As with other non-functional requirements, usability cannot be directly measured but must be quantified by means of indirect measures or attributes such as, for example, the number of reported problems with ease-of-use of a system.

3.2.11 User Interface analysis and design

The overall process for analysing and designing a user interface begins with the creation of different models of system. The human-computer oriented tasks are required to achieve system function are delineated; design issues that apply to all interface designs are considered; tools are used to prototype and ultimately implement the design model; and the result is evaluated by end-users for quality.

3.2.12 Interface Analysis and Design Models

Designing a successful software application or Web site requires a thorough understanding of the persons using the product, what they use it for, and the situations in which they use it.

User Interface Design is a well-used term and one that most of us believe we understand. User Task Analysis is other widely used term and again one which most of us would claim to comprehend.

In any software development life cycle it is widely accepted that there must be a Requirement Phase, followed by an Analysis and Design Phase, followed by a Development Phase, followed by several Testing Phases. Actually Analysis and Design are usually advocated as two separate stages but the working reality is that they are effectively one stage.

Analysis is essentially the understanding and refinement of the requirement and the documenting of such in a clear and logical fashion from which a Design can be developed. A Design for software is essentially the adoption of the analysis and the adaption of it to the technology available for implementation. It is possible to have a single Analysis of a Problem Domain but a separate Design for say a C++ based implementation and a Java based implementation. Design is generally influenced by the technology to be used and is adapted to that technology, thus it is difficult to re-use a design across varying technologies. The Analysis,

however, can be re-used to produce a suite of designs. It makes sense, therefore, to separate Analysis from Design.

So User Interface Analysis is about understanding and refining the detail of the Problem Domain and documenting it in a design independent fashion. The Analysis is technology independent. It can later be used to produce a design, which is technology specific.

3.2.13 The process

The analysis and design process for user interfaces is iterative and can be represented using a spiral model. Interface analysis and design process encompasses four distinct framework activities:

- User, task and environment analysis and modelling.
- Interface design
- Interface construction (implementation)
- Interface validation

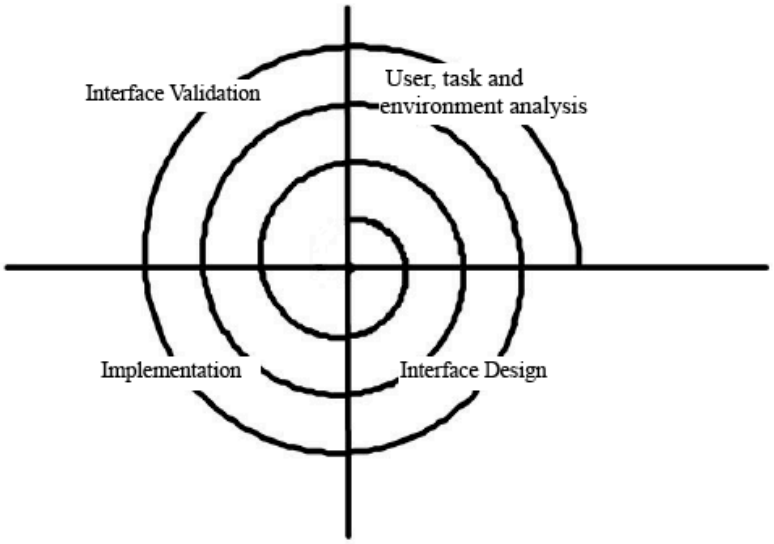


Figure: Spiral Model of Interface design process

The spiral implies that each of these will occur more than once, with each pass around the spiral representing additional elaboration of requirements and the resultant design. In most cases, the construction activity involves prototyping - the only practical way to validate what has been designed.

Interface analysis focuses on the profile of the users who will interact with the system. Skill level, business understanding and general receptiveness to the new system are recorded; and different user categories are defined. For each user category, requirements are elicited. In essence, the software engineer attempts to understand the system perception for each class of users.

Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described and elaborated. The information gathered as part of the analysis activity is used to create an analysis model for the interface. Using this model as a basis, the design activity commences.

The goal of the interface design is to define a set of interface objects and actions that enable a user to perform all defined tasks in a manner that meets every usability goals defined for the system.

Validation focuses on the ability of the interface to implement every user task correctly, to accommodate all task variations, and to achieve all general user requirements, the degree to which the interface is easy to use and easy to learn, and the users' acceptance of the user interface as a useful tool in their work.

3.2.14 Interface design steps

Once interface analysis has been completed, all tasks required by the end-user have been identifies in detail, and the interface design activity commences. Interface design, like all software engineering design, is an iterative process. Each user interface design step occurs a number of times, each elaborating and refining information developed in the preceding step.

Some of the suggested steps are

- Using information developed during interface analysis, define interface objects and actions.
- Define events that will cause the state of the user interface to change. Model this behaviour.
- Depict each interface state, as it will actually look to the end-user.
- Indicate how the user interprets the state of the system from information provided through the interface.

In some cases, the interface designer may begin with sketches of each interface state and ten works backward to define objects, actions and other important design information. Regardless of the sequence of design tasks, the designer must always follow the golden rules, model how the interface will be implemented and consider the environment (display technology, operating system, development tools) that will be used.

3.2.15 Design Issues

As the design of a user interface evolves, four common design issues almost always surface:

- System response time
- User help facilities
- Error handling
- Command labelling

Response time: System response time is measures from the point at which the user performs some control action until the software responds with the desired output or action. It has two important characteristics: length and variability. If system response is too long, user frustration and stress is the inevitable result. Variability refers to the deviation from average response time and in many ways it is the most important response time characteristic.

Help facilities: Users are not supposed to know everything. They require helps frequently and they expect to get help from the interface itself. First of all a intuitive interface helps a lot and second thing is, people looks for help menu, famous F1 key or some help icon. The way of helping is important, depending on the software it can be textual, visual or even can be printed manual.

Error handling: Though error messages scare people, but showing error messages is the worst way of hiding things. Meaningful and helpful error messages help users to fix things. It helps the

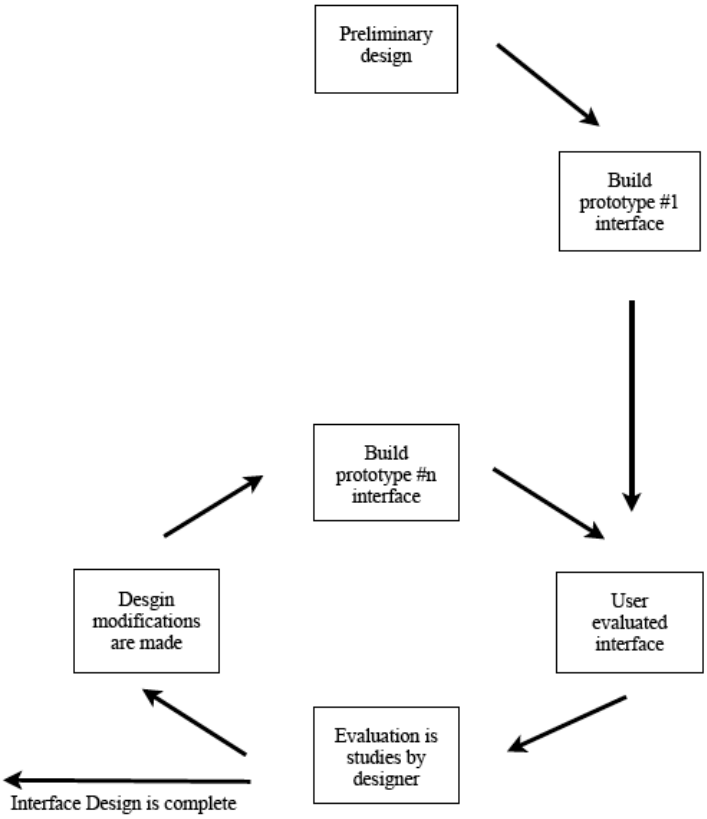
user to ask support from technical support over phone or emails. The error message should contain technical information - precisely. Error code and possible way of solving the problem might help a lot.

Menu and command labelling: The interactive interface has lots of menus. Appropriate and usual labels help users. Menu labels should be self-explanatory. Short-cut keys are very helpful. The users appreciate appropriate short-cut keys.

3.2.16 Design Evaluation

Once an operational user interface prototype has been created, it must be evaluated to determine whether it meets the needs of the user. Evaluation can span a formality spectrum that ranges from an informal test drive, in which a user provides feedback to a formally designed study that uses statistical methods for the evaluation of questionnaires completed by a population of end-users.

We can consider this cycle



3.2.17 UI Design Principals

UI design principals describe a collection of principles for improving the quality of your user interface design. These principles are:

1. **The structure principle:** Your design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things,

differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with your overall user interface architecture.

2. **The simplicity principle:** Your design should make simple, common tasks simple to do, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

3. **The visibility principle:** Your design should keep all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with too many alternatives or confuse them with unneeded information.

4. **The feedback principle**: Your design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.

5. **The tolerance principle**: Your design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions reasonable.

6. **The reuse principle:** Your design should reuse internal and external components and behaviours, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

3.3 Current and Future Prospects

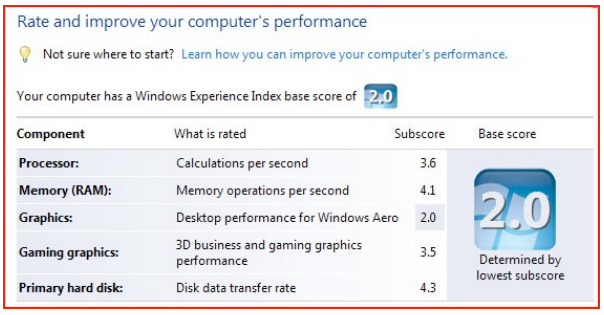
Web development is using the UX term for a long time. As web sites are directly used by the users, site developers are concerned about the usability. With the rise of Web 2.0 and web based applications. Web engineers are considering the needs of UX 2.0 where the study of UX will be done remotely and unmoderated. For rapid usability development study of UX is required.

Google uses this UX study very seriously and they actually develop their web based applications based on users' feedback.

Apple started using user experience studies long ago. They are continuing their trend. Apple's concentration is to be consistent on all applications along with their operating system and even with their hardware.

IBM takes care of their usability by UX researches. They invite users to volunteer in their UX researches for their product evaluations.

Recently Microsoft became concerned about UX. They started a scheme called Windows Experience score. This WE gives idea to the users that how well the Vista will run on their computer. There are many other examples. These are only few examples to give you the major idea.



Chapter 4: OS and UX

4.1 Operating Systems

4.1.1 What is an Operating System

Operating System (OS) is the software that manages the sharing of the resources of a computer and provides programmers with an interface used to access those resources. An operating system processes system data and user input, and responds by allocating and managing tasks and internal system resources as a service to users and programs of the system. At the foundation of all system software, an operating system performs basic tasks such as controlling and allocating memory, prioritizing system requests, controlling input and output devices, facilitating computer networking and managing files. Most operating systems come with an application that provides an interface for managing the operating system. The operating system forms a platform for other software. Modern operating systems can be found on anything composed of integrated circuits, like personal computers, internet servers, cell phones, music players, routers, switches, wireless access points, network storage, game consoles, digital cameras, sewing machines and telescopes. Many of the ideas first developed for embedded systems like Forth for telescopes still boot up modern Macintosh computers today, while the PC embedded systems rely on BIOS. The first system put on "bare metal" hardware is usually called firmware. On desktop computers these are the first system you see, such as Amibios or Phoenix bios, for basic input and output of attached components like memory, monitor, and keyboard. Most users equate a visual interface as the "Operating System", such as a desktop environment made ubiquitous by the IBM Personal Computer, and recently the Apple IPod.

4.1.2 Functions of Operating System

4.1.2.1Process management

Every program running on a computer, be it a service or an application, is a process. Originally only one process per CPU could be run at a time. Older microcomputer OSs such as MS-DOS did not attempt to bypass this limit, with the exception of interrupt processing, and only one process could be run under them.

Most operating systems enable concurrent execution of many processes and programs at once via multitasking, even with one CPU. The mechanism was used in mainframes since the early 1960s, but in the personal computers it became available in the 1990s. Process management is an operating system's way of dealing with running those multiple processes. On the most fundamental of computers (those containing one processor with one core) multitasking is done by simply switching processes quickly. Depending on the operating system, as more processes run, either each time slice will become smaller or there will be a longer delay before each process is given a chance to run. Process management involves computing and distributing CPU time as well as other resources. Most operating systems allow a process to be assigned a priority, which affects its allocation of CPU time. Interactive operating systems also employ some level of feedback in which the task with which the user is working receives higher priority. Interrupt driven processes will normally run at a very high priority. In many systems there is a background process, such as the System Idle Process in Windows, which will run when no other process is waiting for the CPU.

4.1.2.2Memory management

Current computer architectures arrange the computer's system in a hierarchical manner, starting from the fastest registers, CPU cache, RAM and disk storage. An operating system's disk manager coordinates the use of these various types of memory by tracking which one is available, which is

to be allocated or de-allocated and how to move data between them. This activity, usually referred to as virtual memory management, increases the amount of memory available for each process by making the disk storage seem like main memory. There is a speed penalty associated with using disks or other slower storage as memory – if running processes require significantly more RAM than is available, the system may start thrashing. This can happen either because one process requires a large amount of RAM or because two or more processes compete for a larger amount of memory than is available. This then leads to constant transfer of each process's data to slower storage.

Another important part of memory management is managing virtual addresses. If multiple processes are in memory at once, they must be prevented from interfering with each other's memory (unless there is an explicit request to utilize shared memory). This is achieved by having separate address spaces. Each process sees the whole virtual address space, typically from address 0 up to the maximum size of virtual memory, as uniquely assigned to it. The operating system maintains a page table that matches virtual addresses to physical addresses. These memory allocations are tracked so that when a process terminates, all memory used by that process could be made available for other processes.

The operating system can also write inactive memory pages to secondary storage. This process is called "paging" or "swapping" – the terminology varies between operating systems.

It is also typical for operating systems to employ otherwise unused physical memory as a page cache; requests for data from a slower device can be retained in memory to improve performance. The operating system can also preload the in-memory cache with data that may be requested by the user in the near future; Windows Vista's SuperFetch is an example of this.

4.1.2.3Disk and file systems

Generally, operating systems include support for file systems.

Modern file systems comprise a hierarchy of directories. While the idea is conceptually similar across all general-purpose file systems, some differences in implementation exist. Two noticeable examples of this are the character used to separate directories, and case sensitivity.

Unix demarcates its path components with a slash (/), a convention followed by operating systems that emulated it or at least its concept of hierarchical directories, such as Linux, Amiga OS and Mac OS X. MS-DOS also emulated this feature, but had already also adopted the CP/M convention of using slashes for additional options to commands, so instead used the backslash (\) as its component separator. Microsoft Windows continues with this convention; Japanese editions of Windows use ¥, and Korean editions use ₩. Prior to Mac OS X, versions of Mac OS use a colon (:) for a path separator. RISC OS uses a period (.).

Unix and Unix-like operating systems allow for any character in file names other than the slash and NUL characters (including line feed (LF) and other control characters). Unix file names are case sensitive, which allows multiple files to be created with names that differ only in case. For example New, new, NEW are three different folders. By contrast, Microsoft Windows file names are not case sensitive by default. Windows also has a larger set of punctuation characters that are not allowed in file names. For example CON, LPT1.

File systems may provide journaling, which provides safe recovery in the event of a system crash. A journaled file system writes information twice: first to the journal, which is a log of file system operations, then to its proper place in the ordinary file system. In the event of a crash, the system can recover to a consistent state by replaying a portion of the journal. In contrast, non-journaled file systems typically need to be examined in their entirety by a utility such as fsck or chkdsk. Soft update is an alternative to journaling that avoids the redundant writes by carefully ordering the update operations. Log-structured file systems and ZFS also differ from traditional journaled file systems in that they avoid inconsistencies by always writing new copies of the data, eschewing in-place updates.

Many Linux distributions support some or all of ext2, ext3, ReiserFS, Reiser4, GFS, GFS2, OCFS, OCFS2, and NILFS. Linux also has full support for XFS and JFS, along with the FAT file systems, and NTFS.

Microsoft Windows includes support for FAT12, FAT16, FAT32, and NTFS. The NTFS file system is the most efficient and reliable of the four Windows file systems, and as of Windows Vista, is the only file system, which the operating system can be installed on. Windows Embedded CE 6.0 introduced ExFAT; a file system suitable for flash drives.

Mac OS X supports HFS+ with journaling as its primary file system. It is derived from the Hierarchical File System of the earlier Mac OS. Mac OS X has facilities to read and write FAT16, FAT32, NTFS and other file systems, but cannot be installed to them.

Common to all these (and other) operating systems is support for file systems typically found on removable media. FAT12 is the file system most commonly found on floppy discs. ISO 9660 and Universal Disk Format are two common formats that target Compact Discs and DVDs, respectively. Mount Rainier is a newer extension to UDF supported by Linux 2.6 kernels and Windows Vista that facilitates rewriting to DVDs in the same fashion as has been possible with floppy disks.

4.1.2.4Networking

Current operating systems generally support a variety of networking protocols. Most are capable of using the TCP/IP networking protocols. This means that computers running dissimilar operating systems can participate in a common network for sharing resources such as computing, files, printers, and scanners using either wired or wireless connections.

Many operating systems also support one or more vendor-specific legacy networking protocols as well, for example, SNA on IBM systems, DECnet on systems from Digital Equipment Corporation, and Microsoft-specific protocols on Windows. Specific protocols for specific tasks may also be supported such as NFS for file access.

4.1.2.5Security

Many operating systems include some level of security. Security is based on the two ideas that:

- The operating system provides access to a number of resources, directly or indirectly, such as files on a local disk, privileged system calls, personal information about users, and the services offered by the programs running on the system;
- The operating system is capable of distinguishing between some requesters of these resources who are authorized (allowed) to access the resource, and others who are not authorized (forbidden). While some systems may simply distinguish between "privileged" and "non-privileged", systems commonly have a form of requester identity, such as a user name. Requesters, in turn, divide into two categories:
    - Internal security: an already running program. On some systems, a program once it is running has no limitations, but commonly the program has an identity, which it keeps and is used to check all of its requests for resources.
    - External security: a new request from outside the computer, such as a login at a connected console or some kind of network connection. To establish identity there may be a process of authentication. Often a username must be quoted, and each username may have a password. Other methods of authentication, such as magnetic cards or biometric data, might be used instead. In some cases, especially connections from the network, resources may be accessed with no authentication at all.

In addition to the allow/disallow model of security, a system with a high level of security will also offer auditing options. These would allow tracking of requests for access to resources (such as, "who has been reading this file?").

4.1.2.5.a Internal security

Internal security can be thought of as protecting the computer's resources from the programs concurrently running on the system. Most operating systems set programs running natively on the computer's processor, so the problem arises of how to stop these programs doing the same task and having the same privileges as the operating system (which is after all just a program too). Processors used for general-purpose operating systems generally have a hardware concept of privilege. Generally less privileged programs are automatically blocked from using certain hardware instructions, such as those to read or write from external devices like disks. Instead, they have to ask the privileged program (operating system kernel) to read or write. The operating system therefore gets the chance to check the program's identity and allow or refuse the request.

Internal security is especially relevant for multi-user systems; it allows each user of the system to have private files that the other users cannot tamper with or read. Internal security is also vital if auditing is to be of any use, since a program can potentially bypass the operating system, inclusive of bypassing auditing.

4.1.2.5.b External security

Typically an operating system offers (or hosts) various services to other network computers and users. These services are usually provided through ports or numbered access points beyond the operating system's network address. Services include offerings such as file sharing; print services, email, web sites, and file transfer protocols (FTP), most of which can have compromised security.

At the front line of security are hardware devices known as firewalls or intrusion detection/prevention systems. At the operating system level, there are a number of software firewalls available, as well as intrusion detection/prevention systems. Most modern operating systems include a software firewall, which is enabled by default. A software firewall can be configured to allow or deny network traffic to or from a service or application running on the operating system. Therefore, one can install and be running an insecure service, such as Telnet or FTP, and not have to be threatened by a security breach because the firewall would deny all traffic trying to connect to the service on that port.

4.1.2.6Graphical user interfaces

Today, most modern computer systems contain Graphical User Interfaces. In some computer systems the GUI is integrated into the kernel—for example, in the original implementations of Microsoft Windows and Mac OS, the graphical subsystem was actually part of the kernel. Other operating systems, some older ones and some modern ones, are modular, separating the graphics subsystem from the kernel and the Operating System. In the 1980's UNIX, VMS and many others had operating systems that were built this way. Today Linux, and Mac OS X are also built this way.

Many computer systems allow the user to install or create any user interface they desire. The X Window System in conjunction with GNOME or KDE is a commonly found setup on most Unix and Unix-like (BSD, Linux, Minix) systems. Numerous Unix-based GUIs have existed over time, most derived from X11. Competition among the various vendors of Unix (HP, IBM, Sun) led to much fragmentation, though an effort to standardize in the 1990s to COSE and CDE failed for the most part due to various reasons, eventually eclipsed by the widespread adoption of GNOME and KDE. Prior to open source-based toolkits and desktop environments, Motif was the prevalent toolkit/desktop combination (and was the basis upon which CDE was developed).

Graphical user interfaces evolve over time. For example, Windows has modified its user interface almost every time a new major version of Windows is released, and the Mac OS GUI changed dramatically with the introduction of Mac OS X in 2001.

4.1.2.7Device drivers

A device driver is a specific type of computer software developed to allow interaction with hardware devices. Typically this constitutes an interface for communicating with the device, through the specific computer bus or communications subsystem that the hardware is connected to, providing commands to and/or receiving data from the device, and on the other end, the requisite interfaces to the operating system and software applications. It is a specialized hardware-dependent computer program which is also operating system specific that enables another program, typically an operating system or applications software package or computer program running under the operating system kernel, to interact transparently with a hardware device, and usually provides the requisite interrupt handling necessary for any necessary asynchronous time-dependent hardware interfacing needs.

The key design goal of device drivers is abstraction. Every model of hardware (even within the same class of device) is different. Manufacturers that provide more reliable or better performance also release newer models and these newer models are often controlled differently. Computers and their operating systems cannot be expected to know how to control every device, both now and in the future. To solve this problem, OSs essentially dictates how every type of device should be controlled. The function of the device driver is then to translate these OS mandated function calls into device specific calls. In theory a new device, which is controlled in a new manner, should function correctly if a suitable driver is available. This new driver will ensure that the device appears to operate as usual from the operating systems' point of view for any person.

4.1.3 Desktop Operating Systems

What we use in desktop computers is called desktop operating systems. Desktop OSs has to be versatile as the usage of the computer solely depends on the user or users of that computer. For example – in a house, a desktop computer may be used as gaming console, web publication or just as a video player. As operating system is the one who lets the users to use their application, that OS must be able to handle all sorts of things. Desktop and notebook operating systems are same.

In most cases users do not have any idea that they use operating systems all the time. Even if they know, they do not care. Still they experiences differently in different environment. “After installing Windows Vista I do not need to use my Music Player’s CD anymore, Vista gets everything automatically” – this type of user feedback we get regularly.

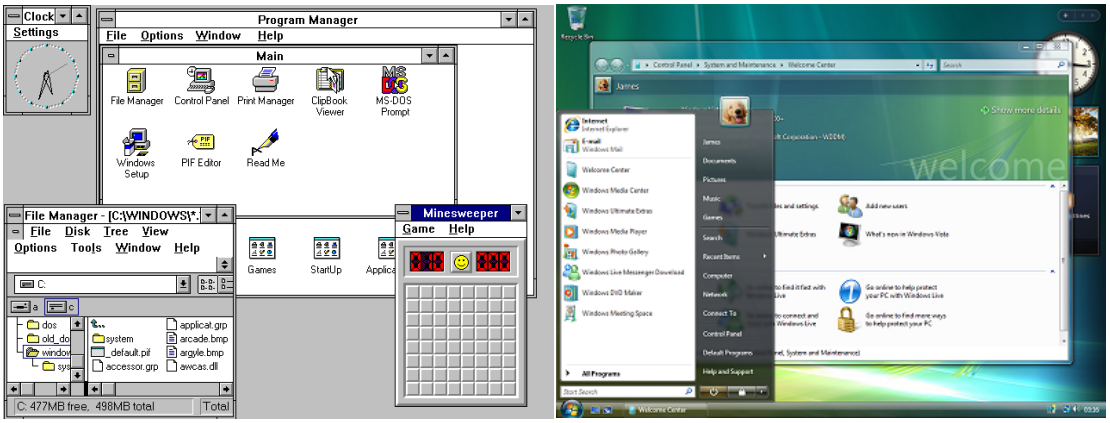
Desktop/notebook computer users use operating systems knowingly or unknowingly. As it (OS) is directly interacting with the users, it needs to satisfy the users in every extent. Some features a desktop OS should have –

- Responsive
- Should have a good number of available application software
- Nice looking friendly interface
- Should support new and old (legacy) devices
- Stability

4.2 Desktop Environments

In graphical computing, a desktop environment (DE, sometimes desktop manager) presents a graphical user interface (GUI) to the user. The name is derived from the desktop metaphor used by most of these interfaces, as opposed to the earlier, textual command line interfaces (CLI). A desktop environment typically provides icons, windows, toolbars, folders, wallpapers, and desktop widgets. In addition, a desktop environment may offer collaboration support like drag and drop and inter-process notification. On the whole, the purpose of a desktop environment is the consistent integration of a graphical user interface and its applications to the user with a consistent look and feel.

DE's are the one who can determine how friendly the interface is. For example, CLI wasn't friendly at all and if we compare Windows 3.1's DE with the Vista's DE we can easily see the difference.



Desktop is the workspace for a user. User has work on the desktop, so is supposed to be convenient if the desktop is customizable and user-friendly. Different operating system comes with different DE. Here are some short notes of few well-known DEs.

4.2.1 Windows Vista's Aero

Windows Aero is the graphical user interface and the default theme in most editions of Windows Vista, an operating system released by Microsoft in November 2006. It will also be available in Windows Server 2008, which was released on February 27, 2008. Its name is a backronym for Authentic, Energetic, Reflective and Open. Intended to be a cleaner, more powerful, more efficient and more aesthetically pleasing user interface than the previously used theme (Luna), it includes new transparencies, live thumbnails, live icons, animations and eye candy. Aero also encompasses a set of user interface design guidelines for Microsoft Windows.

4.2.1.1History

Like Windows XP, The Iconfactory designed Aero's base icons.

Until the release of Windows Vista Beta 1 in July 2005, little or nothing had been shown of Aero in public/leaked builds. Previous user interfaces were Plex, which was featured in Longhorn builds 3683-4029; Slate, which was featured in build 4051 and was available until build 4093; and Jade (build 4074, 4083 and 4093, actually an early preview of Aero). Microsoft started using Aero in public builds in build 5048. The first build with full-featured Aero was build 5219. Build 5270 (released in December 2005) contained an implementation of Aero which was virtually complete, according to sources at Microsoft, though a number of stylistic changes were introduced between then and the operating system's release.

Originally, Aero was to have three levels available, one code-named "To Go", which had the Desktop Window Manager (DWM) composition engine (previously known as DCE) disabled. The next was to be AeroExpress, lacking many features of the highest-level code-named Aero Glass. However, in December 2005, Microsoft announced that there would only be two levels available,

"Windows Vista Aero" and "Windows Vista Basic", with the previous "Express" level integrated into the new "Windows Vista Aero" level. A control panel was added to enable the user to fine-tune this functionality, such as being able to turn off the "glass" translucency effect. These levels are provided so that the Aero interface (to some extent) can be used with a relatively low-end graphics card.

Initially, a variation of Aero, codenamed "Aero Diamond", was slated to be the user interface for the Windows Vista Media Center experience. Although there has been no official mention of Diamond for a number of years, it may refer to the expectation that the interface will be written in pure XAML (as was Aero initially) but this has not been confirmed.

4.2.1.2 User interface

For the first time since the release of Windows 95, Microsoft has completely revised its user interface guidelines, covering aesthetics, common controls such as buttons and radio buttons, task dialogs, wizards, common dialogs, control panels, icons, fonts, user notifications, and the "tone" of text used.

4.2.1.3 Aero Wizards

Wizard 97 has been the prevailing standard for wizard design, visual layout, and functionality used in Windows 98 through to Windows Server 2003, as well as most Microsoft products in that time frame. Aero Wizards are the replacement for Wizard 97, incorporating visual updates to match the aesthetics of the rest of Aero, as well as changing the interaction flow.

More specifically:

- To increase the efficiency of the wizard, The "Welcome" pages in Wizard 97 are no longer used. (A precursor to this change was implied in a number of wizards in products such as SQL Server 2005 where a check box was added to welcome pages, allowing a user to disable the welcome page in future uses of the wizard).
- Aero Wizards can be resized, whereas the Wizard 97 guidelines defined exact sizes for wizard window and content sizes.
- The purpose of any given Aero Wizard page is more clearly stated at the top.
- A new kind of control called a "Command link" provides a single-click operation to choose from a short list of options.
- The "Back" button has moved to the top-left corner of the wizard window and matches the visual style of the back button in other Vista applications. This is done to give more focus to the commit choices. The "Next" button is only shown on pages where it is necessary.
- The notion of "Commit pages" is introduced, where it is made clear that the next step will be the actual process that the wizard is being used to enact. If no follow-up information needs to be communicated, these are the last pages in a wizard. Typically a commit page has a button at the bottom-right that is labeled with the action to be taken, such as "Create account".
- At the end of a wizard, a "Follow-up page" can be used to direct the user to related tasks that they may be interested in immediately after completing the wizard. For example, a follow-up for a CD burning wizard may present options like "Duplicate this disk" and "Make a disk label".

4.2.1.4 Notifications

Notifications allow an application or operating system component with an icon in the system tray to create a pop-up window with some information about an event or problem. These windows, first introduced in Windows 2000 and known colloquially as "balloons", are similar in

appearance to the speech balloons that are commonly seen in comics. Balloons were often criticized in prior versions of Windows due to their intrusiveness, especially with regard to how they interacted with full-screen applications such as games. Notifications in Aero aim to be less intrusive by gradually fading in and out, and not appearing at all if a full-screen application or screensaver is being displayed – in these cases, notifications are queued until an appropriate time. Larger icons and multiple font sizes and colors are also introduced with Aero's notification windows.

4.2.1.5 Font

The Segoe UI typeface is the new default font for Aero with languages that use Latin, Greek, and Cyrillic character sets. The default font size is also increased from 8pt to 9pt to improve readability. In the Segoe UI typeface, the numeral zero ("0") is narrow, while capital letter "O" is wider, and numeral one ("1") has a top hook, while capital letter "I" has equal crown and base.

4.2.1.6 Phrasing tone

The Vista User Experience Guidelines also address the issue of "tone" in the writing of text used with the Aero user interface. Prior design guidelines from Microsoft had not done much to address the issue of how user interface text is phrased, and as such, the way that information and requests are presented to the user had not been consistent between parts of the operating system.

Research done by Microsoft informed them that users were finding Windows difficult to use and understand. Users were dissatisfied or felt insulted because of the phrasing of some messages. In particular, computer terminology and jargon were overused and used inconsistently, creating a barrier to understanding for newer users, and messages were unclear or perceived as patronizing.

The guidelines for Vista and its applications suggest messages that present technically accurate advice concisely, objectively, and positively, and assume an intelligent user motivated to solve a particular problem. Specific advice includes the use of the second person and the active voice (e.g. "Print the photos on your camera") and avoidance of words like "please" and "sorry".

4.2.1.7 Requirements

Microsoft has listed the following requirements for what they call a Vista Premium Ready PC. A PC that meets or exceeds these requirements will be able to use the new Aero technologies.

- A 1 GHz 32-bit (x86) or 64-bit (x64) processor
- 1 GB of system memory
- A Direct3D 9 compatible graphics processor with a Windows Display Driver Model (WDDM) driver, Pixel Shader 2.0 in hardware, and a minimum of 128 MB of Video RAM
- 40 GB hard drive with 15 GB free space
- DVD-ROM Drive
- Audio output and Internet access

4.2.2 Mac OS X's Aqua

Aqua is the graphical user interface and primary visual theme of Apple Inc.'s Mac OS X operating system. It is based around the theme of water, as its name suggests, with droplet-like elements and liberal use of translucency and reflection effects.

The Aqua theme and user interface was first introduced at the January 2000 Macworld Conference & Expo in San Francisco. Aqua's first appearance in a commercial product was in the July 2000 release of iMovie 2.

Aqua design elements make up the uniform appearance of most Mac OS X applications. Its goal is to "incorporate color, depth, translucence, and complex textures into a visually appealing interface" in Mac OS X applications. Although Aqua is made up of the entire user interface, two notable features of Aqua are gel-like buttons (such as the ones colored red, yellow, and green that control the windows), and a Dock, which facilitates the launching of and navigation between applications.

Aqua is the successor to Platinum, which was used in Mac OS 8 through 9.

4.2.2.1 Evolution

Much of Aqua's original design was intended to complement the translucent two-tone look of Apple's contemporaneous hardware, primarily the original bondi blue iMac. In 2003 and 2004, Apple moved to the use of brushed metal in their industrial design (such as with the aluminum Apple Cinema Displays); Aqua changed accordingly, incorporating the additional brushed metal look while deemphasizing the pinstripe backgrounds and transparency effects. In recent years, however, the brushed metal look has also been abandoned, in favor of white semi-reflective plastic, similar to the industrial design of the original iPod. This somewhat inconsistent mix of interface styles has been controversial among the Mac OS X user community. Apple replaced these inconsistent window themes with the introduction of Mac OS X Leopard.

Up until Mac OS X Leopard, each successive release of Mac OS X has brought new “Aqua Blue” wallpaper. It should be noted that in recent releases of OS X, the focus on traditional interface elements, such as drawers, has generally moved to alternative innovations such as movable palettes and inspectors. In general, there has also been a move towards using sidebars, which now appear in many Apple applications; in addition to more contextual interface elements and full-screen interfaces in many applications.

4.2.2.1.a Jaguar

Jaguar brought with it flatter interface elements, such as new buttons and drop-down menus, as well as reducing the transparency to tone down the pinstripes in windows and menus. These changes would continue from this point forward.

4.2.2.1.b Panther

In Mac OS X Panther, brushed metal was fused to the heart of the Macintosh: the Finder. New buttons were made to appear sunken into their surroundings, following a general trend of more flattened interface elements in the operating system. The traditional pinstripes were replaced with a much subtler "milk" theme, most notably in the menu bar, and the use of transparency was again reduced (for example in the title bars of inactive windows). Tabs also changed; they were made flatter and the whole tab area was sunken rather than raised. Tab buttons were centered on the top border of the tab area. New icons appeared across the system, including a new flatter, glossier Finder icon, a new System Preferences icon.

4.2.2.1.c Tiger

Tiger brought more subtle changes, including the Unified title bar scheme. Pinstripes were now removed from the menu bar entirely, replaced with a new glossy look. Tabs were altered to appear as normal buttons. The Apple menu icon was toned down and the Spotlight search facility now had its own icon permanently bound to the right-hand end of the menu bar.

4.2.2.1.d Leopard

In Leopard, several changes have been made to the user interface. The Dock was made to look more three dimensional, with a reflective “floor” for icons to sit on and icon labels having a semi-transparent background. Active applications are no longer indicated by a triangle, but now by a glowing blue ball. The dividing line between applications and other Dock items now resembles a

pedestrian crossing instead of a simple line. Application windows are reflected off the surface of the dock when close enough. “Stacks” are groups of files, which can be stored in the Dock, and fan out when clicked.

The menu bar at the top of the screen now has the option of being semi-transparent, no longer rounded, and menu highlights are now a blue gradient. This feature is only in the newer Intel Macs, and is not used in PowerPC G4 or even the G5. The corners of menus (including Dock menus) are now rounded; conversely, the corners of the menu bar are not. The Apple menu icon is now a glossy black.

The drop shadow of the active window is now greatly enlarged for emphasis. Inactive windows are less prominent for greater contrast between active and inactive windows. Title bars are a darker shade of grey, and all toolbars now use a darker “Unified” scheme. Brushed metal is no longer present, and has been replaced instead by a white “plastic” gradient scheme. Many windows now have no or minimal borders. Pinstripes in window backgrounds have now been completely removed. Sheets are now semi-transparent as well as blurring the area behind them for greater legibility.

Numerous icons have been changed, including a set of new folder icons, a new System Preferences icon and an updated Terminal icon, and all main icons have been redrawn in a high-resolution 512-by-512 size for sharper viewing in Quick Look and Cover Flow

The default background image has also been changed to a purple aurora superimposed over a star field instead of the previous aqua-blue themes in prior versions.

4.2.2.2 Windows Application

The Aqua theme has also been embedded in applications made by Apple for use in Microsoft Windows such as iTunes, QuickTime, and the Safari web browser. ITunes for Windows, which has the exact same theme as the Mac OS X version, also includes Cover Flow, which is incorporated into Leopard. The Windows version of Safari includes a functional Aqua scrollbar, as well as sheet dialogs very similar to those in Mac OS X.

4.2.2.3 User interface

White and blue are two principal colors, which define the Aqua style. Title bars, window backgrounds, buttons, menus and other interface elements are all found in white, and some, like scrollbars and menu items, are accented with a shade of blue. Most of the interface elements have a "glass" or "gel" effect applied to them; for instance, David Pogue described the original Aqua scrollbars as "lickable globs of Crest Berrylicious Toothpaste Gel".

4.2.2.4 Interface elements

Below, all Mac OS X Cocoa interface elements ('controls') and their NEXTSTEP class name are given. Most of the controls are available in three sizes: regular, small and mini.

4.2.2.5 Windows

Both the standard Aqua-themed pinstriped windows (NSWindow) and the brushed metal windows appear to have the navigational buttons sunken into the window, however in versions of Mac OS X prior to 10.2, the buttons appeared to be on top of the pinstriped windows. Brushed metal windows also have more plastic-like buttons. Mac OS X also allows users to choose a Graphite version instead of a Blue version of the interface. (In Graphite, window controls appear silverish-grey instead of red, yellow, and green.)

Toolbars, defined as NSToolbar, are available in two types: standard or unified. Standard retains the normal Aqua title bar and simply places a row of icons below it, whilst the unified look extends the title bar downwards and places icons on top of it, as if the window has one large title bar.

Sheets, which are modal windows, are also defined as NSWindow. When opened, they are thrusted towards the user like a sheet of paper, hence the name. They are partially transparent and focus attention on the content of the sheet. The parent window's controls are disabled until the sheet is dismissed, but the user is able to continue work in other windows (including those in the same application) whilst the sheet is open.

4.2.2.6 Menus

Menus are backed with a solid gray, and when menu items are highlighted they appear blue. In application menus, which run in a single bar across the top of the screen, keyboard shortcuts appear to the right-hand side of the menu whilst the actual menu item is on the left.

Drop down menus for use in windows themselves (NSPopUpButton) are also available in several varieties. The standard "pop up" menu is white with a blue end cap with opposing arrows, whilst 'pull down' menus only have one downward facing arrow in the end cap. 'Pull down' menus are available four different Aqua varieties, most of which have fallen into disuse with subsequent Mac OS X releases.

4.2.2.7 Text Boxes and Fields

Text boxes are black on white text with a sunken effect border, and are classed as NSTextField. In addition to regular square text boxes, rounded search text boxes are available (NSSearchField). For more extensive text requirements, NSTextView provides a larger, multi-line text field. A combined text box and pull down menu is available, NSComboBox, which allows the user to type in a value in addition to choosing from a menu. NSDatePicker is a combination textbox and picker control, which allows the user to type in a date and time or edit it with directional buttons. NSTokenField was introduced with Mac OS X v10.4, and allows the user to drag non-editable 'tokens' to a text box, between which text can be typed.

4.2.2.8 Push buttons

Standard push buttons with rounded corners are available in two varieties: white and blue. A blue button is the default action, and will appear to "pulse" to prompt the user to carry out that action. The action of a blue button can usually also be invoked with the return key. White buttons are usually associated with all other actions.

Also available are rounded bevel buttons, designed to hold an icon; standard square buttons glass square buttons and round buttons. In addition, circular, purple online help buttons are available which display help relative to the current task when clicked. All types of button are classed as NSButton. Disclosure triangles, although technically buttons, allow views of controls to be shown and hidden to preserve space.

4.2.2.9 Checkboxes and radio buttons

In Mac OS X, empty check boxes are small, white rounded rectangles. When they are checked, they turn blue and a check is present. They are defined as NSButtons; in essence they are buttons, which can be toggled on or off. Radio buttons are similar in appearance and behavior except they are circular and contain a dot instead of a check. They are defined as NSMatrix.

4.2.2.10 Tables and lists

Tables and lists can be broadly categorized in three ways: NSTableView, a standard multi-columnar table with space to enter values or place other interface elements such as buttons; NSOutlineView, which is the same as NSTableView except it can contain disclosure triangles to show and hide sets of data; and NSBrowser, akin to the column view in the Finder (software). All table views can use alternating blue and white row backgrounds.

4.2.2.11 Progress indicators

Two main types of progress indicator are available: a progress bar or a spinning wheel (not the "beach ball" wait cursor). Both are defined as NSProgressIndicator. The progress bar itself is available in two varieties: indeterminate, which simply shows diagonal blue and white stripes in animation with no measure of progress; or determinate, which shows a blue pulsing bar against a white background proportional to the percentage of a task completed. The spinning wheel indicator, also found in the Mac OS X startup screen, is simply a series of lines of various tones arranged in a circle spinning, like the side view of a rotating spiked wheel. Many other interfaces have adopted this device, including the Firefox web browser and many web sites.

4.2.2.12 Miscellaneous

Sliders are available in three types: one with tick marks and a triangular scrubber, one with a round scrubber and no tick marks and a circular slider, which can be rotated. All are defined as NSSlider, and are available horizontally or vertically. The circular slider is simply a gray dot on a white circle which can be rotated to set values.

Mac OS X has a standard control for picking colors, NSColorWell, which appears as a regular square button with a color sample in the middle. When clicked, it shows the standard Mac OS X color palette.

Tab views (NSTabView) in Mac OS X appear to be sunken into the window, and are shaded darker and darker each time a new tab view is added inside another. The tabs appear in a row along the top of the sunken area, and are simply a series of white toggle buttons. The currently selected tab is blue. NSBox is a similar control, used to group interface elements, and uses the same sunken appearance, except without tabs. Image "wells" are also available (NSImageView), a small, sunken container into which image files can be dropped.

4.2.2.13 Fonts

Apple uses the Lucida Grande font as the standard system font in various sizes and weights. Some areas of the operating system use another font, Helvetica. Mac OS X makes use of system-wide font anti-aliasing to make edges appear smoother.

4.2.2.14 Animation

Aqua makes heavy use of animation. Examples include:

- Dock icons bounce up and down as their corresponding applications are launched.
- Dock icons also bounce up and down, in a different rhythm, when a background application requires the user's attention.
- Dock icons increase in size when approached by the cursor. This feature (called "magnification") is optional.
- When minimized, windows are "sucked" into the Dock using the "Genie effect" or "Scale effect." Both of the effects are customizable by the user. The former makes a window turn into a curvy shape so it looks like reverse animation of a genie exiting a lamp, and the latter scales down the window until it is small enough to be in the dock. Using the shift key, both effects can be seen in slow motion. These keystrokes can also be applied

to other Aqua effects such as Dashboard, Exposé and Front Row. There is another undocumented effect called "Suck" which can be enabled by hand editing a configuration file.
- When a folder on the desktop is opened or closed, the corresponding Finder window appears to come from, or disappears into, the folder icon.
- Sheets are "posted" out of Metal, Unified or Leopard window title bars. A dark rectangular slot is drawn on the window so it appears that a dialog box is in fact a sheet of paper being thrust towards the user.
- Dashboard widgets appear with a "ripple" effect, as if being dropped onto the surface of a pond. When removed, Widgets are sucked into the close button as if being drawn into a vacuum.
- The contents of a stack will appear to spring out from behind the icon when clicked.
- In the Public Beta of Mac OS X, docked items dragged on to the desktop simply appeared to 'drop' on to the desktop. This behavior was changed with Mac OS X 10.0; from this release onward items dragged off the dock would simply 'disappear' in a cartoon-like puff of smoke, an effect that is used in various places in the system (such as Safari's Bookmarks Bar) – this is called poof effect.

Many of these effects can be turned off by the user or are only available on supporting hardware.

4.2.2.15 System integration and standardization

There are a series of Mac OS X features, which are standardized across the operating system to make the system more accessible, so the user does not have to learn multiple ways of doing the same thing. Included amongst these features are:

- Services menu - found in the application menu of most applications, which gives the user access to features of other applications
- Palettes - Many palettes are repeated across the system, including:
    - Color - The Mac OS X color picker includes multiple ways of choosing colors, including a color wheel, sliders, a wax crayon view, and a "magnifying glass" to select a color from anywhere on the screen
    - Fonts - The Mac OS X font picker gives the user access to advanced typography features like ligatures and shadows in any program which allows the formatting of text,
    - Character Palette - Found as "Special Characters" in the Edit menus of most applications, allows the user to insert characters they are unable to insert with the keyboard
- Open, Save and Print dialogs - Standard in many applications, and usually use a sheet view

4.2.2.16 Underlying technology

Aqua is powered by the Quartz Compositor, the Mac OS X window server.

4.2.3 Linux's KDE

KDE or the K Desktop Environment is a network transparent contemporary desktop environment for UNIX workstations. KDE seeks to fulfill the need for an easy to use desktop for UNIX workstations, similar to desktop environments found on Macintosh and Microsoft Windows operating systems. The UNIX operating system is according to us the best available today. When it comes to stability, scalability and openness UNIX has no competition. In fact UNIX has been the

undisputed choice of information technology professionals for many years. The lack of an easy to use contemporary desktop environment, however, has prevented UNIX from finding its way onto desktops of typical computer users in offices and homes. UNIX today dominates the server market and is the preferred computing platform for computing professionals and scientists alike. The Internet, a household name traces its heritage to UNIX. Inspite of such ubiquitous creations from the UNIX community, average computer users still expect it to be difficult to use and often stay away. This fact is particularly unfortunate since a number of implementations of UNIX, all of which are of exceptional quality and stability (Debian GNU/Linux, FreeBSD, NetBSD etc.) are freely available of the Internet.

The KDE project aims to change all that. KDE now provides an easy to use contemporary desktop environment available for UNIX and compatible systems. Together with a free implementation of UNIX such as GNU/Linux, UNIX/KDE constitutes a completely free and open computing platform available to anyone free of charge. Source code is available for anyone to look at, learn from, modify and improve. Whilst there is always room for improvement, KDE today delivers a viable feature packed alternative to the more commonly found commercial operating systems/desktops combinations available.

4.2.3.1 History

KDE was founded in 1996 by Matthias Ettrich, who was then a student at the EberhardKarls University of Tübingen. At the time, he was troubled by certain aspects of the Unix desktop. Among his qualms was that none of the applications looked, felt, or worked alike. He proposed the formation of not only a set of applications, but rather a desktop environment, in which users could expect things to look, feel, and work consistently. He also wanted to make this desktop easy to use; one of his complaints with desktop applications of the time was that his girlfriend could not use them. His initial Usenet post spurred a lot of interest, and the KDE project was born. The name KDE was intended as a word play on the existing Common Desktop Environment, available for Unix systems. CDE was an X11-based user environment jointly developed by HP, IBM, and Sun, through the X/Open Company, with an interface and productivity tools based on the Motif graphical widget toolkit. It was supposed to be an intuitively easy-to-use desktop computer environment. The K was originally suggested to stand for "Kool", but it was quickly decided that the K should stand for nothing in particular. Additionally, one of the tips in certain versions of KDE 3 incorrectly states that the K currently is just meant to be the letter before L in the Latin alphabet, the first letter in the word Linux (which is where KDE is usually run).

Matthias chose to use the Qt toolkit for the KDE project. Other programmers quickly started developing KDE/Qt applications, and by early 1997, a few applications were being released. At the time, Qt did not use a free software license and members of the GNU project became concerned about the use of such a toolkit for building a free software desktop and applications. Notably, KDE was removed from Debian because the project interpreted the GPL as not allowing KDE to be linked to Qt. Two projects were started: "Harmony", to create a Free replacement for the Qt libraries, and the GNOME (GNU Network model environment) project to create a new desktop without Qt and built entirely on top of free software as well as trying to solve architectural problems with KDE.

In November 1998, the Qt toolkit was dual-licensed under the free/open source Q Public License (QPL) & commercial-license (proprietary software is required to pay a license fee to Trolltech) The same year, the KDE Free Qt foundation was created which guarantees that Qt would fall under a variant of the very liberal BSD license should Trolltech cease to exist or no free/open source version of Qt be released during 12 months (IN 2008 Trolltech was bought by Nokia, in a movement many people qualify as a movement to attack Motorola, that used Qt as core platform for many Linux-based phones). Debate continued about compatibility with the GNU General Public License (GPL), so in September 2000, Trolltech made the Unix version of the Qt libraries available under the GPL, in addition to the QPL, which eliminated the concerns of the Free Software Foundation. Qt 4.0 is available under the GPL for the Unix, Mac and Windows platforms, enabling KDE 4 applications and libraries to have native support on all these platforms. Still the dual-license problem persists. For example, Motorola now has to pay royalties to Nokia to embed the Qt libraries. For this reason GTK/GNOME libraries continue to be preferred by commercial

companies since they don't have to pay licenses to use it, something normal when targeting major desktop platforms like Microsoft Windows or Mac OS X.

Both KDE and GNOME now participate in freedesktop.org, an effort to standardize Unix desktop interoperability, although there is still some competition between them.

4.2.3.2 Release cycle and version numbers

The KDE team releases new versions on a regular basis. There are two main types of releases, major releases and minor releases.

4.2.3.2.a Major releases

Major releases contain new features. There have been 12 major releases so far — the current major release is 4.0, which arrived on January 11, 2008.

As soon as a major release is ready and announced, work on the next major release starts. A major release needs several months to be finished and many bugs that are fixed during this time are back-ported to the stable branch, meaning that these fixes are incorporated into the last stable release.

4.2.3.2.b X.0 releases

The KDE X.0 releases are special, as they are allowed to break both binary and source-compatibility with the predecessor, or to put it differently, all following releases (X.1, X.2, ...) will guarantee binary (ABI) and source compatibility (API). This means, for instance, that software that was developed for KDE 3.0 will work on all (future) KDE 3 releases, in contrast to an application that was developed for KDE 2, which is not guaranteed to be able to make use of the KDE 3 libraries.

The changes between KDE 1 and KDE 2 series were large and many, while the API changes between KDE 2 and KDE 3 were comparatively minor. There have been major changes between KDE 3 and KDE 4. KDE major version numbers follow the Qt release cycle meaning that KDE 4 is based on Qt 4, while KDE 3 was based on Qt 3.

4.2.3.2.c Minor releases

A minor KDE release has three version numbers, e.g. KDE 1.1.1, and a focus on fixing bugs, minor glitches and making small usability improvements. Minor releases in general do not allow new features although some releases in the 3.5.x line have had minor enhancements. For minor releases, a shortened release schedule is used.

4.2.3.3 Stable Releases

Project announced – 14 October 1996by Matthias Ettrich

KDE 1.0 – 12 July 1998

KDE 1.1 – 6 February 1999

KDE 1.2 was planned, but never released

KDE 2.0 – 23 October 2000

KDE 2.1 – 26 February 2001

KDE 2.2 – 15 August 2001

KDE 3.0 – 3 April 2002

KDE 3.1 – 28 January 2003

KDE 3.2 – 3 February 2004

KDE 3.3 – 19 August 2004

KDE 3.4 – 16 March 2005

KDE 3.5 – 29 November 2005

KDE 4.0 – 11 January 2008

4.2.3.4 KDE 4

KDE 4, the current series of KDE releases, is a major revision of KDE, based on the version 4 series of Qt. KDE 4.0 was released on January 11, 2008.

Some of the new features are:

- A new default look and feel, called Oxygen, and 3D effects in the KWin window manager.
- A completely redesigned desktop and panels collectively called Plasma which integrates Kicker, KDesktop, and SuperKaramba and is intended to update the decades-old desktop metaphor.
- A new multimedia interface (Phonon), making KDE independent of any one specific media framework.
- An API for network and portable devices (Solid)
- A new communication framework (Decibel)
- A metadata and search framework, probably named Tenor. It incorporates Strigi as a full-text file indexing service, and NEPOMUK with KDE integration.
- A new default file manager (Dolphin)
- Microsoft Windows and Mac OS X support by KDE libraries so that KDE applications will more easily be ported to these operating systems.
- A new spell checking program called Sonnet with automatic language detection. It replaces Kspell to mark misspellings in text input fields in KDE-based applications. Advantages over Kspell are the automatic language detection and the ability to work even if several languages are mixed in one document.

4.2.3.5 Architecture

KDE is built with Trolltech's Qt toolkit which runs on most Unix and Unix-like systems, Mac OS X and Microsoft Windows. All releases of KDE 3 are built upon Qt 3, which was only released under the GPL for Linux and Unix-like operating systems, including Mac OS X. For that reason, KDE 3 is only available on Windows through ports involving an X server.

KDE 4 is based on Qt 4 which is also released under the GPL for Windows and Mac OS X. Therefore KDE 4 applications can run natively on these operating systems as well.

4.2.3.6 Usability

KDE aims to make easy-to-use programs without sacrificing features. KDE's Usability page states its goal as: "Working within the existing design goals of a system, usability efforts aim to make the implementations of these designs easier to use, faster to learn, more consistent and obvious, and generally more ergonomic for their target audience." To improve the user interface, work has gone into reducing visual complexity for versions 3.2 to 3.5. The most promising effort is the close work with the OpenUsability Project. One of the major goals of KDE 4.0 is to identify further areas that are lacking from a usability perspective and address these concerns. In particular, new human interface guidelines are being developed for KDE 4.0.

KDE strives to make otherwise onerous or difficult tasks easier, such as adding printers (local or networked), setting up 802.11 Wireless security settings (such as WEP), and installing new fonts and window decorations. Third-party web sites LinuxPrinting, art4linux.org and KDE-Look support KDE through adding devices or customizing the environment's look and feel.

The KDE interface has been criticized for being too complex and including too many configurable options. However, a usability report evaluating a customized version of KDE 3.1 showed, as early as 2003, that Windows users quickly became familiar with KDE, enjoyed it and were able to accomplish the proposed task as quickly as with Windows XP.

4.2.3.7 KDE on Windows and Mac OS X

KDE is now installable on Windows and Mac OS X. Following are the links of the two projects:

Windows Project: http://techbase.kde.org/index.php?title=Projects/KDE_on_Windows

Mac OS X Project: http://techbase.kde.org/index.php?title=Projects/KDE_on_Mac_OS_X

4.2.4 Linux's GNOME

GNOME is an international effort to build a complete desktop environment—the graphical user interface which sits on top of a computer operating system—entirely from free software. This goal includes creating software development frameworks, selecting application software for the desktop, and working on the programs, which manage application launching, file handling, and window and task management.

GNOME is part of the GNU Project and can be used with various Unix-like operating systems, most notably Linux, and as part of Java Desktop System in Solaris.

Note: The official pronunciation of the name is IPA: /gəˈnoʊm/, with a hard "G", although /ˈnoʊm/ (as in the English word "gnome", with a silent "G") is also in common usage. The name originally stood for GNU Network Object Model Environment, though this acronym is deprecated.

4.2.4.1 History

In 1996, the KDE project was started. Although KDE was free software, it relied on the then non-free Qt widget toolkit. Members of the GNU project became concerned with the use of such a toolkit for building a free software desktop environment. In August 1997, two projects were started in response to KDE: the Harmony toolkit (a free replacement for the Qt libraries) and GNOME (a different desktop without Qt and built entirely on top of free software). The initial project leaders for GNOME were Miguel de Icaza and Federico Mena.

In place of the Qt toolkit, GTK+ was chosen as the base of the GNOME desktop. GTK+ uses the Lesser General Public License (LGPL), a free software license that allows GPL-incompatible software (including proprietary software) to link to it. The GNOME desktop itself is licensed under the LGPL for its libraries, and the GPL for applications that are part of the GNOME project itself. Having the toolkit and libraries under the LGPL allows applications written for GNOME to use a much wider set of licenses (including proprietary software licenses). While Qt is dual-licensed under both the QPL and the GPL, the freedom to link proprietary software with GTK+ at no charge makes it differ from Qt.

The name "GNOME" was proposed as an acronym of GNU Network Object Model Environment by Elliot Lee, one of the authors of ORBit and the Object Activation Framework. It refers to GNOME's original intention of creating a distributed object framework similar to Microsoft's OLE. This no longer reflects the core vision of the GNOME project, and the full expansion of the name is now considered obsolete. As such, some members of the project advocate dropping the acronym and re-naming "GNOME" to "Gnome".

4.2.4.2 Look and feel

GNOME is designed around the traditional computing desktop metaphor. Its handling of windows, applications and files is similar to that of contemporary desktop operating systems. In its default configuration, the desktop has a launcher menu for quick access to installed programs and file locations; open windows may be accessed by a taskbar along the bottom of the screen and the top-right corner features a notification area for programs to display notices while running in the background. However these features can be moved to almost anywhere the user desires, replaced with other functions or removed altogether.

The appearance of GNOME can be changed by the use of themes, which are sets consisting of an icon set, window manager border and GTK+ theme engine and parameters. Popular GTK+ themes include Bluecurve and Clearlooks (the current default theme).

GNOME puts emphasis on being easy for everyone to use. The HIG helps guide developers in producing applications which look and behave similarly, in order to provide a cohesive GNOME interface.

NOTE: HIG (Human Interface Guideline) This document tells you how to create applications that look right, behave properly, and fit into the GNOME user interface as a whole. It is written for interface designers, graphic artists and software developers who will be creating software for the GNOME environment. Both specific advice on making effective use of interface elements, and the philosophy and general design principles behind the GNOME interface are covered. Full guideline can be found at http://library.gnome.org/devel/hig-book/stable/ last accessed: March 20, 2008.

4.2.4.3 Usability

Since GNOME v2.0, a key focus of the project has been usability. As a part of this, the GNOME Human Interface Guidelines (HIG) were created, which is an extensive guide for creating quality, consistent and usable GUI programs, covering everything from GUI design to recommended pixel-based layout of widgets.

During the v2.0 rewrite, many settings were deemed to be of little or no value to the majority of users and were removed. For instance, the preferences section of the Panel were reduced from a dialog of six tabs to one with two tabs. Havoc Pennington summarized the usability work in his 2002 essay "Free Software UI", emphasizing the idea that all preferences have a cost, and it's better to "unbreak the software" than to add a UI preference to do that:

"*A traditional free software application is configurable so that it has the union of all features anyone's ever seen in any equivalent application on any other historical platform. Or even configurable to be the union of all applications that anyone's ever seen on any historical platform (Emacs *cough*).*

Does this hurt anything? Yes it does. It turns out that preferences have a cost. Of course, some preferences also have important benefits - and can be crucial interface features. But each one has a price, and you have to carefully consider its value. Many users and developers don't understand this, and end up with a lot of cost and little value for their preferences dollar."

Some people believe that GNOME should be more functional. One of these is LinusTorvalds, creator of the Linux kernel, who commented in a usability-related discussion on the GNOME usability mailing list:

" *This "users are idiots, and are confused by functionality" mentality of Gnome is a disease. If you think your users are idiots, only idiots will use it. I don't use Gnome, because in striving to be simple, it has long since reached the point where it simply doesn't do what I need it to do. Please, just tell people to use KDE*."

4.2.4.4 Stable Releases and Version Numbers

Following are the released versions of GNOME.

0.0 (August 1997) GNOME development announced

1.0 (March 1999) First major GNOME release

1.0.53 (October 1999) "October"

1.2 (May 2000) "Bongo"

1.4 (April 2001) "Tranquility"

2.0 (June 2002) Major upgrade based on GTK2. Introduction of the Human Interface Guidelines.

2.2 (February 2003) Multimedia and file manager improvements.

2.4 (September 2003) "Temujin": Epiphany, accessibility support.

2.6 (March 2004) Nautilus changes to a spatial file manager, and a new GTK+ file dialog is introduced. A short-lived fork of GNOME, GoneME, is created as a response to the changes in this version.

2.8 (September 2004) Improved removable device support, adds Evolution.

2.10 (March 2005) Lower memory requirements and performance improvements. Adds: new panel applets (modem control, drive mounter and trashcan); and the Totem and Sound Juicer applications

2.12 (September 2005) Nautilus improvements; improvements in cut/paste between applications and freedesktop.org integration. Adds: Evince PDF viewer; New default theme: Clearlooks; menu editor; keyring manager and admin tools. Based on GTK+ 2.8 with Cairo support.

2.14(March 2006)Performance improvements (over 100% in some cases); usability improvements in user preferences; GStreamer 0.10 multimedia framework. Adds: Ekiga video conferencing application; Deskbar search tool; Pessulus lockdown editor; Fast user switching; Sabayon system administration tool.

2.16 (September 2006) Performance improvements. Adds: Tomboy notetaking application; Baobab disk usage analyzer; Orca screen reader; GNOME Power Manager (improving laptop battery life); improvements to Totem, Nautilus; compositing support for Metacity; new icon theme. Based on GTK+ 2.10 with new print dialog.

2.18 (March 2007) Performance improvements. Adds: Seahorse GPG security application, allowing encryption of emails and local files; Baobab disk usage analyzer improved to support ring chart view; Orca screen reader; improvements to Evince, Epiphany and GNOME Power Manager, Volume control; two new games, GNOME Sudoku and glchess. MP3 and AAC audio encoding.

2.20 (September 2007)Tenth anniversary release.Evolution backup functionality; improvements in Epiphany, EOG, GNOME Power Manager; password keyring management in Seahorse. Adds: PDF forms editing in Evince; integrated search in the file manager dialogs; automatic multimedia codec installer.

2.22 (March 2008) Addition of Cheese, a tool for taking photos from webcams and Remote Desktop Viewer; basic window compositing support in Metacity; introduction of GVFS; improved playback support for DVDs and Youtube, MythTV support in Totem; internationalised clock applet; Google Calendar support and message tagging in Evolution; improvements in Evince, Tomboy, Sound Juicer and Calculator.

4.2.5 Linux's Xfce

Xfce is a lightweight desktop environment for Unix-like operating systems. It aims to be fast and lightweight, while still being visually appealing and easy to use.

Xfce 4.4 embodies the traditional UNIX philosophy of modularity and re-usability. It consists of a number of components that together provide the full functionality of the desktop environment. They are packaged separately and you can pick and choose from the available packages to create the best personal working environment.

Another priority of Xfce 4 is adherence to standards, specifically those defined at freedesktop.org.Xfce is somewhat similar to the commercial CDE, but has been getting a little farther away from that comparison with each new major version.

It is based on the GTK+ 2 toolkit (like GNOME). It uses the Xfwm window manager. Its configuration is entirely mouse-driven, and the configuration files are hidden from the casual user.

Xfce 4 can be installed on several UNIX platforms. It is known to compile on Linux, NetBSD, FreeBSD, OpenBSD, Solaris, Cygwin and MacOS X, on x86, PPC, SPARC, Alpha.

4.2.5.1 History

Olivier Fourdan started the project in 1996. The name "Xfce" originally stood for "XForms Common Environment", but since that time Xfce has been rewritten twice and no longer uses that toolkit. The name survived, but it is no longer capitalized as "XFce", but rather as "Xfce". The developers' current stance is that the initialism does not stand for anything any more.

4.2.5.1.a First versions

Xfce began as a simple project created with XForms, meant to be a free Linux clone of CDE. The program, a simple toolbar, was released by Fourdan to ibiblio (then "SunSITE"), and the community showed an impressive demand for expansion of the project.

4.2.5.1.b Version 2

Fourdan continued developing the project, and in 1998, Xfce 2 was released with the first version of Xfce's window manager, Xfwm. He requested to have the project included in Red Hat Linux, but was refused because the project was based on XForms. Red Hat only accepted software that was free and open source, but, at the time, XForms was closed source and free only for personal use. For the same reason, Xfce was not in Debian before version 3. Xfce 2 was only distributed in Debian'scontrib component.

4.2.5.1.c Version 3

XForms, the proprietary library that Xfce was based on, was limiting the progress of the project. The popularity of the GTK+ toolkit was increasing, and Fourdan saw it as a fitting replacement. In March of 1999, he scrapped the old XFce and began a complete rewrite of the project based on GTK+. The result was Xfce 3.0, which was licensed under the GNU GPL. Along with being based completely on open-source software, the project gained many benefits from using the GTK+ libraries, including drag-and-drop support, native language support, and improved configurability. Xfce was uploaded to SourceForge.net in February of 2001, starting with version 3.8.1.

4.2.5.1.d Version 4

Xfce made a major jump in version 4.0.0 - it was upgraded to use the GTK+ 2 libraries. There were many other major changes made in this series as well, including a compositing manager for Xfwm in 4.2.0 which added built-in support for transparency and drop shadows, as well as a new default SVG icon set. In January of 2007, Xfce 4.4.0 was released. A notable feature of this release was the inclusion of the Thunar file manager, a replacement for the aging Xffm. This release also featured improvements to the "eye candy" aspect of the desktop: Xfwm gained an enhanced compositor, and support for desktop icons was added. Also, various improvements were made to the panel to prevent buggy plugins from crashing the whole panel.

4.2.5.2 Xfwm

The window manager of Xfce, Xfwm, is unique in that starting with version 4.2 it integrates its own compositing manager. Other compositing managers exist, but have been rather unstable, and Xfce was the first to integrate its own compositing manager into the window manager. At its inception, many users called it the most stable one available, though at the time, in late 2004, xcompmgr was the only other compositing manager available.

4.2.5.3 Thunar

Thunar is the default file manager for Xfce, replacing the earlier Xffm. It resembles Nautilus and is designed for speed and a low memory footprint as well as being highly customizable through plugins. Xfce also has a lightweight archive manager called Xarchiver, but this is not part of the core Xfce 4.4.0. More recently, Squeeze has been started as an archive manager designed to integrate better into the Xfce desktop.

4.3 Users View of Operating System

This document is based on general users reply on three community forums. They had to comment on "how they feel about operating system". They were supposed to comment on few points like

a. Knowledge about OS
b. Expectation from OS
c. What they use
d. Why they like their current OS
e. Why they do not like their current OS
f. About other OSs.

Those 3 forums were – Ubuntu Forums, JCXP Forums and TunesBD forums.

NOTE: The user comments are moderated, edited to make it more presentable.

It is clear that everyone is aware of the fact that operating system (OS) is a software and they need it to use their computer. Most of the users really do not care about how their operating system works. They are happy as long as they can use their computers for various purposes.

4.3.1 General Ideas of Operating System

"Yes I know what my operating system is: Windows Vista. I need an operating system so my computer can work..." – jarGS (ubuntuforums)

"Most members of this forum use Linux bases systems […]" – LaRozza (ubuntuforums)

From the above comments, we can see that users at least know their operating systems name. The most prominent reason behind this is advertisement. Most of the people are familiar with Windows and MacOS just because of their media publicity. Still most of the users really do not care what they are using. They are happy as long as they can use their computer for their purposes. Consider the next comment.

"No, I don't really. […]. From where does the OS come in, I have no idea. Don't care either, I like Ubuntu a whole lot better than other OSs." – Vadi (ubuntuforums)

This user does not know what operating system is and he does not care. Still he uses something and the possible reason is he feels comfortable with it. All users are not of same level of expertise, there

are some users who things about their operating system. They do not need to know its actually mechanism but they have simple ideas about the OS.

"I've only got a vague idea of what the OS does: interacts with hardware, processes info, whatever. For the most part, the operating system was a qualifying dimension, and the programs it could run were the differentiating factors." – DarkOx (ubuntuforums)

There is a common misconception that Linux users are power user. Notice the comment above; this user uses Linux regularly, not knowing what operating system is and how it works.

Again take a look at this,

"I always thought the OS was the entire package. The kernel, the interface and everything else. I know Linux is just the kernel and Ubuntu is the user interface built around the kernel, but I would say Ubuntu is an OS, like Windows VISTA and Mac OSX are considered OSs because they are complete packages." – tuebinger (ubuntuforums)

What this user thinks is right (not considering formal bookish OS definition). Now, OS means a collection of packages. Kernel, drivers, framework libraries, desktop environment and few applications come in OS distribution package.

Users do not need to about the operating system definition and mechanism. They are the consumers of a product. We do not need to learn about the mechanism of a refrigerator to use it. The point of the study was to find out the knowledge of the general users and it seems that more or less intermediate users know at least some things about operating systems. This class of users is important for OS development, they are likely to be helpful when it comes for asking feedback.

Those who fall under totally beginner class, their experience depends totally on how fast they can do their job. They do not know and do not want to know any technical details of anything. Ultimately, if Linux can be promoted in better way, this class of users will accept Linux happily.

4.3.2 Technical Ideas of Operating System

As expected on Ubuntu Forums, some users replied with technical details, though I have not asked for it. Most of Linux users know about kernel, at least they heard about it. Unlike other operating systems, Linux kernel has a tendency to release updated versions more often; this is the one of the reasons that users heard about the kernel.

"It is difficult to say what the operating system is. An operating system should basically be the kernel and drivers, and then some way to interact with that, like a command line or desktop environment. But operating systems often include extraneous stuff, libraries, and application such as movie players, simple games." – original_jamingrit (ubuntuforums)

What he knows/thinks is not absolutely right but as a general user, he at least knows that OS has kernel, drivers, libraries and extra applications. When users begin to realize that his operating system must include a kernel, this and that driver and some free application, their experience differs on the number and quality of the features. This class of users usually compares different operating systems if they need to switch into new one.

Another user gave his comment theoretically,

"An Operating System is a system software, that is used for:

- *Process Management*
- *I/O Management*
- *Disk Management*
- *File Management*

Of course to facilitate the use of other applications to execute so that every application developer doesn't have to be concerned with the minute details of the workings of the hardware involved.

An application programming interface (API) and a hardware abstraction layer are its other properties.

In simple term, it is only evident that in order to use a computer that consists of various types of hardware, one manager is needed to control them. That is the OS." – sujoy (ubuntuforums)

He went into too deep topics I did not expected. He is right, operating system is the manager to control things.

4.3.3 I use Linux. Why?

Consider next two comments:

"The operating system must make certain basic requirements before I'll even consider it. I must be able to install it easily, and be able to easily install additional programs once it's running. It must be reasonably fast, and offered at a reasonable cost. It must be compatible with my hardware.

Once that bar is met (and most OS's do meet it), I'll choose which one to use based on the programs I need to get my work done, or game, or whatever it is I want to use a computer for. I tend to prefer Linux's programs to Windows programs and when I don't, the gap between them isn't very great (I'm thinking of Open Office here). So I use Linux." - DarkOx

"I am a Linux user because of a number of reasons, all of which are equally responsible for my using it.

- Open Source philosophy
- Freedom to customize as needed
- Security
- Availability of Support
- Ease of use
- Large number of distros and apps available

I believe it is (Linux in general) better than the rest of the lot because it doesn't restrict my ability to interact and experiment with the system, while also safeguarding me from damaging it." – sujoy (ubuntuforums)

Those who really care for their operating system, they tend to choose from available distributions. Which is of course a good sign. Most common points of using Linux are,

1. Free (as in beer and freedom)
2. Open Source
3. Fast
4. Choices for fit the needs
5. Hardware support (drivers)
6. Stability
7. Security
8. Community (technical support)
9. Easy to use (Installation + operation)
10. Power of customization (Desktop and other features)

Most of the Linux distributions are free, and this is the most important reason Linux is replacing other desktop operating system. Some users are aware of Open Source philosophy, but most of them really do not know what it is.

Linux is fast with little requirement of hardware. Unlike other operating systems, there are lightweight versions of Linux distributions to that users with older hardware can use the latest technology on their computer.

Hardware support was Linux's primary problem but on last 2 years the scenario is changing. Most of the hardware vendors are now providing drivers for their hardware. Another problem of Linux was installation procedure. All of the Linux distro had "text-based" installation procedure. Most of the computer users are scared of this procedure. From last 3 years almost all the distribution are giving the GUI installation method along with previous method. This major change of options pulled a lot of users. Not only the installation method, the overall windowing system, aesthetics, user notifications and overall GUI changed on last few years. Now in case of ease of use, there is almost no difference between Windows and Linux.

Linux was and still is famous for its stability. Windows has a tendency to crash too often and that makes most of the "advanced" users unhappy. Security comes with stability. More stable means users are more secured.

Another most common points why users like Linux is the power of customization. Most of the users like to customize their desktop environment, as they like. Where in other operating system they had to use some non-free tools to do it, on Linux they can do it without anything else.

A nice comment about Linux:

"I use Ubuntu because it requires almost no maintenance. I think many are 'stuck' with Windows because most people don't want an OS so they don't look for an OS. They want a computer and applications so they look for that and end up with the OS that was pre installed on the computer when they got it. People may think "I want a great image editor that I can use to do lots of cool stuff with my photos." but non-geeks usually don't think, "I want an OS with more efficient better memory management, hardware drivers, filesystem etc.". The ones who actually know of and consider alternatives but discard them due to misconceptions like "Linux is hard" are more computer literate than most." – Kvark (ubuntuforums)

4.3.4 Why people are stuck with Windows?

I won't discuss about this topic right now, these comments are enough self-explanatory.

"People are stuck with Windows simply because it was the OS that was installed on their computer. And because they know no different, they aren't aware of alternatives. Even if they are they are unlikely to be confident enough to install an alternative. Or if they could, while the OS may not be perfect, it does enough for them to not consider it worthwhile jumping from the unknown to the unknown. It's what they have become familiar with and the thought of trying something different (when they don't actually have to) seems a bit too much." – AndyCooll (ubuntuforums)

"First, people need to know about the other system and why they would want to switch. The new system would have to work with the dominant system (Windows) pretty much seamlessly, since by definition most people use the dominant system, and people need to work together. The new system would have to be more than just better. It would have to be so much better that one could justify learning to accomplish the same tasks differently. This can be a very significant barrier, even if the different way is better." – jpkotta (ubuntuforums)

One point no user mentioned is "virus". There are a lot of users, who moved to Linux's just because of viruses. Linux has comparatively less (almost no) virus than Windows.

4.3.5 What users want from a new operating system

A JCXP member (Nickname: shrimp) replied like this

Q. *"When a new version of any operating system is coming up. What do you expect from it?"*

A. Superior features and performance.

Q. *"Do you really care about which OS you are using or as long as your computer is working you are fine?"*

A. Yes, If there's something better than what I'm using I'll switch.

Q. *"Consider you use one operating system, why would you or why wouldn't you move to some other OS?"*

A. I would move if:

- If it is better overall than my current OS.

I wouldn't move if:

- The performance and features weren't as good as my other OS.
- If there was no reason to.
- If it was extremely hard to use.

This is obvious that he is at least advanced level user.

In answer of *"When a new version of any operating system is coming up. What do you expect from it?"* another user Bazook (new member of JCXP) replied with this screenshot.

This is not a real OS, but the concept is really catchy. Recent tendency of choosing OS is "being minimalistic". Most of the users know that OS is the media who lets us use the hardware and software. On last few years operating systems are becoming more and more resource hungry - which nobody likes. All the users are looking for an OS, which is efficient, which would have nice looking interfaces, which is stable and which should be cheap.

Another important issue is – gaming platform. Windows has been a famous gaming platform for a long time. In case of OS, Windows is dominating in this place. Most of the users want something with UNIX power but on the same time would be able to run DirectX gaming platform.

There are other issues, like game designers, movie special effects designers, graphics designers; music producers would never use Linux/Unix. The reason is lack of software.

Here is a funny comment,

"I want an OS to be, fast, light on RAM, a nice GUI, does not need tons worth of patching, isn't to prone to viruses.

Well everything above matches OSX, so I'll just get a Mac.

But I need brilliant OpenGL and DirectX support so I can game." – Watchthisspace (JCXP Forums)

Which is true. All the Mac switchers have more or less same reason.

Another suggestion, which users are asking from Microsoft recently, that is "full gaming mode" operating system. Rumor says it was in Vista's feature list but they didn't implement it finally. Windows 7 is supposed to have this feature.If we summarize the expected feature list, that would be

1. Must be able to install it easily
2. Able to install additional software easily
3. Fast [with less RAM] / Lightweight
4. Low cost
5. Hardware (driver) support
6. Efficient / stable / high performance
7. Eye-candy / improved GUI

In the end – we can summarize that, the ideas of operating system differs by the level of expertise but still everyone's expectation is same. All of them want smooth computing. There are two categories of users now, one category doesn't want any change and the other is ready to accept the better ones. A new operating system has to be fast, lightweight and stable with better aesthetics to be acceptable to all level of users.

4.4 Key factors of experience

- Installer media
- Installation method
- Hardware requirements
- Desktop Environments
- Usability
- Consistency
- Driver support
- Installed applications
- Application availability
- Application packaging system
- System Settings/Configuration/Maintenance
- Support/Community

4.4 Usage of UX in OS development (current trends)

4.4.1 Canonical

Comparing to other software companies like Apple and Microsoft, Canonical is a young developer. Canonical, is the commercial sponsor for ubuntu. Within only 3 years, it became number one linux distributor after releasing ubuntu 7.10. They are focusing on user experience a lot. Now they are hiring user experience experts for ubuntu development. As ubuntu releases in 6 month periods, we can see a lot of changes in each version. The best part of the development is, the developers take ideas and suggestions from the end users directly. The ubuntu community (largest operating system user's community, www.ubuntuforums.org) and the launchpad (www.launchpad.net) helps the developers to gather ideas for next ubuntu development. So, unlike other developers – ubuntu is more like user and developer developed operating system. If we compare the first release and the latest release we can see the users experience changes clearly.

About Canonical

Canonical, the commercial sponsor of Ubuntu, is headquartered in Europe and is committed to the development, distribution and support of open source software products and communities. World-class 24x7 commercial support for Ubuntu is available through Canonical's global support team and partners.

Since its launch in October 2004 Ubuntu has become one of the most highly regarded Linux distributions with millions of users around the world. Ubuntu will always be free to download, free to use and free to distribute to others. With these goals in mind, Ubuntu aims to be the most widely used Linux system, and is the centre of a global open source software ecosystem.

4.4.2 Apple

The user experience for Mac OS X applications encompasses the visual appearance, interactive behavior, and assistive capabilities of software. With the Aqua graphical user interface, Universal Access features, and user-assistive technologies like the Address Book framework, Apple Help, and VoiceOver, you can deliver the cohesive and professional user experience that Macintosh users have come to expect. It's easy to leverage the user experience technologies of Mac OS X to make great Macintosh software.

The most visible expression of Mac OS X's integration and power is its Aqua user interface. Aqua incorporates the visual appearance of icons, menus, windows and controls with high-quality graphics and user-centric design to produce a user experience that is both functional and appealing. Consistent with Apple's design philosophy, visual enhancements such as color, transparency and animation serve not just as beautiful images, but as cues to the functionality and operation of the system and its applications. Fast User Switching is an example where Aqua uses the sophisticated graphics system in Mac OS X to provide strong visual cues and user feedback.

4.4.3 Microsoft

According to Microsoft, building a modern and compelling application is challenging. Visuals need to be more impressive and functionality needs to be more complex. Having a differentiated experience that attracts and retains customers is more important than ever. The technical challenge of creating such an application is significant. On one hand, you need a technology that is powerful enough to support your visual goals, be it 3-D, complex animations, or enhanced typography. On the other hand, that same technology needs to enable your application logic. Windows Presentation Foundation (WPF), part of the WinFX offerings, is the foundation for the next generation of Windows applications and content.

WPF enables developers to achieve a new level of quality in the "User Experience" (UX) aspect of applications. Developers will be able to use WPF to build applications similar to current applications, but they will also use WPF to create and display media-rich interactive content, animations, and traditional documents. WPF is not just about graphics, its coverage of all common forms of presentation represents a degree of unification that is new to the Windows platform. The result is an unprecedented ability to combine controls, content, and vector graphics in a seamless manner. WPF also has features intended to streamline the development process, by reducing the amount of procedural code in application specification, and by enabling a greater and more direct collaboration by members of the development team.

4.4.4 IBM

User Experience Design fully encompasses traditional Human-Computer Interaction (HCI) design and extends it by addressing all aspects of a product or service as perceived by users. HCI design addresses the interaction between a human and a computer. In addition, User Experience Design addresses the user's initial awareness, discovery, ordering, fulfillment, installation, service, support, upgrades, and end-of-life activities. HCI design constitutes a major portion of the activities performed by a User Experience Design team, so the following paragraphs provide an overview of HCI design followed by references to additional material about User Experience Design. Human-Computer Interaction, or HCI, is the study, planning, and design of what happens when you and a computer work together. As its name implies, HCI consists of three parts: you the user, the computer itself, and the ways you work together.

The User Sciences & Experience Research (USER) lab focuses on understanding and improving how people interact with technology. Our goal is to improve the ease-of-use of existing products and explore new paradigms in using computers.

Chapter 5: UX Research

5.1 Factors

5.1.1 Human Computer Interaction (HCI)

There is currently no agreed upon definition of the range of topics, which form the area of human-computer interaction. Yet we need a characterization of the field if we are to derive and develop educational materials for it. Therefore we offer a working definition that at least permits us to get down to the practical work of deciding what is to be taught:

Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.

From a computer science perspective, the focus is on interaction and specifically on interaction between one or more humans and one or more computational machines. The classical situation that comes to mind is a person using an interactive graphics program on a workstation. But it is clear that varying what is meant by interaction, human, and machine leads to a rich space of possible topics, some of which, while we might not wish to exclude them as part of human-computer interaction, we would, nevertheless, wish to identify as peripheral to its focus. Other topics we would wish to identify as more central.

Take the notion of machine. Instead of workstations, computers may be in the form of embedded computational machines, such as parts of spacecraft cockpits or microwave ovens. Because the techniques for designing these interfaces bear so much relationship to the techniques for designing workstations interfaces, they can be profitably treated together. But if we weaken the computational and interaction aspects more and treat the design of machines that are mechanical and passive, such as the design of a hammer, we are clearly on the margins, and generally the relationships between humans and hammers would not considered part of human-computer interaction. Such relationships clearly would be part of general human factors, which studies the human aspects of all designed devices, but not the mechanisms of these devices. Human-computer interaction, by contrast, studies both the mechanism side and the human side, but of a narrower class of devices.

Or consider what is meant by the notion human. If we allow the human to be a group of humans or an organization, we may consider interfaces for distributed systems, computer-aided communications between humans, or the nature of the work being cooperatively performed by means of the system. These are all generally regarded as important topics central within the sphere of human-computer interaction studies. If we go further down this path to consider job design from the point of view of the nature of the work and the nature of human satisfaction, then computers will only occasionally occur (when they are useful for these ends or when they interfere with these ends) and human-computer interaction is only one supporting area among others.

There are other disciplinary points of view that would place the focus of HCI differently than does computer science, just as the focus for a definition of the databases area would be different from a computer science vs. a business perspective. HCI in the large is an interdisciplinary area. It is emerging as a specialty concern within several disciplines, each with different emphases: computer science (application design and engineering of human interfaces), psychology (the application of theories of cognitive processes and the empirical analysis of user behavior), sociology and anthropology (interactions between technology, work, and organization), and industrial design (interactive products). In this report, we have adopted, as an ACM committee, an appropriate computer science point of view, although we have tried at the same time to consider human-computer interaction broadly enough that other disciplines could use our analysis and shift the focus appropriately. From a computer science perspective, other disciplines serve as supporting disciplines, much as physics serves as a supporting discipline for civil

engineering, or as mechanical engineering serves as a supporting discipline for robotics. A lesson learned repeatedly by engineering disciplines is that design problems have a context, and that the overly narrow optimization of one part of a design can be rendered invalid by the broader context of the problem. Even from a direct computer science perspective, therefore, it is advantageous to frame the problem of human-computer interaction broadly enough so as to help students (and practitioners) avoid the classic pitfall of design divorced from the context of the problem.

To give a further rough characterization of human-computer interaction as a field, we list some of its special concerns: Human-computer interaction is concerned with the joint performance of tasks by humans and machines; the structure of communication between human and machine; human capabilities to use machines (including the learn ability of interfaces); algorithms and programming of the interface itself; engineering concerns that arise in designing and building interfaces; the process of specification, design, and implementation of interfaces; and design trade-offs. Human-computer interaction thus has science, engineering, and design aspects.

Regardless of the definition chosen, HCI is clearly to be included as a part of computer science and is as much a part of computer science as it is a part of any other discipline. If, for example, one adopts Newell, Perlis, and Simon's (1967) classic definition of computer science as "the study of computers and the major phenomena that surround them," then the interaction of people and computers and the uses of computers are certainly parts of those phenomena. If, on the other hand, we take the recent ACM (Denning, et al., 1988) report's definition as "the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application," then those algorithmic processes clearly include interaction with users just as they include interaction with other computers over networks. The algorithms of computer graphics, for example, are just those algorithms that give certain experiences to the perceptual apparatus of the human. The design of many modern computer applications inescapably requires the design of some component of the system that interacts with a user. Moreover, this component typically represents more than half a system's lines of code. It is intrinsically necessary to understand how to decide on the functionality a system will have, how to bring this out to the user, how to build the system, how to test the design.

Because human-computer interaction studies a human and a machine in communication, it draws from supporting knowledge on both the machine and the human side. On the machine side, techniques in computer graphics, operating systems, programming languages, and development environments are relevant. On the human side, communication theory, graphic and industrial design disciplines, linguistics, social sciences, cognitive psychology, and human performance are relevant. And, of course, engineering and design methods are relevant.

5.1.2 Information architecture (IA)

Information architecture (IA) is the art and science of expressing a model or concept for information used in library systems, web development, user interactions, database development, programming, technical writing, enterprise architecture, critical system software design and other activities that require explicit details of complex systems. Information architecture has somewhat different meanings in these different branches of IS or IT architecture. Most definitions have common qualities: a structural design of shared environments, methods of organizing and labeling websites, intranets, and online communities, and ways of bringing the principles of design and architecture to the digital landscape.

Information architecture is defined by the Information Architecture Institute as:

1. The structural design of shared information environments.

2. The art and science of organizing and labeling web sites, intranets, online communities and software to support findability and usability.

3. An emerging community of practice focused on bringing principles of design and architecture to the digital landscape.

The term information architecture describes a specialized skill set, which relates to the interpretation of information and expression of distinctions between signs and systems of signs. It has some degree of origin in the library sciences. Many library schools teach information architecture.

In the context of information system design, information architecture refers to the analysis and design of the data stored by information systems, concentrating on entities, their attributes and their interrelationships. It refers to the modeling of data for an individual database and to the corporate data models an enterprise uses to coordinate the definition of data in several (perhaps scores or hundreds) of distinct databases. Recently, the "canonical data model" is applied to integration technologies as a definition for specific data passed between the systems of an enterprise. At a higher level of abstraction, it may also refer to the definition of data stores.

5.1.3 Interaction Design (IxD)

Interaction Design (IxD) is the discipline of defining the behavior of products and systems that a user can interact with. The practice typically centers on complex technology systems such as software, mobile devices, and other electronic devices. However, it can also apply to other types of products and services, and even organizations themselves. Interaction design defines the behavior (the "interaction") of an artifact or system in response to its users.

Certain basic principles of cognitive psychology provide grounding for interaction design. These include mental models, mapping, interface metaphors, and affordances. Many of these are laid out in Donald Norman's influential book The Design of Everyday Things. Academic research in Human Computer Interaction (HCI) includes methods for describing and testing the usability of interacting with an interface, such as cognitive dimensions and the cognitive walkthrough.

Interaction designers are typically informed through iterative cycles of user research. They design with an emphasis on user goals and experience, and evaluate designs in terms of usability and affective influence.

5.1.4 User centered interaction design

As new technologies are often overly complex for their intended target audience, interaction design aims to minimize the learning curve and to increase accuracy and efficiency of a task without diminishing usefulness. The objective is to reduce frustration and increase user productivity and satisfaction.

Interaction design attempts to improve the usability and experience of the product, by first researching and understanding certain users' needs and then designing to meet and exceed them. (Figuring out who needs to use it, and how those people would like to use it.)

Only by involving users who will use a product or system on a regular basis will designers be able to properly tailor and maximize usability. Involving real users, designers gain the ability to better understand user goals and experiences. There are also positive side effects, which include enhanced system capability awareness and user ownership. It is important that the user be aware of system capabilities from an early stage so that expectations regarding functionality are both realistic and properly understood. Also, user's who have been active participants in a product's development are more likely to feel a sense of ownership, thus increasing overall satisfaction.

5.1.5 Relationship with user interface design

Interaction Design is often associated with the design of system interfaces in a variety of media (see also: Interface design, Experience design) but concentrates on the aspects of the interface that define and present its behavior over time, with a focus on developing the system to respond

to the user's experience and not the other way around. The system interface can be thought of as the artifact (whether visual or other sensory) that represents an offering's designed interactions. Interactive voice response (Telephone User Interface) is an example of interaction design without graphical user interface as a media.

Interactivity, however, is not limited to technological systems. People have been interacting with each other as long as humans have been a species. Therefore, interaction design can be applied to the development of all solutions (or offerings), such as services and events. Those who design these offerings have, typically, performed interaction design inherently without naming it as such.

5.1.6 General steps in interaction design

There is a general process that most interaction designers follow, the point of which is to create a solution (not the solution) to a known problem. A key element in this process is the idea of iteration, where the aim is to build quick prototypes and test them with the users to make sure the proposed solution is satisfactory.

These are the major steps in interaction design. Based on user feedback, several iteration cycles of any set of steps may occur.

1. Design research

Using design research techniques (observations, interviews, questionnaires, and related activities) designers investigate users and their environment in order to learn more about them and thus be better able to design for them.

2. Research analysis and concept generation

Drawing on a combination of user research, technological possibilities, and business opportunities, designers create concepts for new software, products, services, or systems. This process may involve multiple rounds of brainstorming, discussion, and refinement.

To help designers realize user requirements, they may use tools such as personas or user profiles that are reflective of their targeted user group. From these personae, and the patterns of behavior observed in the research, designers create scenarios (or user stories) or storyboards, which imagine a future workflow the users will go through using the product or service.

After thorough analysis using various tools and models, designers create a high level summary spanning across all levels of user requirements. This includes a vision statement regarding the current and future goals of a project.

3. Alternative design and evaluation

Once clear view of the problem space exists, designers will develop alternative solutions with crude prototypes to help convey concepts and ideas. Proposed solutions are evaluated and perhaps even merged. The end result should be a design that solves as many of the user requirements as possible.

Some tools that may be used for this process are wire-framing and flow diagrams. The features and functionality of a product or service are often outlined in a document known as a wireframe ("schematics" is an alternate term). Wireframes are a page-by-page or screen-by-screen detail of the system, which include notes ("annotations") as to how the system will operate. Flow Diagrams outline the logic and steps of the system or an individual feature.

4. Prototyping and usability testing

Interaction designers use a variety of prototyping techniques to test aspects of design ideas. These can be roughly divided into three classes: those that test the role of an artifact, those that test its look and feel and those that test its implementation. Sometimes, these are called

experience prototypes to emphasize their interactive nature. Prototypes can be physical or digital, high- or low-fidelity.

5. Implementation

Interaction designers need to be involved during the development of the product or service to ensure that what was designed is implemented correctly. Often, changes need to be made during the building process, and interaction designers should be involved with any of the on-the-fly modifications to the design.

6. System testing

Once the system is built, often-another round of testing, for both usability and errors ("bug catching") is performed. Ideally, the designer will be involved here as well, to make any modifications to the system that are required.

5.1.7 User interface design (UI Design)

User interface design or user interface engineering is the design of computers, appliances, machines, mobile communication devices, software applications, and websites with the focus on the user's experience and interaction. Where traditional graphic design seeks to make the object or application physically attractive, the goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals—what is often called user-centered design. Where good graphic/industrial design is bold and eye catching, good user interface design is to facilitate finishing the task at hand over drawing attention to itself. Graphic design may be utilized to apply a theme or style to the interface without compromising its usability. The design process of an interface must balance the meaning of its visual elements that conform the mental model of operation, and the functionality from a technical engineering perspective, in order to create a system that is both usable and easy to adapt to the changing user needs.

User Interface design is involved in a wide range of projects from computer systems, to cars, to commercial planes; all of these projects involve much of the same basic human interaction yet also require some unique skills and knowledge. As a result, user interface designers tend to specialize in certain types of projects and have skills centered on their expertise, whether that be software design, user research, web design, or industrial design.

5.1.8 Processes

There are several phases and processes in the user interface design some of which are more demanded upon than others depending on the project. (note for the remainder of this section the word system is used to denote any project whether it is a web site, application, or device)

- Functionality requirements gathering - assembling a list of the functionality required of the system to accomplish the goals of the project and the potential needs of the users.
- User analysis - analysis of the potential users of the system either through discussion with people who works with the users and/or the potential users themselves. Typical questions involve:
    - What would the user want the system to do?
    - How would the system fit in with the user's normal workflow or daily activities?
    - How technically savvy is the user and what similar systems does the user already use?
    - What interface look & feel styles appeal to the user?

- Information architecture - development of the process and/or information flow of the system (i.e. for phone tree systems, this would be an option tree flowchart and for web sites this would be a site flow that shows the hierarchy of the pages).
- Prototyping - development of wireframes, either in the form of paper prototypes or simple interactive screens. These prototypes are stripped of all look & feel elements and most content in order to concentrate on the interface.
- Usability testing - testing of the prototypes on an actual user—often using a technique called talk aloud protocol where you ask the user to talk about their thoughts during the experience.
- Graphic Interface design - actual looks & feels design of the final graphical user interface (GUI.) It may be based on the findings developed during the usability testing if usability is unpredictable, or based on communication objectives and styles that would appeal to the user. In rare cases, the graphics may drive the prototyping, depending on the importance of visual form versus function. If the interface requires multiple skins, there may be multiple interface designs for one control panel, functional feature or widget. This phase is often a collaborative effort between a graphic designer and a user interface designer, or handled by one who is proficient in both disciplines.

User interface design needs good understanding of user needs.

5.1.9 A Summary of Principles for User-Interface Design

Note: These principles were compiled together by Talin

1. The principle of user profiling

Before we can answer the question "How do we make our user-interfaces better", we must first answer the question: Better for whom? A design that is better for a technically skilled user might not be better for a non-technical businessman or an artist.

One way around this problem is to create user models. [TOG91] has an excellent chapter on brainstorming towards creating "profiles" of possible users. The result of this process is a detailed description of one or more "average" users, with specific details such as:

- What are the user's goals?
- What are the user's skills and experience?
- What are the user's needs?

Armed with this information, we can then proceed to answer the question: How do we leverage the user's strengths and create an interface that helps them achieve their goals?

In the case of a large general-purpose piece of software such as an operating system, there may be many different kinds of potential users. In this case it may be more useful to come up with a list of user dichotomies, such as "skilled vs. unskilled", "young vs. old", etc., or some other means of specifying a continuum or collection of user types.

Another way of answering this question is to talk to some real users. Direct contact between end-users and developers has often radically transformed the development process.

2. The principle of metaphor

Frequently a complex software system can be understood more easily if the user interface is depicted in a way that resembles some commonplace system. The ubiquitous "Desktop metaphor" is an overused and trite example. Another is the tape deck metaphor seen on many audio and video player programs. In addition to the standard transport controls (play, rewind, etc.), the tape deck metaphor can be extended in ways that are quite natural, with functions such

as time-counters and cueing buttons. This concept of "extendibility" is what distinguishes a powerful metaphor from a weak one.

There are several factors to consider when using a metaphor:

- Once a metaphor is chosen, it should be spread widely throughout the interface, rather than used once at a specific point. Even better would be to use the same metaphor spread over several applications (the tape transport controls described above is a good example.) Don't bother thinking up a metaphor, which is only going to apply to a single button.
- There's no reason why an application cannot incorporate several different metaphors, as long as they don't clash. Music sequencers, for example, often incorporate both "tape transport" and "sheet music" metaphors.
- Metaphor isn't always necessary. In many cases the natural function of the software itself is easier to comprehend than any real-world analog of it. Don't strain a metaphor in adapting it to the program's real function. Nor should you strain the meaning of a particular program feature in order to adapt it to a metaphor.
- Incorporating a metaphor is not without certain risks. In particular, whenever physical objects are represented in a computer system, we inherit not only the beneficial functions of those objects but also the detrimental aspects.
- Be aware that some metaphors don't cross cultural boundaries well. For example, Americans would instantly recognize the common U.S. Mailbox (with a rounded top, a flat bottom, and a little red flag on the side), but there are no mailboxes of this style in Europe.

3. The principle of feature exposure

Software developers tend to have little difficulty keeping large, complex mental models in their heads. But not everyone prefers to "live in their heads" -- instead, they prefer to concentrate on analyzing the sensory details of the environment, rather than spending large amounts of time refining and perfecting abstract models. Both type of personality (labeled "Intuitive" and "Sensible" in the Myers-Briggs personality classification) can be equally intelligent, but focus on different aspects of life. It is to be noted that according to some psychological studies "Sensible" outnumber "Intuitive" in the general population by about three to one.

Intuitive prefer user interfaces that utilize the power of abstract models -- command lines, scripts, plug-ins, macros, etc. Sensible prefer user interfaces that utilize their perceptual abilities -- in other words, they like interfaces where the features are "up front" and "in their face". Toolbars and dialog boxes are an example of interfaces that are pleasing to this personality type.

This doesn't mean that you have to make everything a GUI. What it does mean, for both GUI and command line programs, is that the features of the program need to be easily exposed so that a quick visual scan can determine what the program actually does. In some cases, such as a toolbar, the program features are exposed by default. In other cases, such as a printer configuration dialog, the exposure of the underlying printer state (i.e. the buttons and controls which depict the conceptual printing model) are contained in a dialog box, which is brought up by a user action (a feature which is itself exposed in a menu).

Of course, there may be cases where you don't wish to expose a feature right away, because you don't want to overwhelm the beginning user with too much detail. In this case, it is best to structure the application like the layers of an onion, where peeling away each layer of skin reveals a layer beneath. There are various levels of "hiding": Here's a partial list of them in order from most exposed to least exposed:

- Toolbar (completely exposed)
- Menu item (exposed by trivial user gesture)
- Submenu item (exposed by somewhat more involved user gesture)
- Dialog box (exposed by explicit user command)

- Secondary dialog box (invoked by button in first dialog box)
- "Advanced user mode" controls -- exposed when user selects "advanced" option
- Scripted functions

The above notwithstanding, in no case should the primary interface of the application be a reflection of the true complexity of the underlying implementation. Instead, both the interface and the implementation should strive to match a simplified conceptual model (in other words, the design) of what the application does. For example, when an error occurs, the explanation of the error should be phrased in a way that relates to the current user-centered activity, and not in terms of the low-level fault that caused there error.

4. The principle of coherence

There's been some argument over whether interfaces should strive to be "intuitive", or whether an intuitive interface is even possible. However, it is certainly arguable that an interface should be coherent -- in other words logical, consistent, and easily followed. ("Coherent" literally means "stick together", and that's exactly what the parts of an interface design should do.)

Internal consistency means that the program's behaviors make "sense" with respect to other parts of the program. For example, if one attribute of an object (e.g. color) is modifiable using a pop-up menu, then it is to be expected that other attributes of the object would also be editable in a similar fashion. One should strive towards the principle of "least surprise".

External consistency means that the program is consistent with the environment in which it runs. This includes consistency with both the operating system and the typical suite of applications that run within that operating system. One of the most widely recognized forms of external coherence is compliance with user-interface standards. There are many others, however, such as the use of standardized scripting languages, plug-in architectures or configuration methods.

5. The principle of state visualization

Each change in the behavior of the program should be accompanied by a corresponding change in the appearance of the interface. One of the big criticisms of "modes" in interfaces is that many of the classic "bad example" programs have modes that are visually indistinguishable from one another.

Similarly, when a program changes its appearance, it should be in response to a behavior change; A program that changes its appearance for no apparent reason will quickly teach the user not to depend on appearances for clues as to the program's state.

One of the most important kinds of state is the current selection, in other words the object or set of objects that will be affected by the next command. It is important that this internal state be visualized in a way that is consistent, clear, and unambiguous. For example, one common mistake seen in a number of multi-document applications is to forget to "dim" the selection when the window goes out of focus. The result of this is that a user, looking at several windows at once, each with a similar-looking selection, may be confused as to exactly which selection will be affected when they hit the "delete" key. This is especially true if the user has been focusing on the selection highlight, and not on the window frame, and consequently has failed to notice which window is the active one. (Selection rules are one of those areas that are covered poorly by most UI style guidelines, which tend to concentrate on "widgets", although the Mac and Amiga guidelines each have a chapter on this topic.)

6. The principle of shortcuts

Once a user has become experienced with an application, she will start to build a mental model of that application. She will be able to predict with high accuracy what the results of any particular user gesture will be in any given context. At this point, the program's attempts to make things "easy" by breaking up complex actions into simple steps may seem cumbersome. Additionally, as this mental model grows, there will be less and less need to look at the "in your face" exposure of the application's feature set. Instead, pre-memorized "shortcuts" should be available to allow rapid access to more powerful functions.

There are various levels of shortcuts, each one more abstract than its predecessor. For example, in the Emacs editor commands can be invoked directly by name, by menu bar, by a modified keystroke combination, or by a single keystroke. Each of these is more "accelerated" than its predecessor.

There can also be alternate methods of invoking commands that are designed to increase power rather than to accelerate speed. A "recordable macro" facility is one of these, as is a regular-expression search and replace. The important thing about these more powerful (and more abstract) methods is that they should not be the most exposed methods of accomplishing the task. This is why Emacs has the non-regexp version of search assigned to the easy-to-remember "C-s" key.

7. The principle of focus

The human eye is a highly non-linear device. For example, it possesses edge-detection hardware, which is why we see Mach bands whenever two closely matched areas of color come into contact. It also has motion-detection hardware. As a consequence, our eyes are drawn to animated areas of the display more readily than static areas. Changes to these areas will be noticed readily.

The mouse cursor is probably the most intensely observed object on the screen -- it's not only a moving object, but mouse users quickly acquire the habit of tracking it with their eyes in order to navigate. This is why global state changes are often signaled by changes to the appearance of the cursor, such as the well-known "hourglass cursor". It's nearly impossible to miss.

The text cursor is another example of a highly eye-attractive object. Changing its appearance can signal a number of different and useful state changes.

8. The principle of grammar

Many of the operations within a user interface require both a subject (an object to be operated upon), and a verb (an operation to perform on the object). This naturally suggests that actions in the user interface form a kind of grammar. The grammatical metaphor can be extended quite a bit, and there are elements of some programs that can be clearly identified as adverbs, adjectives and such.

The two most common grammars are known as "Action->Object" and "Object->Action". In Action->Object, the operation (or tool) is selected first. When a subsequent object is chosen, the tool immediately operates upon the object. The selection of the tool persists from one operation to the next, so that many objects can be operated on one by one without having to re-select the tool. Action->Object is also known as "modality", because the tool selection is a "mode" which changes the operation of the program. An example of this style is a paint program -- a tool such as a paintbrush or eraser is selected, which can then make many brush strokes before a new tool is selected.

In the Object->Action case, the object is selected first and persists from one operation to the next. Individual actions are then chosen which operate on the currently selected object or objects. This is the method seen in most word processors -- first a range of text is selected, and then a text style such as bold, italic, or a font change can be selected. Object->Action has been called "non-modal" because all behaviors that can be applied to the object are always available. One powerful type of Object->Action is called "direct manipulation", where the object itself is a kind of tool -- an example is dragging the object to a new position or resizing it.

Modality has been much criticized in user-interface literature because early programs were highly modal and had hideous interfaces. However, while non-modality is the clear winner in many situations, there are a large number of situations in life that are clearly modal. For example, in carpentry, its generally more efficient to hammer in a whole bunch of nails at once than to hammer in one nail, put down the hammer, pick up the measuring tape, mark the position of the next nail, pick up the drill, etc.

9. The principle of help

In an essay in [LAUR91] it states that are five basic types of help, corresponding to the five basic questions that users ask:

1. Goal-oriented: "What kinds of things can I do with this program?"
2. Descriptive: "What is this? What does this do?"
3. Procedural: "How do I do this?"
4. Interpretive: "Why did this happen?"
5. Navigational: "Where am I?"

The essay goes on to describe in detail the different strategies for answering these questions, and shows how each of these questions requires a different sort of help interface in order for the user to be able to adequately phrase the question to the application.

For example, "about boxes" are one way of addressing the needs of question of type 1. Questions of type 2 can be answered with a standard "help browser", "tool tips" or other kinds of context-sensitive help. A help browser can also be useful in responding to questions of the third type, but these can sometimes be more efficiently addressed using "cue cards", interactive "guides", or "wizards" which guide the user through the process step-by-step. The fourth type has not been well addressed in current applications, although well-written error messages can help. The fifth type can be answered by proper overall interface design, or by creating an application "roadmap". None of the solutions listed in this paragraph are final or ideal; they are simply the ones in common use by many applications today.

10. The principle of safety

Ted Nelson once said, "Using DOS is like juggling with straight razors. Using a Mac is like shaving with a bowling pin."

Each human mind has an "envelope of risk", that is to say a minimum and maximum range of risk-levels which they find comfortable. A person who finds herself in a situation that is too risky for her comfort will generally take steps to reduce that risk. Conversely, when a person's life becomes too safe -- in other words, when the risk level drops below the minimum threshold of the risk envelope -- she will often engage in actions that increase their level of risk.

This comfort envelope varies for different people and in different situations. In the case of computer interfaces, a level of risk that is comfortable for a novice user might make a "power-user" feel uncomfortably swaddled in safety.

It's important for new users that they feel safe. They don't trust themselves or their skills to do the right thing. Many novice users think poorly not only of their technical skills, but of their intellectual capabilities in general (witness the popularity of the "...for Dummies" series of tutorial books.) In many cases these fears are groundless, but they need to be addressed. Novice users need to be assured that they will be protected from their own lack of skill. A program with no safety net will make this type of user feel uncomfortable or frustrated to the point that they may cease using the program. The "Are you sure?" dialog box and multi-level undo features are vital for this type of user.

At the same time, an expert user must be able to use the program as a virtuoso. She must not be hampered by guard rails or helmet laws. However, expert users are also smart enough to turn off the safety checks -- if the application allows it. This is why "safety level" is one of the more important application configuration options.

Finally, it should be noted that many things in life are not meant to be easy. Physical exercise is one -- "no pain, no gain". A concert performance in Carnegie Hall, a marathon, or the Guinness World Record would be far less impressive if anybody could do it. This is especially pertinent in the design of computer game interfaces, which operate under somewhat different principles than those listed here (although many of the principles in fact do apply).

11. The principle of context

Each user action takes place within a given context -- the current document, the current selection, the current dialog box. A set of operations that is valid in one context may not be valid in another. Even within a single document, there may be multiple levels -- for example, in a structured drawing application, selecting a text object (which can be moved or resized) is generally considered a different state from selecting an individual character within that text object.

It's usually a good idea to avoid mixing these levels. For example, imagine an application that allows users to select a range of text characters within a document, and also allows them to select one or more whole documents (the latter being a distinct concept from selecting all of the characters in a document). In such a case, it's probably best if the program disallows selecting both characters and documents in the same selection. One unobtrusive way to do this is to "dim" the selection that is not applicable in the current context. In the example above, if the user had a range of text selected, and then selected a document, the range of selected characters could become dim, indicating that the selection was not currently pertinent. The exact solution chosen will of course depend on the nature of the application and the relationship between the contexts.

Another thing to keep in mind is the relationship between contexts. For example, it is often the case that the user is working in a particular task-space, when suddenly a dialog box will pop up asking the user for confirmation of an action. This sudden shift of context may leave the user wondering how the new context relates to the old. This confusion is exacerbated by the terseness of writing style that is common amongst application writers. Rather than the "Are you sure?" confirmation mentioned earlier, something like "There are two documents unsaved. Do you want to quit anyway?" would help to keep the user anchored in their current context.

12. The principle of aesthetics

It's not necessary that each program be a visual work of art. But it's important that it not be ugly. There are a number of simple principles of graphical design that can easily be learned, the most basic of which was coined by artist and science fiction writer William Rotsler: "Never do anything that looks to someone else like a mistake." The specific example Rotsler used was a painting of a Conan-esque barbarian warrior swinging a mighty broadsword. In this picture, the tip of the broadsword was just off the edge of the picture. "What that looks like", said Rotsler, "is a picture that's been badly cropped. They should have had the tip of the sword either clearly within the frame or clearly out of it."

An interface example can be seen in the placement of buttons -- imagine five buttons, each with five different labels that are almost the same size. Because the buttons are packed using an automated-layout algorithm, each button is almost but not exactly the same size. As a result, though the author has placed much care into his layout, it looks carelessly done. A solution would be to have the packing algorithm know that buttons that are almost the same size look better if they are exactly the same size -- in other words, to encode some of the rules of graphical design into the layout algorithm. Similar arguments hold for manual widget layout.

Another area of aesthetics to consider is the temporal dimension. Users don't like using programs that feel sluggish or slow. There are many tricks that can be used to make a slow program "feel" snappy, such as the use of off-screen bitmaps for rendering, which can then be blotted forward in a single operation. (A pet peeve of this particular author is buttons that flicker when the button is being activated or the window is being resized. Multiply redundant refreshing of buttons when changing state is one common cause of this.)

13. The principle of user testing

In many cases a good software designer can spot fundamental defects in a user interface. However, there are many kinds of defects which are not so easy to spot, and in fact an experienced software designer is often less capable of spotting them than the average person. In other cases, a bug can only be detected while watching someone else use the program.

User-interface testing, that is, the testing of user-interfaces using actual end-users, has been shown to be an extraordinarily effective technique for discovering design defects. However, there are specific techniques that can be used to maximize the effectiveness of end-user testing. These are outlined in both [TOG91] and [LAUR91] and can be summarized in the following steps:

- Set up the observation. Design realistic tasks for the users, and then recruit end-users that have the same experience level as users of your product (Avoid recruiting users who are familiar with your product however).
- Describe to the user the purpose of the observation. Let them know that you're testing the product, not them, and that they can quit at any time. Make sure that they understand if anything bad happens, it's not their fault, and that it's helping you to find problems.
- Talk about and demonstrate the equipment in the room.
- Explain how to "think aloud". Ask them to verbalize what they are thinking about as they use the product, and let them know you'll remind them to do so if they forget.
- Explain that you will not provide help.
- Describe the tasks and introduce the product.
- Ask if there are any questions before you start; then begin the observation.
- Conclude the observation. Tell them what you found out and answer any of their questions.
- Use the results.

User testing can occur at any time during the project, however, it's often more efficient to build a mock-up or prototype of the application and test that before building the real program. It's much easier to deal with a design defect before it's implemented than after. Tognazzini suggests that you need no more than three people per design iteration -- any more than that and you are just confirming problems already found.

14. The principle of humility

Some of the most valuable insights can be gained by simply watching other people attempt to use your program. Others can come from listening to their opinions about the product. Of course, you don't have to do exactly everything they say. It's important to realize that each of you, user and developer, has only part of the picture. The ideal is to take a lot of user opinions, plus your insights as a developer and reduce them into an elegant and seamless whole -- a design that, though it may not satisfy everyone, will satisfy the greatest needs of the greatest number of people.

One must be true to one's vision. A product built entirely from customer feedback is doomed to mediocrity, because what users want most are the features that they cannot anticipate.

But a single designer's intuition about what is good and bad in an application is insufficient. Program creators are a small, and not terribly representative, subset of the general computing population.

Some things designers should keep in mind about their users:

- Most people have a biased idea as to what the "average" person is like. This is because most of our interpersonal relationships are in some way self-selected. It's a rare person whose daily life brings them into contact with other people from a full range of personality types and backgrounds. As a result, we tend to think that others think "mostly like we do." Designers are no exception.
- Most people have some sort of core competency, and can be expected to perform well within that domain.

- The skill of using a computer (also known as "computer literacy") is actually much harder than it appears.
- The lack of "computer literacy" is not an indication of a lack of basic intelligence. While native intelligence does contribute to one's ability to use a computer effectively, there are other factors which seem to be just as significant, such as a love of exploring complex systems, and an attitude of playful experimentation. Much of the fluency with computer interfaces derives from play -- and those who have dedicated themselves to "serious" tasks such as running a business, curing disease, or helping victims of tragedy may lack the time or patience to be able to devote effort to it.
- A high proportion of programmers are introverts, compared to the general population. This doesn't mean that they don't like people, but rather that there are specific social situations that make them uncomfortable. Many of them lack social skills, and retreat into the world of logic and programming as an escape; As a result, they are not experienced people-watchers.

The best way to avoid misconceptions about users is to spend some time with them, especially while they are actually using a computer. Do this long enough, and eventually you will get a "feel" for how the average non-technical person thinks. This will increase your ability to spot defects, although it will never make it 100%, and will never be a substitute for user testing.

Tips and Techniques for Interface design

Note: These tips and techniques were described by Scott W. Ambler (IBM Rational)

1. Consistency, consistency, consistency. I believe the most important thing you can possibly do is ensure your user interface works consistently. If you can double-click on items in one list and have something happen, then you should be able to double-click on items in any other list and have the same sort of thing happen. Put your buttons in consistent places on all your windows, use the same wording in labels and messages, and use a consistent color scheme throughout. Consistency in your user interface enables your users to build an accurate mental model of the way it works, and accurate mental models lead to lower training and support costs.
2. Set standards and stick to them. The only way you can ensure consistency within your application is to set user interface design standards, and then stick to them. You should follow Agile Modeling (AM)'s Apply Modeling Standards practice in all aspects of software development, including user interface design.
3. Be prepared to hold the line. When you are developing the user interface for your system you will discover that your stakeholders often have some unusual ideas as to how the user interface should be developed. You should definitely listen to these ideas but you also need to make your stakeholders aware of your corporate UI standards and the need to conform to them.
4. Explain the rules. Your users need to know how to work with the application you built for them. When an application works consistently, it means you only have to explain the rules once. This is a lot easier than explaining in detail exactly how to use each feature in an application step-by-step.
5. Navigation between major user interface items is important. If it is difficult to get from one screen to another, then your users will quickly become frustrated and give up. When the flow between screens matches the flow of the work the user is trying to accomplish, then your application will make sense to your users. Because different users work in different ways, your system needs to be flexible enough to support their various approaches. User interface-flow diagrams should optionally be developed to further your understanding of the flow of your user interface.
6. Navigation within a screen is important. In Western societies, people read left to right and top to bottom. Because people are used to this, should you design screens that are also organized left to right and top to bottom when designing a user interface for people from this culture? You want to organize navigation between widgets on your screen in a manner users will find familiar to them.
7. Word your messages and labels effectively. The text you display on your screens is a primary source of information for your users. If your text is worded poorly, then your users will

perceive your interface poorly. Using full words and sentences, as opposed to abbreviations and codes, makes your text easier to understand. Your messages should be worded positively, imply that the user is in control, and provide insight into how to use the application properly. For example, which message do you find more appealing, "You have input the wrong information" or "An account number should be eight digits in length." Furthermore, your messages should be worded consistently and displayed in a consistent place on the screen. Although the messages "The person's first name must be input" and "An account number should be input" are separately worded well, together they are inconsistent. In light of the first message, a better wording of the second message would be "The account number must be input" to make the two messages consistent.

8. Understand the UI widgets. You should use the right widget for the right task, helping to increase the consistency in your application and probably making it easier to build the application in the first place. The only way you can learn how to use widgets properly is to read and understand the user-interface standards and guidelines your organization has adopted.
9. Look at other applications with a grain of salt. Unless you know another application has been verified to follow the user interface-standards and guidelines of your organization, don't assume the application is doing things right. Although looking at the work of others to get ideas is always a good idea, until you know how to distinguish between good user interface design and bad user interface design, you must be careful. Too many developers make the mistake of imitating the user interface of poorly designed software.
10. Use color appropriately. Color should be used sparingly in your applications and, if you do use it, you must also use a secondary indicator. The problem is that some of your users may be colorblind and if you are using color to highlight something on a screen, then you need to do something else to make it stand out if you want these people to notice it. You also want to use colors in your application consistently, so you have a common look and feel throughout your application.
11. Follow the contrast rule. If you are going to use color in your application, you need to ensure that your screens are still readable. The best way to do this is to follow the contrast rule: Use dark text on light backgrounds and light text on dark backgrounds. Reading blue text on a white background is easy, but reading blue text on a red background is difficult. The problem is not enough contrast exists between blue and red to make it easy to read, whereas there is a lot of contrast between blue and white.
12. Align fields effectively. When a screen has more than one editing field, you want to organize the fields in a way that is both visually appealing and efficient. I have always found the best way to do so is to left-justify edit fields: in other words, make the left-hand side of each edit field line up in a straight line, one over the other. The corresponding labels should be right justified and placed immediately beside the field. This is a clean and efficient way to organize the fields on a screen.
13. Expect your users to make mistakes. How many times have you accidentally deleted some text in one of your files or in the file itself? Were you able to recover from these mistakes or were you forced to redo hours, or even days, of work? The reality is that to err is human, so you should design your user interface to recover from mistakes made by your users.
14. Justify data appropriately. For columns of data, common practice is to right-justify integers, decimal align floating-point numbers, and to left-justify strings.
15. Your design should be intuit able. In other words, if your users don't know how to use your software, they should be able to determine how to use it by making educated guesses. Even when the guesses are wrong, your system should provide reasonable results from which your users can readily understand and ideally learn.
16. Don't create busy user interfaces. Crowded screens are difficult to understand and, hence, are difficult to use. Experimental results show that the overall density of the screen should not exceed 40 percent, whereas local density within groupings should not exceed 62 percent.
17. Group things effectively. Items that are logically connected should be grouped together on the screen to communicate they are connected, whereas items that have nothing to do with each other should be separated. You can use white space between collections of items to group them and/or you can put boxes around them to accomplish the same thing.
18. Take an evolutionary approach. Techniques such as user interface prototyping and Agile Model Driven Development (AMDD) are critical to your success as a developer.

5.1.10 Usability

Usability is a term used to denote the ease with which people can employ a particular tool or other human-made object in order to achieve a particular goal. Usability can also refer to the methods of measuring usability and the study of the principles behind an object's perceived efficiency or elegance.

In human-computer interaction and computer science, usability usually refers to the elegance and clarity with which the interaction with a computer program or a web site is designed. The term is also used often in the context of products like consumer electronics, or in the areas of communication, and knowledge transfer objects (such as a cookbook, a document or online help). It can also refer to the efficient design of mechanical objects such as a door handle or a hammer.

Usability is a quality attribute that assesses how easy user interfaces are to use. The word "usability" also refers to methods for improving ease-of-use during the design process.

Usability is defined by five quality components:

- Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
- Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- Satisfaction: How pleasant is it to use the design?

There are many other important quality attributes. A key one is utility, which refers to the design's functionality: Does it do what users need? Usability and utility are equally important: It matters little that something is easy if it's not what you want. It's also no good if the system can hypothetically do what you want, but you can't make it happen because the user interface is too difficult. To study a design's utility, you can use the same user research methods that improve usability.

5.1.11 Why Usability is Important

On the Web, usability is a necessary condition for survival. If a website is difficult to use, people leave. If the homepage fails to clearly state what a company offers and what users can do on the site, people leave. If users get lost on a website, they leave. If a website's information is hard to read or doesn't answer users' key questions, they leave. Note a pattern here? There's no such thing as a user reading a website manual or otherwise spending much time trying to figure out an interface. There are plenty of other websites available; leaving is the first line of defense when users encounter a difficulty.

The first law of e-commerce is that if users cannot find the product, they cannot buy it either.

For intranets, usability is a matter of employee productivity. Time users waste being lost on your intranet or pondering difficult instructions is money you waste by paying them to be at work without getting work done.

Current best practices call for spending about 10% of a design project's budget on usability. On average, this will more than double a website's desired quality metrics and slightly less than double an intranet's quality metrics. For software and physical products, the improvements are typically smaller — but still substantial — when you emphasize usability in the design process.

For internal design projects, think of doubling usability as cutting training budgets in half and doubling the number of transactions employees perform per hour. For external designs, think of doubling sales, doubling the number of registered users or customer leads, or doubling whatever other desired goal motivated your design project.

5.1.12 How to Improve Usability

There are many methods for studying usability, but the most basic and useful is user testing, which has three components:

- Get hold of some representative users, such as customers for an e-commerce site or employees for an intranet (in the latter case, they should work outside your department).
- Ask the users to perform representative tasks with the design.
- Observe what the users do, where they succeed, and where they have difficulties with the user interface. Shut up and let the users do the talking.

It's important to test users individually and let them solve any problems on their own. If you help them or direct their attention to any particular part of the screen, you have contaminated the test results.

To identify a design's most important usability problems, testing five users is typically enough. Rather than run a big, expensive study, it's a better use of resources to run many small tests and revise the design between each one so you can fix the usability flaws as you identify them. Iterative design is the best way to increase the quality of user experience. The more versions and interface ideas you test with users, the better.

User testing is different from focus groups, which are a poor way of evaluating design usability. Focus groups have a place in market research, but to evaluate interaction designs you must closely observe individual users as they perform tasks with the user interface. Listening to what people say is misleading: you have to watch what they actually do.

5.1.13 When to Work on Usability

Usability plays a role in each stage of the design process. The resulting need for multiple studies is one reason I recommend making individual studies fast and cheap. Here are the main steps:

1. Before starting the new design, test the old design to identify the good parts that you should keep or emphasize, and the bad parts that give users trouble.
2. Unless you're working on an intranet, test your competitors' designs to get cheap data on a range of alternative interfaces that have similar features to your own. (If you work on an intranet, read the intranet design annuals to learn from other designs.)
3. Conduct a field study to see how users behave in their natural habitat.
4. Make paper prototypes of one or more new design ideas and test them. The less time you invest in these design ideas the better, because you'll need to change them all based on the test results.
5. Refine the design ideas that test best through multiple iterations, gradually moving from low-fidelity prototyping to high-fidelity representations that run on the computer. Test each iteration.
6. Inspect the design relative to established usability guidelines, whether from your own earlier studies or published research.
7. Once you decide on and implement the final design, test it again. Subtle usability problems always creep in during implementation.

Don't defer user testing until you have a fully implemented design. If you do, it will be impossible to fix the vast majority of the critical usability problems that the test uncovers. Many of these problems are likely to be structural, and fixing them would require major rearchitecting.

The only way to a high-quality user experience is to start user testing early in the design process and to keep testing every step of the way.

5.1.14 Where to Test

If you run at least one user study per week, it's worth building a dedicated usability laboratory. For most companies, however, it's fine to conduct tests in a conference room or an office — as long as you can close the door to keep out distractions. What matters is that you get hold of real users and sit with them while they use the design. A notepad is the only equipment you need.

5.1.15 User Centered Design (UCD)

In broad terms, user-centered design (UCD) is a design philosophy and a process in which the needs, wants, and limitations of the end user of an interface or document are given extensive attention at each stage of the design process. User-centered design can be characterized as a multi-stage problem solving process that not only requires designers to analyze and foresee how users are likely to use an interface, but to test the validity of their assumptions with regards to user behavior in real world tests with actual users. Such testing is necessary as it is often very difficult for the designers of an interface to understand intuitively what a first-time user of their design experiences, and what each user's learning curve may look like.

The chief difference from other interface design philosophies is that user-centered design tries to optimize the user interface around how people can, want, or need to work, rather than forcing the users to change how they work to accommodate the system or function.

5.1.16 Elements

5.1.16.1 Visibility

Visibility helps the user construct a mental model of the document. Models help the user predict the effect(s) of their actions while using the document. Important elements (such as those that aid navigation) should be emphatic. Users should be able to tell from a glance what they can and cannot do with the document.

5.1.16.2 Accessibility

Users should be able to find information quickly and easily throughout the document, whether it is long or short. Users should be offered various ways to find information (such navigational elements, search functions, table of contents, clearly labeled sections, page numbers, color coding, etc). Navigational elements should be consistent with the genre of the document. ‘Chunking’ is a useful strategy that involves breaking information into small pieces that can be organized into some type meaningful order or hierarchy The ability to skim the document allows users to find their piece of information by scanning rather than reading. bold and italic words are often used.

5.1.16.3 Legibility

Text should be easy to read: Through analysis of the rhetorical situation the designer should be able to determine a useful font style. Ornamental fonts and text in all capital letters are hard to read, but italics and bolding can be helpful when used correctly. Large or small body text is also hard to read. (9-11 pt sans serif and 11-12 pt serif is recommended.) High figure-ground contrast between text and background increases legibility. Dark text against a light background is most legible.

5.1.16.4 Language

Depending on the rhetorical situation certain types of language are needed. Short sentences are helpful. Unless the situation calls for it don't use jargon or technical terms. Many writer will choose to use active voice, verbs (instead of noun strings or nominals), and simple sentence structure.

5.1.17 Rhetorical Situation

A User Centered Design is focused around the rhetorical situation. The rhetorical situation shapes the design of an information medium. There are three elements to consider in a rhetorical situation: Audience, Purpose, Context.

5.1.17.1 Audience

The audience is the people who will be using the document. The designer must consider their age, geographical location, ethnicity, gender, education, etc.

5.1.17.2 Purpose

The purpose is how the document will be used, and what the audience will be trying to accomplish while using the document. The purpose usually includes purchasing a product, selling ideas, performing a task, instruction, and all types of persuasion.

5.1.17.3 Context

The context is the circumstances surrounding the situation. The context often answers the question: What situation has prompted the need for this document? Context also includes any social or cultural issues that may surround the situation.

5.1.18 Typical UCD Methodology

Most user-centered design methodologies are more detailed in suggesting specific activities, and the time within a process when they should be completed. Here, the UCD activities are broken down into four phases: Analysis, Design, Implementation and Deployment, with suggested activities for each phase. They are:

Analysis Phase

- Meet with key stakeholders to set vision
- Include usability tasks in the project plan
- Assemble a multidisciplinary team to ensure complete expertise
- Develop usability goals and objectives
- Conduct field studies
- Look at competitive products
- Create user profiles
- Develop a task analysis
- Document user scenarios
- Document user performance requirements

Design Phase

- Begin to brainstorm design concepts and metaphors
- Develop screen flow and navigation model
- Do walkthroughs of design concepts
- Begin design with paper and pencil
- Create low-fidelity prototypes
- Conduct usability testing on low-fidelity prototypes
- Create high-fidelity detailed design
- Do usability testing again
- Document standards and guidelines
- Create a design specification

Implementation Phase

- Do ongoing heuristic evaluations
- Work closely with delivery team as design is implemented
- Conduct usability testing as soon as possible

Deployment Phase

- Use surveys to get user feedback
- Conduct field studies to get info about actual use
- Check objectives using usability testing

5.1.19 Focus on more than just computers and single users

While user-centered design is often viewed as being focused on the development of computer and paper interfaces, the field has a much wider application. The design philosophy has been applied to a diverse range of user interactions, from car dashboards to service processes such as the end-to-end experience of visiting a restaurant, including interactions such as being seated, choosing a meal, ordering food, paying the bill etc.

When user-centered design is applied to more than single user interactions, it is often referred to as user experience. A user experience comprises a number of separate interfaces, human-to-human contacts, transactions and conceptual architectures. The restaurant example (above) is an example of this - ordering a meal or paying the bill are two user interactions, but they are a part of the "user experience" called dining out. It is not enough to have the separate interactions that comprise an experience being usable. The goal is that each interaction should integrate with every other interaction that forms a part of a single experience. In this way, the experience as a whole is rendered usable.

In product design, this is sometimes referred to as the "out of the box experience," referring to all tasks the user must complete from first opening the box the product is shipped in, through unpacking, reading the directions, assembly, first use and continuing use.

5.2 Guidelines

5.2.1 What is guideline

A guideline is any document that aims to streamline particular processes according to a set routine. By definition, following a guideline is never mandatory (protocol would be a better term for a mandatory procedure). Guidelines are an essential part of the larger process of governance.

Guidelines may be issued by and used by any organization (governmental or private) to make the actions of its employees or divisions more predictable, and presumably of higher quality.

5.2.2 HIG and its Importance

Human interface guidelines (HIG) are software development documents, which offer application developers a set of recommendations. Their aim is to improve the experience for the users by making application interfaces more intuitive, learnable, and consistent. Most guides limit themselves to defining a common look and feel for applications in a particular desktop environment. The guides enumerate specific policies. Policies are sometimes based on studies of human-computer interaction (so called usability studies), but most are based on arbitrary conventions chosen by the platform developers.

Human interface guidelines will dictate a set of rules for general usability. They often describe the visual design rules, including icon and window design and style. Frequently they specify how user input and interaction mechanisms work. Some describe the language style.

A HIG will sometimes also define standard terminology and semantics related to certain elements or actions. In general this is restricted to the semantics of the desktop environment or the file system. As the majority of user problems come from the semantics of the applications, this is a drawback to most HIGs.

The more important version of HIGs isthose done for groups or applications. In this case the HIG will build on a platform HIG by adding the common semantics for a range of functions.

The central aim of a HIG is to create a consistent experience across the environment (generally an operating system or desktop environment), including the applications and other tools being used. This means both applying the same visual design and creating consistent access to and behavior of common elements of the interface - from simple ones such as buttons and icons up to more complex constructions, such as dialog boxes.

HIGs should be taken at face value; their recommendations and advice are meant to help developers create better applications, but developers are naturally free to break them if they think that the guidelines do not fit their application. The only repercussion for doing so may be that the organization publishing the HIG does not give the application it's blessing. Mozilla Firefox's user interface, for example, goes against the GNOME project's HIG, which is one of the main arguments for including Epiphany instead of Firefox in the GNOME distribution. But such departures should only be taken when there is evidence from usability testing that the interface is improved.

5.3 Existing Guidelines (UX/UI/HCI)

5.3.1 Apple (Mac OS X)

User Experience encompasses the visual appearance, interactive behavior, and assistive capabilities of software. As much as a powerful feature set, the look and feel of the application can determine its appeal to users.

For example, an application with a great user experience should:

- Embody user-friendly design principles
- Have a professional, consistent look, with quality icons and graphics
- Support alternative input devices for users with disabilities
- Offer a simple installation experience

Apple provides a complete guideline and reference library so that the Apple users get the maximum user experience.

Apple has the world's most advanced operating system, Mac OS X, which combines a powerful core foundation with a compelling user interface called Aqua. With advanced features and an aesthetically refined use of color, transparency, and animation, Mac OS X makes computing even easier for new users, while providing the productivity that professional users have come to expect of the Macintosh. The user interface features, behaviors, and appearances deliver a well-organized and cohesive user experience available to all applications developed for Mac OS X.

These guidelines are designed to assist the developers in developing products that provide Mac OS X users with a consistent visual and behavioral experience across applications and the operating system. Following the guidelines is to developers' advantage because:

- Users will learn application faster if the interface looks and behaves like applications they're already familiar with.

- Users can accomplish their tasks quickly, because well-designed applications don't get in the user's way.
- Users with special needs will find the product more accessible.
- The application will have the same modern, elegant appearance as other Mac OS X applications.
- The application will be easier to document, because an intuitive interface and standard behaviors don't require as much explanation.
- Customer support calls will be reduced (for the reasons cited above).
- The application will be easier to localize, because Apple has worked through many localization issues in the Aqua design process.
- Media reviews of the product will be more positive; reviewers easily target software that doesn't look or behave the way "true" Macintosh applications do.

The implementation of Apple's human interface principles make the Macintosh what it is: intuitive, friendly, elegant, and powerful.

The Apple UX site is located at http://developer.apple.com/documentation/UserExperience

NOTE: Apple being non-open source OS does not give the developers the OS UX guideline. They only provide application UX guideline so that the developers' made application suits with their OS (Mac OS X) perfectly.

5.3.2 Microsoft (Windows OS)

The goals for these official Windows Vista® User Experience Guidelines (or "UX Guide" for short) are to:

- Establish a high quality and consistency baseline for all Windows Vista-based applications.
- Answer developers' specific user experience questions.
- Make developers' job easier!

5.3.2.1 Top Rules for the Windows Vista User Experience

This article summarizes the top rules for creating high-quality, consistent Windows Vista® user interfaces (UIs).

1. Use the Aero Theme and System Font (Segoe UI)
2. Use common controls and common dialogs
3. Use the standard window frame, use glass judiciously
4. Use icons and graphics consistent with the Windows Vista style and quality
5. Use task dialogs for new or frequently used dialog boxes and error messages
6. Use Aero Wizards
7. Use Windows Explorer-hosted, navigation-based user interfaces, provide a Back button
8. Use the Windows Search model
9. Use the Windows Vista tone in all UI text
10. Clean up the UI
11. Use notifications judiciously
12. Reserve time for "fit and finish"!

Microsoft Windows Vista UX articles: http://msdn2.microsoft.com/en-us/library/aa185848.aspx

Microsoft Windows Vista UX guidelines: http://msdn2.microsoft.com/en-us/library/aa511258.aspx

NOTE: Microsoft being non-open source OS does not give the developers the OS UX guideline. They only provide application UX guideline so that the developers' made application suits with their OS (Windows Vista) perfectly.

5.3.3 KDE

The Human Interface Guidelines are part of an important set of documents, which describe how KDE applications behave and look. The guidelines document knowledge gathered by KDE developers as well as experts in the various fields that these guidelines touch.

KDE is serious about consistency and high standards of usability throughout its suite of applications. Therefore a group of experts have recently been formed to create just such a guide, and do so in the open source manner that KDE was formed in.

5.3.3.1 KDE4 Vision

KDE4, the next major version of the K Desktop Environment, will make a great leap forward and make Linux an excellent OS choice for the more sophisticated 50% of computer users out there, compared to the current 5%: the determined and the devoted.

KDE4 will deliver a desktop environment (including core applications) so purely task-oriented and delivering such a seamlessly flowing user experience that it makes the users of the current monopolist OS "want to have it".

KDE4 will not alienate the KDE3 user base, though inevitably some will be upset by changes for the better. Anything that makes Linux interesting for technical users (shells, compilation, drivers, minute user settings) will be available; not as the default way of doing things, but at the user's discretion.

In short:

- KDE stands out for technical excellence, reliability and stability.
- KDE delivers a seamlessly flowing and consistent use experience and supports users' workflows.
- KDE is powerful, yet simple and clear.
- KDE has a breathtakingly beautiful look and feel.
- KDE fully supports accessibility.

KDE4 guidelines: http://wiki.openusability.org/guidelines/index.php/Main_Page

5.3.4 GNOME

Theguidelines tells the developers how to create applications that look right, behave properly, and fit into the GNOME user interface as a whole. It is written for interface designers, graphic artists and software developers who will be creating software for the GNOME environment. Both specific advice on making effective use of interface elements, and the philosophy and general design principles behind the GNOME interface are covered.

These guidelines are meant to help developer design and write applications that are easy to use and consistent with the GNOME desktop. Following these guidelines will have many benefits:

- Users will learn to use the program faster, because interface elements will look and behave the way they are used to.
- Novice and advanced users alike will be able accomplish tasks quickly and easily, because the interface won't be confusing or make things difficult.

- Your application will have an attractive look that fits in with the rest of the desktop.
- Your application will continue to look good when users change desktop themes, fonts and colors.
- Your application will be accessible to all users, including those with disabilities or special needs.

To help the developers achieve these goals, these guidelines will cover basic interface elements, how to use them and put them together effectively, and how to make applications integrate well with the desktop.

The recommendations here build on design aspects that have worked well in other systems, including Mac OS, Windows, Java and KDE. At the same time they retain a uniquely GNOME flavor.

For the full GNOME Human Interface Guideline, head to:

GNOME HIG 2.0: http://library.gnome.org/devel/hig-book/stable/

Old GNOME HIG: http://developer.gnome.org/projects/gup/hig/

5.4 Learning from Apple

5.4.1 Apple UX overview

The user experience for Mac OS X applications encompasses the visual appearance, interactive behavior, and assistive capabilities of software. With the Aqua graphical user interface, Universal Access features, and user-assistive technologies like VoiceOver, Core Animation (which enables animation of the user interface), and the Instant Message framework, you can deliver the cohesive and professional user experience that Macintosh users have come to expect. It's easy to leverage the user experience technologies of Mac OS X to make great Macintosh software.

The most visible expression of the integration and power of Mac OS X is its Aqua user interface. Aqua incorporates the visual appearance of icons, menus, windows and controls with high-quality graphics and user-centric design to produce a user experience that is both functional and appealing. Consistent with Apple's design philosophy, visual enhancements such as color, transparency and animation serve not just as beautiful images, but as cues to the functionality and operation of the system and its applications. Time Machine is one example where Aqua uses the sophisticated graphics and animation capabilities in Mac OS X to provide strong visual cues and user feedback. Time Machine allows for quick and efficient disk backup and recovery, and uses a fly-through metaphor to generate the illusion of movement through time.

Core Animation is a framework that makes it simple for Mac developers to add visually stunning user interfaces, graphics, and animations to applications. Easily used with Cocoa views and controls, Core Animation lets you animate your user interface so that users immediately see the affects of choices they've made. Whether it's shifting items in a list to create room for a new item you fade into view—as in the iChat Buddy List—or the shifting of icons in the Dock to allow another icon to be added, or the gorgeous movement of options within FrontRow, Core Animation is behind it all, providing a powerful solution for presenting advanced user interfaces in your application.

Apple offers developers a complete guide to Mac OS X user experience design with the Apple Human Interface Guidelines. These guidelines provide detailed instructions on how to create an intuitive interface that enables users to accomplish tasks quickly and efficiently, while maintaining the consistency and ease of learning that characterizes most successful Macintosh applications. Interface Builder, part of the Xcode Tools, is Apple's graphical editor for designing an application's Aqua user interface.

Mac OS X provides equal access to everyone, going beyond the requirements of the federally mandated accessibility statute to provide smooth and elegant features to those with visual, hearing, and learningdifficulties. The Universal Access capabilities of Mac OS X provide a great user experience to help users access the Macintosh through speech, audible cues, and keyboard navigation. As a developer, you can make sure your application is easily accessible by testing it against available accessibility clients such as Accessibility Verifier, Accessibility Inspector, and VoiceOver.

Universal Access includes excellent speech recognition and speech synthesis interfaces that can recognize and speak U.S. English. Fully integrated into the Mac OS X Aqua user interface, VoiceOver reads out loud the content of documents such as webpages, email messages, and word-processing files. It provides acomprehensive audible description of your workspace and all the activities taking place on your computer, and includes a rich set of keyboard commands that allows you to navigate the Mac OS X interface andinteract with application and system controls. VoiceOver includes some exciting advanced features, such as the ability to handle fast speaking rates (over 750 words per minute) without choppiness, and the evaluation of an entire paragraph to determine the speaking context. VoiceOver also includes a realistic voice in Alex, and support for refreshable braille displays.

The Mac OS X accessibility architecture also supports assistive technologies, including hardware such as screen readers and specialized input devices. In Mac OS X, most system-wide accessibility features just work, and Cocoa applications automatically take advantage of those features. You should take the time to provide full keyboard navigation in the custom views and controls of your application, and also provide keyboard alternatives for actions such as drag and drop.

Mac OS X also includes numerous built-in software technologies that enhance productivity and collaboration. The Instant Message framework allows you to programatically determine who is available online, and starts iChat Theater sessions to share supporting video and audio content during a videoconference. The Address Book framework is a centralized system to manage users' contacts and personal information. You can use it in your application to give users context-sensitive access to a single address book and simplify the process of managing contacts in multiple applications. Apple Help and Help Tags are two built-in technologies that deliver context-sensitive help to your users and, when used together, contribute to improved ease of use and learning.

With advanced features and an esthetically refined use of color, transparency, and animation, Mac OS X makes computing even easier for new users, while providing the productivity that professional users have come to expect on the Macintosh. The user interface features, behaviors, and appearances deliver a well-organized and cohesive user experience available to all applications developed for Mac OS X.

5.4.2 Learning from Apple UX Guideline

Apple has the world's most advanced operating system, Mac OS X, which combines a powerful core foundation with a compelling user interface called Aqua. With advanced features and an aesthetically refined use of color, transparency, and animation, Mac OS X makes computing even easier for new users, while providing the productivity that professional users have come to expect of the Macintosh. The user interface features, behaviors, and appearances deliver a well-organized and cohesive user experience available to all applications developed for Mac OS X.

Apple designs their Mac OS X in such a way so that:

- Users will learn the operations faster, because of the intuitive and familiar user interface.
- When it comes to operating, users can accomplish their tasks quickly, because well-designed interface does not get in the user's way.
- Accessibility features helps special users to be gain productivity.

- All the applications that are designed for Mac OS X will have the same, modern appearance, which saves a lot of time if the user had to learn the new interface.
- Less documentation is needed, as the intuitive interface and standard behaviors don't require as much explanation.

The implementation of Apple's human interface principles makes the Macintosh what it is: intuitive, friendly, elegant, and powerful.

Apple does not disclose what they do for gaining Mac OS X user experience. But they suggest the developers to maintain their provided guideline to achieve maximum user experience. So, it is obvious that they also use some sort of guideline so that they can achieve user experience as well.

Here is some short notes what Apple might use themselves for Mac OS X development.

5.4.3 Application Design Fundamentals

[Reference: Apple Human Interface Guidelines, ADC, 2006-10-03.]

5.4.3.1 The Design Process

5.4.3.1.aInvolving Users in the Design Process

Apple cares about users experience. They have their own philosophy and psychology behind it. They want their Operating System intuitive and attractive to gain only one thing - users acceptance. They believe involving the users for development helps decision-making processes and better quality designs.

- Know The Audience
- Analyze User Tasks
- Build Prototypes
- Observe Users
- Guidelines for Conducting User Observations

5.4.3.1.bMaking Design Decisions

- Avoid Feature Cascade
- Apply the 80 Percent Solution

5.4.3.1.cCharacteristics of Great Software

Users are attracted to the Macintosh in general and to Mac OS X specifically because they feel the combination offers a superior user experience over other platforms. Macintosh computers are stylish, flexible, easy to set up, easy to maintain, and powerful. Mac OS X combines a reliable core with an intuitive design, stunning graphics, excellent security, and the features users want. Third-party applications enhance this package by delivering specific vertical solutions with sophisticated features and behaviors that are consistent with Apple guidelines.

5.4.3.1.dHigh Performance

Performance is the perceived measure of how fast or efficient your software is and it is critical to the success of all software. If your software seems slow, users may be less inclined to buy it. Even software that uses the most optimal algorithms may seem slow if it spends more time processing data than responding to the user.

5.4.3.1.eEase of Use

An easy-to-use program offers a compelling, intuitive experience for the user. It offers elegant solutions to complex problems and has a well thought out interface that uses familiar paradigms. It is easy to install and configure because it makes intelligent choices for the user, but it also gives the user the option to override those choices when needed. It presents the user with tools that are relevant in the current context, eliminating or disabling irrelevant tools. It also warns the user against performing dangerous actions and provides ways to undo those actions if taken.

5.4.3.1.fAttractive Appearance

One feature that draws users to the Macintosh platform, and to Mac OS X in particular, is the stylish design and attractive appearance of the hardware and software. Although creating attractive hardware and system software is Apple's job, you should take advantage of the strengths of Mac OS X to give your own software an attractive appearance. The Finder and other applications that come with Mac OS X use high-resolution, high-quality graphics and icons that include 32-bit color and transparency. Make sure that your applications also use high-quality graphics both for the sake of appearance and to better convey relevant information tousers. For example, the system uses pulsing buttons to identify the most likely choice and transparency effects to add a dimensional quality to windows.

5.4.3.1.gReliability

A reliable program is one that earns the user's trust. Such a program presents information to the user in an expected and desired way. A reliable program maintains the integrity of the user's data and does everything possible to prevent data loss or corruption. It also has a certain amount of maturity to it and can handle complex situations without crashing. Reliability is important in all areas of software design, but especially in areas where a program may be running for an extended period of time. For example, scientific programs often perform calculations on large data sets and can take a long time to complete. If such a program were to crash during a long calculation, the scientist could lose days or weeks worth of work.

5.4.3.1.hAdaptability

An adaptable program is one that adjusts appropriately to its surroundings; that is, it does not stop working when the current conditions change. If a network connection goes down, an adaptable program lets the user continue to work offline. Similarly, if certain resources are locked or become unavailable, an adaptable program finds other ways to meet the user's request. One of the strengths of Mac OS X is its ability to adapt to configuration changes quickly and easily. For example, if the user changes a computer's network configuration from System Preferences, the changes are automatically picked up by applications such as Safari and Mail, which use CFNetwork to handle network configuration changes automatically.

5.4.3.1.iInteroperability

Interoperability refers to a program's ability to communicate across environments. This communication can occur at either the user or the program level and can involve processes on the current computer or on remote computers. At the program level, an interoperable program supports ways to move data back and forth between itself and other programs. It might therefore support the pasteboard and be able to read file formats from other programs on either the same or a different platform. It also makes sure that other programs on the system can read the data it creates. Users see interoperability in features such as the pasteboard (the Clipboard in the user interface), drag and drop, AppleScript, Bonjour, and services in the Services menu. All these features provide ways for the user to get data into or out of an application.

5.4.3.1.jMobility

Designing for mobility has become increasingly important as laptop usage soars. A program that supports mobility doesn't waste battery power by polling the system or accessing peripherals unnecessarily, nor does it break when the user moves from place to place, changes monitor configurations, puts the computer to sleep, or wakes the computer up. To support mobility, programs need to be able to adjust to different system configurations, including network configuration changes. Many hardware devices can be plugged in and unplugged while the computer is still running. Mobility-aware programs should respond to these changes gracefully. They should also be sensitive to issues such as power usage. Constantly accessing a hard drive or optical drive can drain the battery of a laptop quickly. Be considerate of mobile users by helping them usetheir computer longer on a single battery charge.

5.4.3.2 Human Interface Design

Good product design incorporates a number of timeless principles for human-computer interaction. This chapter presents these principles for your consideration as you design your product. It also points out what to consider for worldwide compatibility and universal access.

5.4.3.2.a Human Interface Design Principles

- Metaphors
- Reflect the User's Mental Model
- Explicit and Implied Actions
- Direct Manipulation
- User Control
- Feedback and Communication
- Consistency
- WYSIWYG (What You See Is What You Get)
- Forgiveness
- Perceived Stability
- Aesthetic Integrity
- Modelessness
- Managing Complexity in Your Software

5.4.3.2.b Keep Your Users in Mind

In addition to the basic principles of interface design, consider the needs of your audience. Are the users more comfortable in a language other than English? Do they have special needs that might affect the way you present data to them? The following sections identify areas that might influence your design.

- Worldwide Compatibility (Localization)
- Universal Accessibility (Disables)

5.4.3.2.c Extending the Interface

This section contains information on how to determine when it's appropriate to go beyond the guidelines, how to use the existing interface elements to build new elements, and what to avoid when you design additional interface elements.

- Build on the Existing Interface
- Don't Assign New Behaviors to Existing Objects
- Create a New Interface Element Cautiously

5.5 Twelve Questions and Answers (Old)

The article was posted in OSNews in10th Mar 2003 by Eugenia Loli-Queru. It was an interview with Bastian and Aaron J. Seigo from the KDE project and Havoc Pennington from the Gnome project.

The following is the actual article:

This article we host here today is a must read for all Gnome and KDE users. We are happy to feature an exclusive interview with Waldo Bastian and Aaron J. Seigo from the KDE project and Havoc Pennington from the Gnome project. Waldo and Havoc are developers working on many "under the hood" places of their respective DEs, but they are also "sensitive" at UI and usability issues, so we could also call them "usability engineers". Aaron is the head of usability in the KDE project. All three of them were... brave enough to answer twelve hard questions about interoperability, standards, UI etc. between the two leading Unix DEs. Note that this is not a Gnome Vs KDE article, it is in fact exactly the opposite: 'KDE for Gnome' and 'Gnome for KDE'. The beginning of a deeper collaboration and sharing that will bring the Unix desktop into a new era.

1. *Some users want infinite number of options and preferences, while others prefer a non-bloated interface where the best options for them is already decided by the system. Now, we all know that there is no such thing as the "Perfect UI", but would it be acceptable to sacrifice certain configurability and... bloat --with the possible outcome of losing some users-- in order to provide a cleaner interface? Do you think such a move would simplify things for the user or do little but rob power from those who know enough to use it?*

Aaron J. Seigo: Addressing the overall usability of an application or environment in terms of "number of configuration options" is rather naive. Most people use whatever the default settings are and only interact with the configuration systems when the default settings are not what they need or want.

The sad irony is that the people who lose out the most when configuration is limited in an ad-hoc fashion are those who care about configuring their software in the first place. Those who don't are rarely affected one way or the other. Default configurations are therefore a much more interesting and useful discussion as they effect everyone.

In that vein, it is our challenge as technology creators to deliver what our users need and want in a way that actually works; we can't shoehorn people into the mold of our personal comfort zones. In the world of open source development, our usability lab is much closer to our development offices than is generally typical. In such a cooperative and open environment the question to the answer of "how much is enough" manifests itself naturally if you let it. This does not mean effort is unnecessary on the part of developers, but rather that oligarchical scheming and Grand Unified Opinions are.

It disheartens me when I see a "developer-knows-best" attitude as it almost always lead to "too many", "to few" or just "the wrong" options for the users of the software. Listening to and watching our users in action, I observe three types of requests:

- Options, options and more options, please!
- Make it easy to use these options!
- Make the defaults sensible!

I almost never hear from actual users, "Hey, can you remove a bunch of these options?" So that is our mandate: to make it easy to access featureful software. This may not be the easiest path available, but it is the most rewarding. Anyone clammering for across-the-board removal of options is simply not in touch with our user base.

Of course, this does not mean that we should add every single option imaginable or that we should make everything infinitely configurable. Not all options are created equal, and you can oversaturated a system with microconfigurations to the point that it becomes unmanageable. Some options are the result of developer indecision, some are band-aids over bugs that should be fixed, while others are clearly at odds with the overall design of the software. But the majority of options are worthwhile and there because they were wanted or even needed by actual users.

Havoc Pennington: My opinion on this is pretty well known.

It's all about balance, as I say in that article. The question is not whether configuration is good or bad (some people reduce the argument to this), but about which configuration you have, and how you decide. Claiming "all configuration is good" ignores real tradeoffs - not just with simplicity, but also with stability and speed of development.

People miss that GNOME is still more configurable than OS X or Windows, even though I talk about how it's moved the configurability tradeoff back into the realm of sanity. You have to keep things in perspective.

Waldo Bastian: Configuration options have a cost, they make the software more complex and complexity leads to bugs. More configuration options also makes it increasingly difficult to write good user interfaces for them. On the other hand configuration options can help to make the software perform it's task better for specific users. As a developer you have to balance these two conflicting demands, for each extra configuration option you have to ask yourself, is it really needed, does it really improve the overall quality of my application?

If a configuration option adds very little extra value to the application but stands in the way of a usable interface then I think it can be justified to remove it. But that should be a last resort, the first goal of improving a user interface should be to optimize usability while preserving all features. Presenting the available options in a way so that they aren't overwhelming and form a logical and coherent whole.

2. *Microsoft has a widely-used Windows certification standard, and Apple has long been known to strongly encourage developers to follow its guidelines and standards. Has your group considered creating an official "stamp of approval" for applications, assuring potential users that certain rules are obeyed and certain functionality present for the sake of consistency?*

Aaron J. Seigo: Applications that are shipped with KDE are required to follow the KDE guidelines and standards, so obviously the project understands the importance of such things. In fact, a tremendous amount of effort has gone into codifying as much of these standards as possible in the KDE foundation technologies so that it isn't necessary to burden individual developers with the task of compliance. This guarantees users get consistent applications when they are built on KDE.

To me, that's much more impressive and important than a certification system, since it addresses the problem directly rather than poke at the symptoms. This is especially poignant in the Free software world where the motivation for achieving a "certified KDE" sticker would be far less than it would be in the commercial development world.

Havoc Pennington: I think this would be unrealistic right now, because there's no one who would have the substantial time required to do the evaluation work. But it's certainly something we'd like to have as usage of Linux/UNIX on the desktop grows.

Waldo Bastian: There is certainly a strong encouragement to follow guidelines and standards within KDE. Consistency among applications was one of the major driving forces for the creation of the KDE project. We try to achieve that both through the KDE style guide, which was published several years ago already, and by offering standardized solutions through the KDE framework. Especially the latter is very effective.

We have considered creating some form of unofficial "stamp of approval" or at least "rating" wrt standards compliance of applications but so far we haven't been able to find enough manpower to pull that off.

An "official" stap of approval is even harder in that respect since it would put much higher demands on such process with respect to issues as fairness, objectivity and processing time.

3. *FreeDesktop.org is a very useful organization, created to do what in my opinion should have been done years ago... I can't help thinking how things can work out though (in order to unify the two desktops in a way that brings consistency to the Unix desktop), when the two HIGs are not compatible. For example, the OK/Cancel order in a window just to name one and change all the third party apps to comply with any new rules, can be quite time consuming, if not possible. How can the FreeDesktop bring interoperability between the two DEs with such issues at hand?*

Aaron J. Seigo: FreeDesktop.org has already brought KDE and GNOME closer together in terms of interoperability when it comes to things that really matter like the clipboard, so there is no question about it having already been a benefit. But FreeDesktop.org is not a source of magic solutions. There will most likely continue to be differences when it comes to interface details for the foreseeable future. The situation is similar on other contemporary platforms as well, so I don't know if this is really a problem to be too concerned about though it is something I'd like to see addressed over time. This is why I helped start the Open HCI project which is currently in its earliest of days.

Of course we are ultimately at the mercy of the individual development teams and their developers. If one group decides cooperation, consistency and user-centric solutions are more important than achieving their own interpretation of an aesthetic world, then everyone benefits (including the developers). When users and developers support the project(s) which have values similar to their own, this reinforces "good" decision making and removes mobility from those making "poor" decisions. Having more than one project for people to support makes this process fault tolerant.

In other words, grass roots user and developer support is more potent than FreeDesktop.org alone can be in these matters. In turn, FreeDesktop.org needs to reflect those broader interests to be effective.

Semi-off-topic sidebar on the Ok/Cancel issue: KDE has implemented things in such a way that allows all KDE applications to have the order of these buttons flipped with a change to a single line of code. It is exactly this sort of brilliant design that allows KDE to be so internally consistent. So the question often comes down to whether or not we SHOULD make a change, rather than if we CAN. Personally I think it is irresponsible to impose personal aesthetics on your users in a seemingly random fashion by disrupting the interface they know without *very* compelling reasons to do so.

Havoc Pennington: There are certainly benefits to each intermediate step. For example, if all my apps work with the same "recent documents" feature, then that is a big user-visible improvement (bug fix, really). It's a big improvement whether or not all orthogonal problems are also solved.

That said, the shared HIG problem is an interesting one and I'd like to see some progress there. Seth and Aaron are just starting on it so I'm optimistic.

Waldo Bastian: There are always solutions possible. However, instead of focussing on the hardest issues, I think it is much more productive to focus on those areas where results can be achieved relative easily. 100% consistency between the different enironments doesn't come overnight, it will be a long process, but I'm convinced that we will also be able to solve the hard issues somewhere along the road.

4. *Some systems, such as Mac OS X, hide the directory structure from users who don't purposefully look for it. However, products such as Konqueror and Nautilus currently do not do this, and almost flaunt it with address bars in the file manager and so forth. Do you forsee more "hiding" of the Unix underpinnings in the future, or would you prefer to expose users to them? Also, how has Apple's OS X interface influenced you, if at all?*

Aaron J. Seigo: I foresee both happening simultaneously. No to sound too Zen about it, but there is more than one path that leads to the top of the mountain. As there are benefits to various amounts and types of exposition depending on the particular use case, rather than simply choose one level of abstraction we need to decide when and how to abstract away the underlying details and when to allow easy but direct interaction with those details, preferably with the user in control of that decision.

Today there is room for improvement on both ends of that spectrum. So I expect in the future we'll some developments that help insulate the user from gross detail, while other developments work on reflecting the underlying OS more accurately in the interface.

As for OSX's interface, for me it was a lesson in user respect and self-confidence, both good and bad.

It's taught me that no matter how well you've done in the past, you can still make amazingly obvious and stupid mistakes. Everyone is fallible and nobody has all the answers yet. It also showed me that when you take away options you disenfranchise your users: look at how many things were added back into Jaguar due to user demand that had been removed in 10.0.

It's also taught me that eye candy is a more important part of the user experience for most people than I had previously considered. They'll put up with dreadfully slow start up times and all sorts of "dot-oh" interface blunders if it looks like a visual masterpiece.

Finally, it showed that you can introduce radical new concepts into an interface that have real benefits (such as dialog sheets) and users will grok them even if they aren't available in other popular interfaces. So we shouldn't be too afraid to introduce new UI concepts.

Havoc Pennington: I think more hiding is generally appropriate, yes. In star trek future, something better than a hierarchical filesystem might be nice. In the meantime, users don't want to see /usr, in the general/default case.

Regarding Aqua, when doing a new UI component, we generally survey existing UIs in that area; for example here you can read about the new GTKFileSelection.

So OS X usually factors into those kinds of discussion.

Waldo Bastian: I see it as a similar issue as wit the command line. Users shouldn't need to know about it but it should be available to them when they so desire.

5. *What are a few things you like and dislike about the Windows XP interface? Additionally, Microsoft is pursuing the "task-based" interface. What do you think of it and how well could that work in practice?*

Aaron J. Seigo: I've used Windows XP for a scant total of 2 days, so I'm anything but an authority on Windows XP. Due to the closed nature of Windows XP and the fact that it doesn't provide me with the features I need nor run the applications I use I don't have any need for it.

I will say that during my brief time with the system, I found the file manager pleasant to look at but horrible to use: it imposes too much of itself on the user without offering enough in return. Most laughably, I could not move a music file from the desktop while it was being viewed in the file manager. I would suggest to Microsoft that they fix those sorts of problems first before spending any more time on their embedded photo album view. The new start menu is also an abomination. In fact, those two days with Windows XP reminded me why most people hate computers. I'd hate them too if that was all that was out there.

Like any potentially successful methodology, task based mechanisms have their place and can be very effective when applied to an appropriately task based problem. Task based interfaces can be quite good but they can also be quite out of place and/or implemented poorly. I really don't know how XP stacks up in that regard.

Havoc Pennington: XP seems to have lots of nice small details, but is overall a bit too complex. The task-based interface looks interesting, but I've never tried it out or seen a detailed analysis of how it works.

Waldo Bastian: I am not familar with Windows XP.

6. *Recently, we have seen a few attempts to help users better store and organize their files, and there have been two extremely different approaches. The approach taken by the BeOS, among others, was to provide a general framework from which the user could organize files in rather powerful ways (by using file system attributes). The other approach, taken most strikingly by Apple, is to provide filetype-specific applications to manage the files (such as iTunes managing a database of all your music, and iPhoto your photos, etc.). Which do you think is the better approach, and to what extent, if any, do you see [your project] adopting those practices?*

Aaron J. Seigo: You're asking whether metadata management belongs at the OS level or on the application level. The benefit of it happening at the filesystem level is that it can be fast and universally available to all applications, but at the expense of portability and flexibility. Leaving it to the applications allows more flexibility, portability and choice but makes sharing the results between applications harder. I think both of these approaches leave much to be desired.

Another approach is a hybrid class of program that sits below the user application level but above the filesystem. This allows choice, availability and portability. Such a program would offer collections of data from and about the filesystem and other programs would provide an interface from which to view and interact with these collections. The inklings of this can be seen in the KDE KParts and KIOSlave technologies. There is more to be done, of course, but it is the most promising approach I've yet to see in actual use.

Havoc Pennington: I think GNOME is following the Apple approach at the moment, though we also do have "emblems" and other file system attributes, they are underused.

My intuition would be that most users are a bit lost by the filesystem concept, and making it even more complicated than files/folders doesn't necessarily help. While a dedicated application can do a lot of nice things automatically and is based on the task the user is trying to do - play music, organize/retouch their photos.

7. *The biggest problem I personally see today with all the X11 DEs when compared to OSX, BeOS, OS/2 and Windows, is the pretty much non-existent integration to the underlying system. No X11 DE "truly knows" about the system, its drivers and configuration, or how to deal with them, as every distribution/Unix has its own way of making the OS work. Is there a way to bring these DEs in a status where they "understand" the system and provide preferences/settings or application*

behavior/features that normally the distribution would have to provide? For example, there is a "mouse" panel under Gnome's preferences configuring the options for the mouse driver used, but there is also "mouse" panel under the "system settings" Red Hat menu, this time about the driver itself. This is a necessary additional step from the distro maker to add more similar panels, as Gnome and KDE don't know how to deal with the system itself. But in order to create a low-bloat, consistent and integrated interface that makes the usage of the OS easy, the DEs will have to learn about the system and use it accordingly. What is your opinion on this, and how could this be achieved when we have so many different and sometimes incompatible distributions (despite LSB), while not even counting the BSDs, Solaris, IRIX and AIX ports.

Aaron J. Seigo: Underlying this is the same issue underlying question #9, which I've answered below.

Havoc Pennington: I believe this can be done, but it is highly nontrivial and no one has really attacked it with the level of seriousness required.

The first task is to figure out how to present the root/user split, and the idea that some settings will affect all users and some will affect only yourself. The coward's way out, in my view, is to just have people log in as root. The reason I think this is bad is that it throws out a big reason Linux is *better* than Windows, our competitive advantages: manageability and security. If people are logged in as root they can break the system much more seriously (reducing stability), as well. So I think it's OK to ask users to understand the multiuser nature of the OS on some level; but I'm not sure what the best way to present it is.

After figuring out the UI we want, it's more or less a simple matter of coding. I'm hoping that some new technology such as D-BUS will be helpful here. It will give us a way to pop up a dialog triggered by kernel events, for example - something we haven't really had in the past.

Waldo Bastian: I think system integration is a very weak point of Linux. On one hand there is the fragmentation in terms of user inteface (GNOME, KDE) and on the other hand there is the fragmentation in terms of distributions which all tend to do things differently (not to mention non-Linux operating systems)

The result is that system integration comes mostly down on the distributor.

There have been several attempts to create "standard" configuration tools by different groups but none of those have been widely adopted across distributions. I'm pessimistic about this area since I don't think the current mainstream distro's are willing to change this given the investments they have done in their own set of tools.

8. *Both DEs (especially KDE) come with a large number of applications to add in the mix. This is convenient for the user, but do you find it necessary adding applications with each release? What about the issue of including applications that do similar things? (e.g. Kate, Kedit, KWrite).*

Aaron J. Seigo: I've never heard people worry about having too much application choice for so little cost before. In fact, I distinctly remember a time when the complaint was exactly the opposite. It amazes me what people will choose to worry about.

A desktop is useless unless it enables you to get your work done, therefore it should be our aim to provide people with as complete a solution set as defined by the general needs of the userbase (as oppose to, say, our personal opinion). Nobody is required to use or even install every available application included with KDE, but unless they are available the environment is that much less attractive and useful to at least some segment of the user base. The proliferation of quality applications that fulfill real world user needs is vital to our success. The importance of this can not be understated.

Outside of the official KDE distribution, competition between different efforts can be a great things. Cooperation can often be even better, of course. But within the base KDE distribution duplication of features is kept to a minimum and generally frowned upon.

You offer the example of KEdit, KWrite and Kate, but all of those apps do something quite different. KEdit simply edits plain text files, nothing more and nothing less. KWrite is a source code editor for programmers while Kate is a light-weight IDE. So why do all three exist? Because there are three different types of users and use cases which are best served by each. This is the difference between user driven design and "developer knows best" design.

Havoc Pennington: I pushed for GNOME to come in small modular tarballs, and generally I like to see independent, healthy communities writing each GNOME app in parallel. GTK+, AbiWord, Gnumeric, etc. all have large communities of developers with release schedules that aren't tied to GNOME.

Originally this made it kind of annoying for users to build GNOME, but we've addressed that via tools such as jhbuild and GARNOME. Plus most people use a distribution version anyway.

My feeling is that if an app has to come with GNOME in order to get it properly integrated with the desktop, we have a model that's not scalable; it will have to bog down at some point as the Linux desktop becomes more successful and there are more and more apps. A scalable model is based on documentation, guidelines, APIs that anyone can pick up and use without having to release in sync with GNOME proper.

So I like having the GNOME project itself focus on the "desktop and developer platform" release which is just the libraries and the desktop shell, more or less. But we also have the GNOME Office release, and the "fifth toe" add-on apps release, which are on separate release cycles.

Waldo Bastian: Just because KDE releases an application doesn't mean you actually have to install it ;-) Until recently we used to put applications in large modules and those tended to end up in single binary packages. The reason for that was that that is easier to manage from a development point of view, and our users liked the limited number of packages they had to download to get a complete set of applications.

Since last year we have changed that course somewhat, we keep the large modules that we have as much as possible but new applications are now mostly added to "KDE Extra Gear". The extra gear applications are supposed to be released according to their own schedule and packaged as single binary package so that users can choose per application whether to install it or not.

9. *Despite the advancements of RPM handling, apt-get from Debian and Gentoo's Portage, users are still not comfortable downloading applications and easily installing them. Either dependency hell (RPM) when downloading apps from the web, bad interfaces for apt-get (Synaptic is not what I would call "niiice") while Gentoo itself is a nightmare to install for new users, makes the installation of... random Linux applications pretty impossible for new users. With all the advancements recently, this domain is still not as easy as in Windows, OSX or BeOS. Do you think that the DEs themselves should require a special packaging (doesn't have to be a new technology or something different than RPM or apt) that somehow eliminates the current problems and adds visual un-installation, ability to install a package without the need to be the administrator, or automatically categorize the installed application etc? In essence, could the two leading DEs "force" the Linux distros towards a common standard which will be modified in a way that eliminates most of the problems mentioned above?*

Aaron J. Seigo: This question is applying old-style thinking to problems in a new space. There is a near zero chance of KDE and GNOME, even together, forcing anything on operating systems integrators if they don't want it. The Linux distributions already modify KDE to their will, sometimes with good results and other times to the detriment of everyone involved.

While this lack of leverage on the part of KDE and GNOME may seem like impotence, it actually is a strength since it allows divergent systems to use the same interface. Thanks to this agnostic principle I can use KDE on any Linux, BSD or UNIX (including MacOS X) I wish.

The real problem is not with KDE, since it is actually the perfect incubator for safe cooperation on the UI level between the OS vendors. And *that* is where the real problem lies. The OS vendors

feel it is perfectly allright to modify (or "fork") the desktops to suit their own needs so as to create artificial benefits over their competitors. They should instead realize that KDE represents an opportunity to work together within a larger vendor neutral community to create consistent and more compelling systems at a lower cost to each of them individually. Instead of differentiating each from the other, they should be differentiating together from the real competitors: Microsoft Windows and MacOS X.

This if Free Software 101, and I am completely baffled why the likes of Red Hat, Mandrake and SuSe don't seem to get it. Perhaps because the desktop space is a new-ish phenomenon in the Open Source world, since they deffinitely get it when it comes to server and systems software.

Havoc Pennington: I don't think this is feasible. For one thing, a packaging system is an extremely nontrivial undertaking. For another, there is no way you could move existing distributions off their current systems. Finally, it doesn't make a lot of sense to have two packaging systems at once (the distribution one, and the desktop one).

The road I think we have to take instead, is to add better integration with RPM/dpkg/etc. - possibly extending those to give the UI the information it needs to make things nice.

Another key issue in this area is that we have to make the LSB a reality, so software can be provided in distribution-neutral RPM packages as described in the LSB. Right now, most software comes as source code, or as a tarball with binaries, because that's the only cross-distribution method. And obviously that distribution method is hopeless from a UI standpoint.

Waldo Bastian: KDE limits itself to providing source code and most of our binary packages are provided as courtesy by the major distributions (with Red Hat being a notable exception), as such I don't think we are in any position to set a standard wrt binary packages.

10. *Windows Explorer was among the first file managers which tried to be more than just that. Nautilus does a lot these days, but Konqueror has surpassed all, adding a lot of functionality and adding literally the kitchen sink. Do you believe that this is the way to go in the future for the file managers or does it lead to bloat and to unnecessary functionality (functionality that could be served by individual apps)?*

Aaron J. Seigo: Konqueror is not a single application. It is an interface through which hundreds of individual components cooperate to offer a full user experience. If that sounds familiar, it's because it is: the UNIX shell does the same thing for command line interfaces. Where Konqueror goes one step further is that the functionality it offers "just works" based on context without the user having to explicitly command it to do something special, although the user is still fully empowered to do just that.

The vast majority of the functionality in Konqueror is either provided by or available to separate applications, so this is not an either/or situation. You can do many of the same things from within Konqueror as well as in separate apps, because they're the same thing in KDE. There is no distinction between an application and a component for viewing or editing.

This march towards radical componentization of the desktop will only continue. Already we have reached the point where people, even those who are quite aware of interface issues and design, stop distinguishing between individual applications and instead simply experience the desktop as a coherent entity doing a lot of great and cool things.

Sounds like a very compelling desktop computing paradigm to me.

Havoc Pennington: It depends. Having views such as a "photo album" in the file manager makes sense to me. I don't think having the web browser in there necessarily does; my file manager and web browser just don't have that much overlap (in Windows, IE hardly looks like the file manager, even though I guess they share code in some way). Epiphany and Nautilus are both quite clean and simple alone but the merger would be sort of a mess.

Waldo Bastian: I think it is Mozilla that has a kitchen sync, I haven't seen one in Konqueror yet.

Konqueror has a very modular design so it is possible to add lots of optional functionality without bloating it. On the other hand I have always advocated that Konqueror is a browser and functionality that doesn't fall in that category doesn't belong in Konqueror.

11. *Personally, I see Linux getting a big boost in the UI and desktop experience via the "one DE". Having more than one, gives the user choice of course, but it can be inconsistent and prone to incompatibilities between the various distros and Unices. The "less is more" approach has been taken by Be, Apple and MS with quite some success judging from the desktop experience they offer, so I kind of lean in this direction. Now, I don't mean to dismiss any of the two DEs, in fact, I would like to see both thriving, but what I would really want is to have a single desktop, with absolutely compatible development frameworks, one based on GTK+/gnome_libs and one on Qt/kde_libs (and why not add more to the mix? Development choice is good). Pretty much, is like saying that we have the Win32 API and the MFC API and the .NET API, but at the end, no matter with what you compiled or developed your application with, the end result application will look and behave the same as all the other ones, under this "one" DE. Do you agree with the prospect of such a "unification", or does it sound too extreme (or simply "the OSS world is not ready for something like it yet, if ever")?*

Aaron J. Seigo: I have a unified desktop. Everything looks and works the same and I get everything I need and want done accomplished by using only KDE and command line applications. I may well be able to do the same with a GNOME-only desktop. I understand that not everyone has all their needs met in such a way yet, but that just means we need more applications not that we need to destroy all hints of variety between the applications.

Or should MacOS and BeOS and Microsoft Windows all become a single interface, too? The way I look at it is that the situation on Linux is like being able to run both MacOS and Microsoft Windows on the same machine simultaneously. Yes, the applications are different looking and I can choose to stick with one set of apps only, but I have the CHOICE to do so rather than being forced to do so.

Are there people stubbing their toes and falling into comas because of this choice? Or is diversity allowing us to explore the problem space more effectively and to create an atmosphere of enjoyable and mutual coopetition?

Havoc Pennington: I believe it's a perfectly achievable goal for apps written with either GTK+ or Qt to nicely integrate with either desktop environment.

My hope for what will happen is that more and more infrastructure bits (such as fontconfig, or the menu system) will become shared over time, until basically what we have is two flavors of API (GTK+ and Qt) for developers with different tastes, but desktop integration is using the same mechanisms regardless.

At that point I'm quite sure we'll always have lots of desktop environments (you're forgetting XFCE, ROX, and many others). I mean, look at the list of window managers.

However, one or the other may come to be the dominant environment. It's hard to predict that sort of thing. I don't think the OSS world will get to decide on this issue; I think it'll be more of a market decision. Since we really define "dominance" as "having most of the users."

Waldo Bastian: I think consistency among applications is very important. It has been one of the founding principles of KDE. The original thought was that it was achievable by offering a very compelling, advanced and easy to use framework that everyone could use. The GNOME/KDE split pretty much killed that off though. I think given the current situation, standards to ensure consistency among application based on the two different frameworks is the only viable solution to get consistency on the Linux desktop.

12. *What major changes in user interfaces do you predict we will see in the next five years? What steps is [your project] taking to accommodate this?*

Aaron J. Seigo: Anybody who tells you what their project is doing to revolutionize user interface in five years time is either lying or delusional. I'm more comfortable pointing to what KDE has accomplished in its first 6 years of life, especially its ever increasing pace of development and innovation. I don't know what the average computing interface will look like in five years, but given current trends it is safe to say that KDE will be there and will likely be the definition of leading edge desktop software.

Havoc Pennington: I think they should tend to get simpler. Computers are still too complicated, unreliable, etc.; they should be more appliance-like. There seems to be a lot of consensus on this point, too.

GNOME is generally tending in that direction, though it's all evolutionary, not revolutionary. We are dedicated to time-based releases, every half year or so - we aren't going to go into a cave for 3 years and rewrite everything.

Waldo Bastian: I think desktop user interface technology is very mature and I don't expect any major changes. We have this running gag in KDE that the window manager should implement "(window) focus follows mind", a breakthrough might happen if the hardware people manage to make a reliable "mouse follows mind" device, or at least a convenient working "mouse follows eyes" one. It wouldn't surprise me if it had been developed already for military purposes in which case I don't think it's farfetched to predict that it will become available as consumer product within the next five years as well.

Other than that I think that interesting user interface developments will mostly happen wrt non-desktop related computer products.

Chapter 6: Linux and UX

6.1 What is Linux

Linux is a Unix-like computer operating system. Linux is one of the most prominent examples of free software and open source development: typically all underlying source code can be freely modified, used, and redistributed by anyone.

The name "Linux" comes from the Linux kernel, started in 1991 by Linus Torvalds. The system's utilities and libraries usually come from the GNU operating system, announced in 1983 by Richard Stallman. The GNU contribution is the basis for the alternative name GNU/Linux.

Predominantly known for its use in servers, Linux is supported by corporations such as Dell, Hewlett-Packard, IBM, Novell, Oracle Corporation, Red Hat, and Sun Microsystems. It is used as an operating system for a wide variety of computer hardware, including desktop computers, supercomputers,and video game systems, such as the PlayStation 2 and PlayStation 3, several arcade games, and embedded devices such as mobile phones, routers, and stage lighting systems.

6.2 History

The Linux kernel has been marked by constant growth throughout its history. Since the initial release of its source code in 1991, it has grown from a small number of C files under a license prohibiting commercial distribution to its state in 2007 of about 290 megabytes of source under the GNU General Public License.

6.2.1 Pre-creation

The Unix operating system was conceived and implemented in the 1960s and first released in 1970. Its wide availability and portability meant that it was widely adopted, copied and modified by academic institutions and businesses, with its design being influential on authors of other systems.

In 1983, Richard Stallman started the GNU project with the goal of creating a free UNIX-like, POSIX-compatible operating system. As part of this work, he wrote the GNU General Public License (GPL). By the early 1990s there was almost enough available to create a full operating system. However, the GNU kernel, called Hurd, had failed to attract enough development momentum.

Another free operating system project in the 1980s was the Berkeley Software Distribution (BSD). This was developed by UC Berkeley from the 6th edition of Unix from AT&T. Since AT&T Unix code was contained in BSD, AT&T filed a lawsuit in the early 1990s against the University of Berkeley, which strongly limited the development of BSD and greatly slowed adoption.

MINIX, a Unix-like system intended for academic use, was released by Andrew S. Tanenbaum in 1987. While source code for the system was available, modification and redistribution were restricted. In addition, MINIX's 16-bit design was not well adapted to the 32-bit features of the increasingly cheap and popular Intel 386 architecture for personal computers.

Linux or GNU/Linux?
You've probably noticed that Linux as an operating system is referred to in some cases as "Linux" and in others as "GNU/Linux." The reason behind this is that Linux is the *kernel* of an operating system. The wide range of applications that make the operating system useful are the *GNU software*. For example, the windowing system, compiler, variety of shells, development tools, editors, utilities, and other applications exist outside of the kernel, many of which are GNU software. For this reason, many consider "GNU/Linux" a more appropriate name for the operating system, while "Linux" is appropriate when referring to just the kernel.

Source:
http://www.ibm.com/developerworks/linux/library/l-linux-kernel/

These factors of a lack of a widely-adopted, free kernel provided the impetus for Torvalds's starting his project. He has stated that had either the GNU or 386BSD kernels been available at the time, he likely would not have written his own.

In 1991, in Helsinki, Linus Torvalds began a project that later became the Linux kernel. It was initially a terminal emulator, which Torvalds used to access the large UNIX servers of the university. He wrote the program specifically for the hardware he was using and independent of an operating system because he wanted to use the functions of his new PC with an 80386 processor. Development was done on Minix using the GNU C compiler, which is still the main choice for compiling Linux today (although the code can be built with other compilers, such as the Intel C Compiler).

As Torvalds wrote in his book Just for Fun, he eventually realized that he had written an operating system kernel. On 25 August 1991, he announced this system in a Usenet posting to the newsgroup "comp.os.minix.":

> *Hello everybody out there using minix -*
>
> *I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).*
>
> *I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)*
>
> *Linus (torvalds@kruuna.helsinki.fi)*
>
> *PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.*
>
> *– Linus Torvalds*

6.2.2 The Name

Linus Torvalds had wanted to call his invention Freax, a portmanteau of "freak," "free," and "x," an allusion to Unix. During the start of his work on the system, he stored the files under the name "Freax" for about a half year. Torvalds had already considered the name "Linux," but initially dismissed it as too egotistical.

In order to facilitate development, the files were uploaded to the ftp server (ftp.funet.fi) of the Helsinki University of Technology (HUT) in September 1991. Ari Lemmke, Torvald's coworker at the HUT who was responsible for the servers at the time, did not feel Freax was a good name. Consequently, he dubbed the project "Linux" without consulting Torvalds. Later, however, Torvalds consented to "Linux": "After many arguments, he finally admitted that Linux was simply the better name. In the source code of version 0.01 of Linux, the name 'Freax' was still used in the makefile. Only later was the name Linux used. Thus the name actually not planned at all became generally accepted world-wide.

6.2.3 Linux under the GNU GPL

Torvalds first published the Linux kernel—then exclusively known as Linux—under its own license, which was, essentially, a shared source license with a restriction on commercial activity. With code from the GNU system freely available, it seemed advantageous if this could be used with the Linux kernel. In 1992, he suggested to switch to the GNU General Public License. He first

announced this change in the release notes of version 0.12. In the middle of December 1992 he published version 0.99 using the GNU GPL.

Linux and GNU developers worked to integrate GNU components with Linux to make a fully functional and free operating system.

Torvalds has stated, “Making Linux GPL'd was definitely the best thing I ever did.”

6.2.4 GNU/Linux naming controversy

The designation "Linux" was initially used by Torvalds only for the Linux kernel. The kernel was, however, frequently used together with other software, especially that of the GNU project. This quickly became the most popular adoption of GNU software. In June 1994 in GNU's bulletin, Linux was referred to as a "free UNIX clone", and the Debian project began calling its product Debian GNU/Linux. In May 1996, Richard Stallman published the editor Emacs 19.31, in which the type of system was renamed from Linux to Lignux. This spelling was intended to refer specifically to the combination of GNU and Linux, but this was soon abandoned in favor of "GNU/Linux".

This name garnered varying reactions. The GNU and Debian projects use the name, although most developers simply use the term "Linux" to refer to the combination.

6.2.5 Official mascot

Torvalds announced in 1996 that there would be a mascot for Linux, a penguin. Larry Ewing provided the original draft of today's well-known mascot based on this description. The name Tux was suggested by James Hughes as derivative of Torvalds's UniX.

6.2.6 Kernel

There are many other well-known maintainers for the Linux kernel beside Torvalds such as Alan Cox and Marcelo Tosatti. Cox maintained version 2.2 of the kernel until it was discontinued at the end of 2003. Likewise, Tosatti maintained version 2.4 of the kernel until the middle of 2006. Andrew Morton steers the development and administration of the 2.6 kernel, which was released on 18 December 2003 in its first stable incarnation. Also the older branches are still constantly improved.

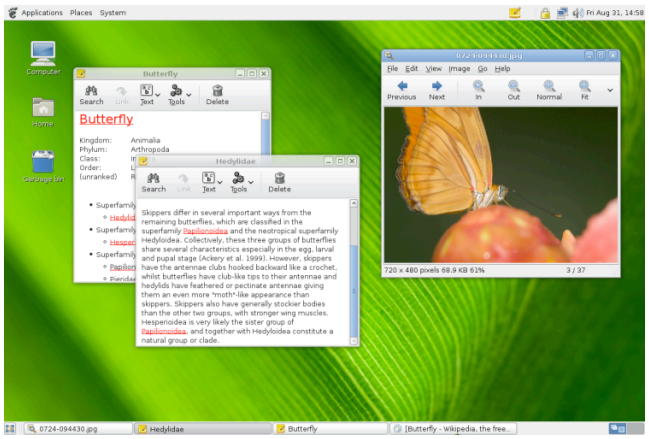
The success of Linux in many areas of application is mostly due to the lack of licensing costs and the characteristics of free software concerning stability, security, expandability and maintenance of leading back.

6.2.7 Community

The Community performs the largest part of the work on Linux, the programmers that use Linux and send their suggested improvements to the maintainers. Various companies have also helped not only with the development of the Kernels, but also with the writing of the body of auxiliary software, which is distributed with Linux.

It is released both by organized projects such as Debian, and by projects connected directly with companies such as Fedora and openSUSE. The members of the respective projects meet at various conferences and fairs, in order to exchange ideas. One of the largest of these fairs is the LinuxTag in Germany (currently in Berlin), where about 10,000 people assemble annually, in order to discuss Linux and the projects associated with it.

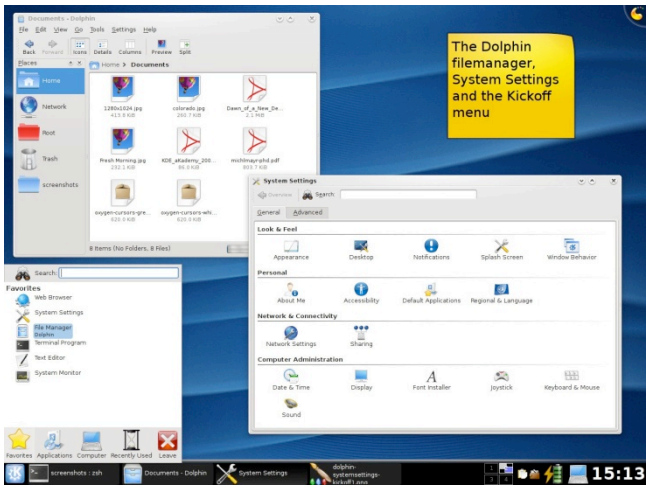
6.2.8 Desktop

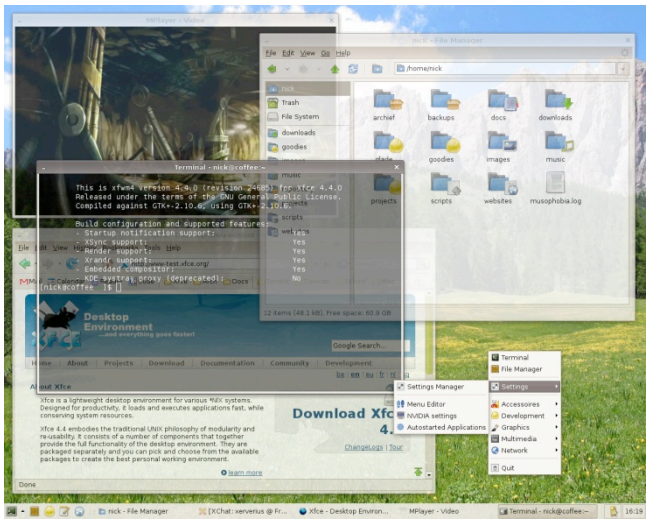


Although there is a lack of Linux ports for some Mac OS X and Microsoft Windows programs in domains such as desktop publishing and professional audio, applications equivalent to those available for Mac and Windows are available for Linux.

Most Linux distributions provide a program for browsing a list of thousands of free software applications that have already been tested and configured for a specific distribution. These free programs can be downloaded and installed with one mouse click and a digital signature guarantees that no one has added a virus or a spyware to these programs.

Many free software titles that are popular on Windows, such as Pidgin, Mozilla Firefox, Openoffice.org, and GIMP, are available for Linux. A growing amount of proprietary desktop software is also supported under Linux, examples being Adobe Flash Player, Acrobat Reader, Matlab, Nero Burning ROM, Opera, RealPlayer, and Skype. In the field of animation and visual effects, most high end software, such as AutoDesk Maya, Softimage XSI and Apple Shake, is available for Linux, Windows and/or Mac OS X. CrossOver is a proprietary solution based on the open source Wine project that supports running older Windows versions of Microsoft Office and Adobe Photoshop versions through CS2. Microsoft Office 2007 and Adobe Photoshop CS3 are known not to work.





Besides the free Windows compatibility layer Wine, most distributions offer Dual boot and X86 virtualization for running both Linux and Windows on the same computer.

Linux's open nature allows distributed teams to localize Linux distributions for use in locales where localizing proprietary systems would not be cost-effective. For example the Sinhalese language version of the Knoppix distribution was available for a long time before Microsoft Windows XP was translated to Sinhalese. In this case the Lanka Linux User Group played a major part in developing the

localized system by combining the knowledge of university professors, linguists, and local developers.

The performance of Linux on the desktop has been a controversial topic, with at least one key Linux kernel developer, Con Kolivas, accusing the Linux community of favoring performance on servers. He quit Linux development because he was frustrated with this lack of focus on the desktop, and then gave a 'tell all' interview on the topic.

6.2.9 Linux Is Not Unix

Linux is frequently referred to as a Unix-like system. We cannot call Linux a Unix system because it has not passed the tests and certifications required by The Open Group to be officially called Unix. Practically speaking, however, Linux is functionally similar to Unix; it was designed to work like Unix and for most purposes it has accomplished that goal. If you know Unix, you'll be comfortable working with Linux; if you want to learn Unix, a good way to do that is to install and work with Linux. If you come from the Windows or Macintosh worlds and don't know either Linux or Unix, the Linux desktop environments KDE and GNOME provide a familiar interface for learning, while making it easy to become productive quickly.

6.2.10 Open Source Development Lab and Linux Foundation

The Open Source Development Lab (OSDL) was created in the year 2000, and is an independent nonprofit organization, which pursues the goal of optimizing Linux for employment in data centers and in the carrier range. It served as sponsored working premises for Linus Torvalds and also for Andrew Morton, until the middle of 2006 when he transferred to Google, which runs on the Linux kernel. Torvalds works full time on behalf the OSDL, developing the Linux Kernels. The noncommercial mechanism of several major companies is financed as Red Hat, Novell, Mitsubishi, Intel, IBM, Dell and HP.

On January 22, 2007, OSDL and the Free Standards Group merged to form The Linux Foundation, narrowing their respective focuses to that of promoting GNU/Linux in competition with Microsoft Windows.

6.2.11 Companies

Despite being open-source a few companies profit from Linux. These companies, most of which are also members of the Open Source Development Lab, invest substantial resources into the advancement and development of Linux, in order to make it suited for various application areas. This includes hardware donations for driver developers, cash donations for people who develop Linux software, and the employment of Linux programmers at the company. Some examples are IBM and HP, which use Linux first of all on their own servers, and Red Hat, which maintains its own distribution. Likewise Trolltech supports Linux by the development and GPL licensing of Qt, which makes the development of KDE possible, and by the employment of some X and KDE developers.

6.2.12 Getting and Installing Linux

The most common ways of getting Linux are to download a distribution from the Internet or to buy a commercial version. Other possibilities are to borrow or copy disks from someone else who downloaded them or to do an installation over the Internet. Most of the general-purpose Linux distributions are quite large, so even with a high-speed connection, you might prefer to buy CDs; if you are on a dial-up connection, it's almost imperative that you get CDs.

Determining which distribution to get often comes down to personal preference. The distribution list on LWN.net has a brief description of each distribution, or you might go to some of the vendor websites and see what they say about their systems. If you have the luxury of time and the curiosity, you can try several and see what you like.

When you are ready to install your system, you will probably have a set of CDs containing the distribution you selected; assuming your system can boot from CD; the included installation program will take over and prompt you through the process.

You don't have to dedicate your computer solely to running Linux. It's very common to install Linux so it coexists with your Windows or Macintosh system; that is known as dual booting. With dual booting, a program known as a boot loader lets you select at boot time which system you want to run.

It's possible, but not necessarily easy, to build your own Linux distribution from scratch. If you are interested in doing that, the Linux From Scratch project provides instructions for building a customized Linux system starting from the source code of each component.

What if you aren't sure you want to commit to Linux, but you'd like to at least try it out? There are distributions like Knoppix or Ubuntu Live that let you run Linux directly from a CD; no installation onto your hard disk is required. As long as you can boot from your CD drive, you can download the distribution and burn it to CD, then reboot your computer, and a Linux desktop will appear and you can begin to work in Linux.

6.3 Why Linux

6.3.1 General Users

6.3.1.1 Open source

Open source is a set of principles and practices on how to write software, the most important of which is that the source code is openly available. The Open Source Definition, which was created by Bruce Perens for the Debian project and is currently maintained by the Open Source Initiative, adds additional meaning to the term: one should not only get the source code but also have the right to use it. If the latter is denied the license is categorized as a shared source license.

6.3.1.a The Open Source Definition

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

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 a royalty or other fee for such sale.

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.

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.

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 for genetic research.

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.

9. License Must Not Restrict 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.

6.3.1.2 Cost

Most of the Linux distributions (distro) are free of cost. These free distros come with a good number of applications, which are free as well. If any user does not want to pay a single penny for the operating system and the applications – Linux is the best choice.

There are some special Linux versions, which are not free. They come with non-free codec, plugins and driver. As they have to pay for those non-free licensing, they put a small amount of money for the distribution. Still the cost is less than Microsoft's Windows or Apple's Mac OS X.

6.3.1.3 Performance

Linux is the performer OS, it takes considerably low amount of resources to run. If any user has a Pentium-II level PC in his storeroom considering it as an antic show piece. Good news for him – Linux can revive the PC's life.

6.3.1.4 Security

Linux do not have virus/spyware/malware comparing to Windows. Almost all of the Linux users are using their Linux based PC without worrying about any virus.

6.3.1.5 Support

Linux community is the biggest community in existence. Besides, most of the free Linux distributors provide instant free technical support. There are official forums, advanced and general users to help any user for free anytie.

6.3.1.6 Customization

Linux is hugely customizable. Everything that a Linux distribution has is customizable. The desktop environment is the one, which users want to customize on any operating system. Linux has a number of desktop environments (KDE, GNOME, Xfce, Fluxbox, Enlightment), whichcan be modified in various ways as the users want.

6.3.2 For This Research

For this research purpose, Linux is the best platform. Following are some important reasons:

6.3.2.1 Legal Issue

Being open source, it is legal to modify.

Free and Available to Download

Being free and available to download, it became easy to download a good number of distributions to check out which distro gives what user experience.

6.3.2.2 Lots of flavor

Only Linux comes in a hundreds of flavors. Depending on its developer and packaging type and desktop environment one distro is different from another distro.

6.3.2.3 Easily customizable

The desktop environment is customizable. The kernel is customizable. If I need to show that a minimized kernel with light desktop environment gives better user experience, with Linux it is possible to show practically.

6.3.2.4 No installation

Most of the Linux distro comes in Live CD version. Which needs no installation to run on a computer. No other operating system comes in Live CD format.

6.4 Linux User Experience

User experience solely depends on the user. Depending on the usage, the type differs. For example, an accountant who uses only one software on his computer and nothing else, does not care which operating system he is using. On the other hand, a hardcore programmer who only uses only terminal to code does not care if the OS has any GUI or not. To provide maximum user experience, a single operating system should provide “options” for all sorts of users.

When it is about Linux, one thing comes first – which version to chose. It is a problem for the users. Linux comes in thousands of distros. For the beginners, choosing from this number is not a easy job to do.

6.4.1 The Linux Kernel and General Users

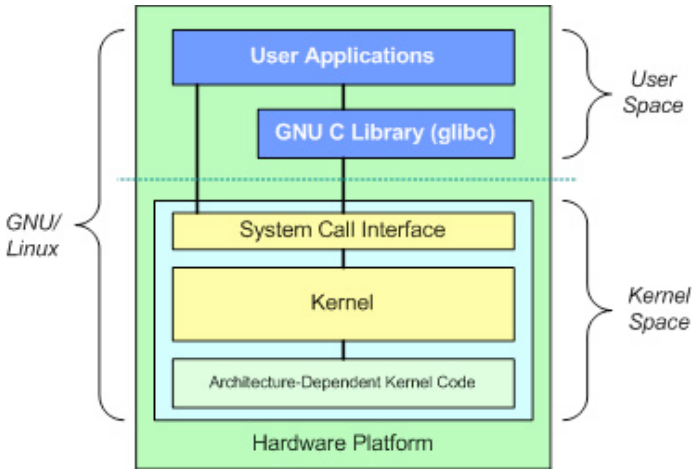
General users do not care about the Linux kernel but the latest kernel update attracts them. Most the users want to use the latest software without knowing what it does. But for the advanced and technical users – Linux kernel is important. They know that kernel is “the” operating system.

To understand the Linux kernel architecture, an article published in IBM developer works by M. Tim Jonesseems easier for all level of users. Here is the main part of it:

6.4.1.1 Introduction to the Linux kernel

Think about an operating system from two levels, as shown in Figure below.

At the top is the user, or application, space. This is where the user applications are executed. Below the user space is the kernel space. Here, the Linux kernel exists.



There is also the GNU C Library (glibc). This provides the system call interface that connects to the kernel and provides the mechanism to transition between the user-space application and the kernel. This is important because the kernel and user application occupy different protected address spaces. And while each user-space process occupies its own virtual address space, the kernel occupies a single address space.

The Linux kernel can be further divided into three gross levels. At the top is the system call interface, which implements the basic functions such as read and write. Below the system call interface is the kernel code, which can be more accurately defined as the architecture-independent kernel code. This code is common to all of the processor architectures supported by Linux. Below this is the architecture-dependent code, which forms what is more commonly, called a BSP (Board Support Package). This code serves as the processor and platform-specific code for the given architecture.

6.4.1.2 Properties of the Linux kernel

When discussing architecture of a large and complex system, you can view the system from many perspectives. One goal of an architectural decomposition is to provide a way to better understand the source, and that's what we'll do here.

The Linux kernel implements a number of important architectural attributes. At a high level, and at lower levels, the kernel is layered into a number of distinct subsystems. Linux can also be considered monolithic because it lumps all of the basic services into the kernel. This differs from a microkernel architecture where the kernel provides basic services such as communication, I/O, and memory and process management, and more specific services are plugged in to the microkernel layer. Each has its own advantages, but I'll steer clear of that debate.

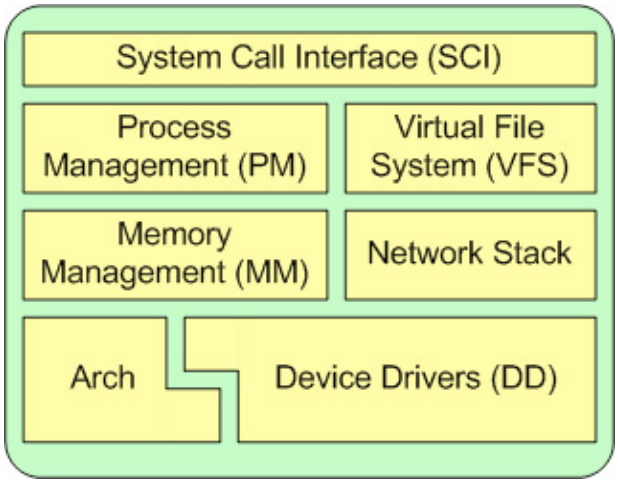
Over time, the Linux kernel has become efficient in terms of both memory and CPU usage, as well as extremely stable. But the most interesting aspect of Linux, given its size and complexity, is its portability. Linux can be compiled to run on a huge number of processors and platforms with different architectural constraints and needs. One example is the ability for Linux to run on a process with a memory management unit (MMU), as well as those that provide no MMU. The uClinux port of the Linux kernel provides for non-MMU support.

6.4.1.3 Major subsystems of the Linux kernel

Some of the major components of the Linux kernel using the breakdown shown in Figure below

6.4.1.3.aSystem call interface

The SCI is a thin layer that provides the means to perform function calls from user space into the kernel. As discussed previously, this interface can be architecture dependent, even within the same processor family. The SCI is actually an interesting function-call multiplexing and demultiplexing service. You can find the SCI implementation in ./linux/kernel, as well as architecture-dependent portions in ./linux/arch.



6.4.1.3.b Process management

Process management is focused on the execution of processes. In the kernel, these are called *threads* and represent an individual virtualization of the processor (thread code, data, stack, and CPU registers). In user space, the term *process* is typically used, though the Linux implementation does not separate the two concepts (processes and threads). The kernel provides an application program interface (API) through the SCI to create a new process (fork, exec, or Portable Operating System Interface [POSIX] functions), stop a process (kill, exit), and communicate and synchronize between them (signal, or POSIX mechanisms).

Also in process management is the need to share the CPU between the active threads. The kernel implements a novel scheduling algorithm that operates in constant time, regardless of the number of threads vying for the CPU. This is called the O(1) scheduler, denoting that the same amount of time is taken to schedule one thread as it is to schedule many. The O(1) scheduler also supports multiple processors (called Symmetric MultiProcessing, or SMP). You can find the process management sources in ./linux/kernel and architecture-dependent sources in ./linux/arch).

6.4.1.3.c Memory management

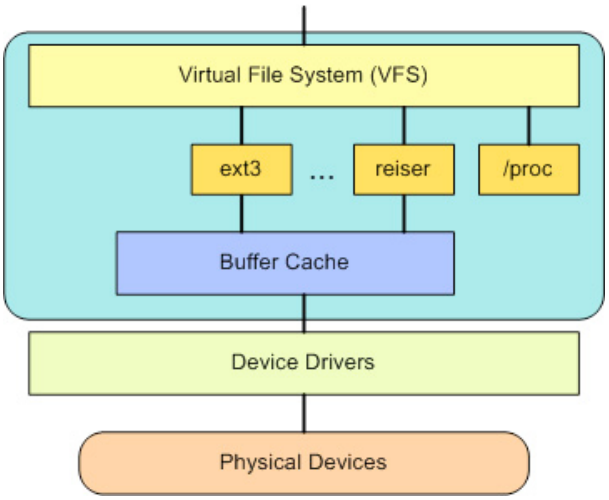
Another important resource that's managed by the kernel is memory. For efficiency, given the way that the hardware manages virtual memory, memory is managed in what are called *pages* (4KB in size for most architectures). Linux includes the means to manage the available memory, as well as the hardware mechanisms for physical and virtual mappings.

But memory management is much more than managing 4KB buffers. Linux provides abstractions over 4KB buffers, such as the slab allocator. This memory management scheme uses 4KB buffers as its base, but then allocates structures from within, keeping track of which pages are full, partially used, and empty. This allows the scheme to dynamically grow and shrink based on the needs of the greater system.

Supporting multiple users of memory, there are times when the available memory can be exhausted. For this reason, pages can be moved out of memory and onto the disk. This process is called *swapping* because the pages are swapped from memory onto the hard disk. You can find the memory management sources in ./linux/mm.

6.4.1.3.d Virtual file system

The virtual file system (VFS) is an interesting aspect of the Linux kernel because it provides a common interface abstraction for file systems. The VFS provides a switching layer between the SCI and the file systems supported by the kernel.



At the top of the VFS is a common API abstraction of functions (see figure) such as open, close, read, and write. At the bottom of the VFS are the file system abstractions that define how the upper-layer functions are implemented. These are plug-ins for the given file system (of which over 50 exist). You can find the file system sources in ./linux/fs.

Below the file system layer is the buffer cache, which provides a common set of functions to the file system layer (independent of any particular file system). This caching layer optimizes access to the physical devices by keeping data around for a short time (or speculatively read ahead so that the data is available when needed). Below the buffer cache are the device drivers, which implement the interface for the particular physical device.

6.4.1.3.e Network stack

The network stack, by design, follows a layered architecture modeled after the protocols themselves. Recall that the Internet Protocol (IP) is the core network layer protocol that sits below the transport protocol (most commonly the Transmission Control Protocol, or TCP). Above TCP is the sockets layer, which is invoked through the SCI.

The sockets layer is the standard API to the networking subsystem and provides a user interface to a variety of networking protocols. From raw frame access to IP protocol data units (PDUs) and up to TCP and the User Datagram Protocol (UDP), the sockets layer provides a standardized way to manage connections and move data between endpoints. You can find the networking sources in the kernel at ./linux/net.

6.4.1.3.f Device drivers

The vast majority of the source code in the Linux kernel exists in device drivers that make a particular hardware device usable. The Linux source tree provides a drivers subdirectory that is further divided by the various devices that are supported, such as Bluetooth, I2C, serial, and so on. You can find the device driver sources in ./linux/drivers.

6.4.1.3.g Architecture-dependent code

While much of Linux is independent of the architecture on which it runs, there are elements that must consider the architecture for normal operation and for efficiency. The ./linux/arch subdirectory defines the architecture-dependent portion of the kernel source contained in a number of subdirectories that are specific to the architecture (collectively forming the BSP). For a typical desktop, the i386 directory is used. Each architecture subdirectory contains a number of other subdirectories that focus on a particular aspect of the kernel, such as boot, kernel, memory management, and others. You can find the architecture-dependent code in ./linux/arch.

6.4.1.4 Interesting features of the Linux kernel

If the portability and efficiency of the Linux kernel weren't enough, it provides some other features that could not be classified in the previous decomposition.

Linux, being a production operating system and open source, is a great test bed for new protocols and advancements of those protocols. Linux supports a large number of networking protocols, including the typical TCP/IP, and also extension for high-speed networking (greater than 1 Gigabit Ethernet [GbE] and 10 GbE). Linux also supports protocols such as the Stream Control Transmission Protocol (SCTP), which provides many advanced features above TCP (as a replacement transport level protocol).

Linux is also a dynamic kernel, supporting the addition and removal of software components on the fly. These are called dynamically loadable kernel modules, and they can be inserted at boot when they're needed (when a particular device is found requiring the module) or at any time by the user.

A recent advancement of Linux is its use as an operating system for other operating systems (called a hypervisor). Recently, a modification to the kernel was made called the Kernel-based Virtual Machine (KVM). This modification enabled a new interface to user space that allows other operating systems to run above the KVM-enabled kernel. In addition to running another instance of Linux, Microsoft® Windows® can also be virtualized. The only constraint is that the underlying processor must support the new virtualization instructions.

6.4.2 Distributions

When Linus Torvalds first developed Linux back in August of 1991, the operating system basically consisted of his kernel and some GNU tools. With the help of others Linus added more and more tools and applications.

In 2008, there are over 500 stable Linux distributions. The positive impact of having too many distribution is users have the option to choose from a huge number of selections. The negative impact is, which is very common – users are confused which one to choose.

There are some mainstream distributions (also known as major distributions). Most of the Linux distributions known as “minor distribution” are derived from the major distributions. Some major distributions are:

6.4.2.1 Gentoo

The concept of Gentoo Linux was devised in around the year 2000 by Daniel Robbins, a former Stampede Linux and FreeBSD developer. It was the author's exposure to FreeBSD and its "autobuild" feature called "ports", which inspired him to incorporate some of the FreeBSD software management principles into Gentoo under the name of "portage". The idea was to develop a Linux distribution that would allow users to compile the Linux kernel and applications from source code directly on their own computers, thus maintaining a highly optimized and always up-to-date system. By the time the project released its 1.0 version in March 2002, Gentoo's package management was considered a superior alternative to some binary package management systems, especially the then widely-used RPM.

Gentoo Linux was designed for power users. Originally, the installation was cumbersome and tedious, requiring hours or even days of compiling on the command line to build a complete Linux distribution; however, in 2006 the project simplified the installation procedure by developing an installable Gentoo live CD with a point-and-click installer. Besides providing an always up-to-date set of packages for installation with a single command, the other important features of the distribution are excellent security, extensive configuration options, support for many architectures, and the ability to keep the system up-to-date without re-installing. The

Gentoo documentation was repeatedly labeled as the best online documentation of any distribution.

Gentoo Linux has lost much of its original glory in recent years. Some Gentoo users have come to a realization that the time-consuming compiling of software packages brings only marginal speed and optimization benefits. Ever since the resignation of Gentoo's founder and benevolent dictator from the project in 2004, the newly established Gentoo Foundation has been battling with lack of clear directions and frequent developer conflicts, which resulted in several high-profile departures of well-known Gentoo personalities. It remains to be seen whether Gentoo can regain its innovative qualities of the past or whether it will slowly disintegrate into a loose collection of personal sub-projects lacking clearly defined goals.

6.4.2.2 Debian

Debian GNU/Linux was first announced in 1993. Its founder, Ian Murdock, envisaged the creation of a completely non-commercial project developed by hundreds of volunteer developers in their spare time. With skeptics far outnumbering optimists at the time, it was destined to disintegrate and collapse, but the reality was very different. Debian not only survived, it thrived and, in less than a decade, it became the largest Linux distribution and possibly the largest collaborative software project ever created!

The following numbers can illustrate the success of Debian GNU/Linux. Over 1,000 volunteer developers develop it, its software repositories contain more than 20,000 packages (compiled for 11 processor architectures), and it is responsible for inspiring over 120 Debian-based distributions and live CDs. These figures are unmatched by any other Linux-based operating system. The actual development of Debian takes place in three main branches (or four if one includes the bleeding-edge "experimental" branch) of increasing levels of stability: "unstable" (also known as "sid"), "testing" and "stable". This progressive integration and stabilisation of packages and features, together with the project's well-established quality control mechanisms, has earned Debian its reputation of being one of the best-tested and most bug-free distributions available today.

However, this lengthy and complex development style also has some drawbacks: the stable releases of Debian are not particularly up-to-date and they age rapidly, especially since new stable releases are only published once every 1 - 3 years. Those users who prefer the latest packages and technologies are forced to use the potentially buggy Debian testing or unstable branches. The highly democratic structures of Debian have led to controversial decisions and gave rise to infighting among the developers. This has contributed to stagnation and reluctance to make radical decisions that would take the project forward.

6.4.2.3 Ubuntu

The launch of Ubuntu was first announced in September 2004. Although a relative newcomer to the Linux distribution scene, the project took off like no other before, with its mailing lists soon filled in with discussions by eager users and enthusiastic developers. In the few years that followed, Ubuntu has grown to become the most popular desktop Linux distribution and has greatly contributed towards developing an easy-to-use and free desktop operating system that can compete well with any proprietary ones available on the market.

What was the reason for Ubuntu's stunning success? Firstly, Mark Shuttleworth, a charismatic South African multimillionaire, a former Debian developer and the world’s second space tourist, whose company, the Isle of Man-based Canonical Ltd, is currently financing the project, created the project. Secondly, Ubuntu had learnt from the mistakes of other similar projects and avoided them from the start - it created an excellent web-based infrastructure with a Wiki-style documentation, creative bug-reporting facility, and professional approach to the end users. And thirdly, thanks to its wealthy founder, Ubuntu has been able to ship free CDs to all interested users, thus contributing to the rapid spread of the distribution.

On the technical side of things, Ubuntu is based on Debian "Sid" (unstable branch), but with some prominent packages, such as GNOME, Firefox and OpenOffice.org, updated to their latest

versions. It has a predictable, 6-month release schedule, with an occasional Long Term Support (LTS) release that is supported with security updates for 3 - 5 years, depending on the edition (non-LTS release are supported for 18 months). Other special features of Ubuntu include an installable live CD, creative artwork and desktop themes, migration assistant for Windows users, support for the latest technologies, such as 3D desktop effects, easy installation of proprietary device drivers for ATI and NVIDIA graphics cards and wireless networking, and on-demand support for non-free or patent-encumbered media codecs.

Ubuntu is a Debian derived distribution, but because of its popularity and the number of Ubuntu derived distributions it became a major distribution in no time. Now around 50 well known distributions are made based on Ubuntu including Linux Mint, gOS.

6.4.2.4 Slackware

Slackware Linux, created by Patrick Volkerding in 1992, is the oldest surviving Linux distribution. Forked from the now-discontinued SLS project, Slackware 1.0 came on 24 floppy disks and was built on top of Linux kernel version 0.99pl11-alpha. It quickly became the most popular Linux distribution, with some estimates putting its market share to as much as 80% of all Linux installations in 1995. Its popularity decreased dramatically with the arrival of Red Hat Linux and other, more user-friendly distributions, but Slackware Linux still remains a much-appreciated operating system among the more technically-oriented system administrators and desktop users.

Slackware Linux is a highly technical, clean distribution, with only a very limited number of custom utilities. It uses a simple, text-based system installer and a comparatively primitive package management system that does not resolve software dependencies. As a result, Slackware is considered one of the cleanest and least buggy distributions available today - the lack of Slackware-specific enhancements reduces the likelihood of new bugs being introduced into the system. All configuration is done by editing text files. There is a saying in the Linux community that if you learn Red Hat, you'll know Red Hat, but if you learn Slackware, you'll know Linux. This is particularly true today when many other Linux distributions keep developing heavily customized products to meet the needs of less technical Linux users.

While this philosophy of simplicity has its fans, the fact is that in today's world, Slackware Linux is increasingly becoming a "core system" upon which new, custom solutions are built, rather than a complete distribution with a wide variety of supported software. The only exception is the server market, where Slackware remains popular, though even here, the distribution's complex upgrade procedure and lack of officially supported automated tools for security updates makes it increasingly uncompetitive. Slackware's conservative attitude towards the system's base components means that it requires much manual post-installation work before it can be tuned into a modern desktop system.

6.4.2.5 SuSE

The beginnings of openSUSE date back to 1992 when four German Linux enthusiasts -- Roland Dyroff, Thomas Fehr, Hubert Mantel and Burchard Steinbild -- launched the project under the name of SuSE (Software und System Entwicklung) Linux. In the early days, the young company sold sets of floppy disks containing a German edition of Slackware Linux, but it wasn't long before SuSE Linux became an independent distribution with the launch of version 4.2 in May 1996. In the following years, the developers adopted the RPM package management format and introduced YaST, an easy-to-use graphical system administration tool. Frequent releases, excellent printed documentation, and easy availability of SuSE Linux in stores across Europe and North America resulted in growing popularity of the distribution.

SuSE Linux was acquired by Novell, Inc. in late 2003. Major changes in the development, licensing and availability of SUSE Linux followed shortly afterwards - YaST was released under the General Public License, the ISO images were freely distributed from public download servers, and, most significantly, the development of the distribution was opened to public participation for the first time ever. Since the launch of the openSUSE project and the release of version 10.0 in October 2005, the distribution became completely free in both senses of the word. The openSUSE code

has become a base system for Novell's commercial products, first named as Novell Linux, but later renamed to SUSE Linux Enterprise Desktop and SUSE Linux Enterprise Server.

Today, openSUSE has a large following of satisfied users. The principal reason for openSUSE getting high marks from its users are pleasant and polished desktop environments (KDE and GNOME), excellent system administration utility (YaST), and, for those who buy the boxed edition, some of the best printed documentation available with any distribution. However, the recent deal between Novell and Microsoft, which apparently concedes to Microsoft's argument that it has intellectual property rights over Linux, has resulted in a string of condemnation by many Linux personalities and has prompted some users to switch distributions. Although Novell has downplayed the deal and Microsoft has yet to exercise any rights, this issue remains a thorn in the side of the otherwise very community-friendly Linux Company.

6.4.2.6 Fedora (Red Hat)

Although Fedora was formally unveiled only in September 2004, its origins effectively date back to 1995 when it was launched by two Linux visionaries -- Bob Young and Marc Ewing -- under the name of Red Hat Linux. The company's first product, Red Hat Linux 1.0 "Mother's Day", was released in the same year and was quickly followed by several bug-fix updates. In 1997, Red Hat introduced its revolutionary RPM package management system with dependency resolution and other advanced features which greatly contributed to the distribution's rapid rise in popularity and its overtaking of Slackware Linux as the most widely-used Linux distribution in the world. In later years, Red Hat standardized on a regular, 6-month release schedule.

In 2003, just after the release of Red Hat Linux 9, the company introduced some radical changes to its product line-up. It retained the Red Hat trademark for its commercial products, notably Red Hat Enterprise Linux, and introduced Fedora Core, a Red Hat-sponsored, but community-oriented distribution designed for the "Linux hobbyist". After the initial criticism of the changes, the Linux community accepted the "new" distribution as a logical continuation of Red Hat Linux. A few quality releases was all it took for Fedora to regain its former status as one of the best-loved operating systems on the market. At the same time, Red Hat quickly became the biggest and most profitable Linux company in the world, with an innovative product line-up and other interesting initiatives, such as its Red Hat Certified Engineer (RHCE) certification program.

Although Fedora's direction is still largely controlled by Red Hat, Inc.and the product is sometimes seen -- rightly or wrongly -- as a test bed for Red Hat Enterprise Linux, there is no denying that Fedora is one of the most innovative distributions available today. Its contributions to the Linux kernel, glibc and GCC are well-known and its more recent integration of SELinux functionality, Xen virtualization technologies and other enterprise-level features are much appreciated among the company's customers. On a negative side, Fedora still lacks a clear desktop-oriented strategy that would make the product easier to use for those beyond the "Linux hobbyist" target.

6.4.2.7 Mandriva (Mandrake)

Mandriva Linux was launched by Gaël Duval in July 1998 under the name of Mandrake Linux. At first, it was just a re-mastered edition of Red Hat Linux with the more user-friendly KDE desktop, but the subsequent releases also added various user-friendly touches, such as a new installer, improved hardware detection, and intuitive disk partitioning utility. As a result of these enhancements, Mandrake Linux flourished. After attracting venture capital and turning into a business, the fortunes of the newly established MandrakeSoft fluctuated widely between a near bankruptcy in early 2003 to a flurry of acquisitions in 2005. The latter, after merging with Brazil's Conectiva, saw the company change its name to Mandriva.

Mandriva Linux is primarily a desktop distribution. Its best loved features are cutting edge software, superb system administration suite (DrakConf), excellent implementation of its 64-bit edition, and extensive internationalization support. It had an open development model long before many other popular distributions, with intensive beta testing and frequent stable releases. In recent years, it has also developed an array of installable live CDs and has launched Mandriva Flash - a complete Mandriva Linux system on a bootable USB Flash device.

Despite the technical excellence, Mandriva Linux has been losing momentum in recent years. This has partly to do with the emergence of other user-friendly distributions that have caught up with Mandriva, but also with some controversial decisions by the company which have alienated a large sector of the distribution's user base. Mandriva's web presence is a messy conglomeration of several different web sites, while its "Mandriva Club", originally designed to provide added value to paying customers, has been getting mixed reviews. Although the company has been addressing some of the criticism, it continues to face an uphill battle in persuading new Linux users or users of other distributions to try (and buy) its products.

6.4.3 Minor Distributions

Some prominent and popular minor distributions are:

6.4.3.1 PCLinuxOS

PCLinuxOS is an English only live CD initially based on Mandriva Linux that runs entirely from a bootable CD. Data on the CD is decompressed on the fly, allowing up to 2GB of programs on one CD including a complete X server, KDE desktop, OpenOffice.org and many more applications all ready to use. In addition to the live CD, you can also install PCLinuxOS to your hard drive with an easy-to-use livecd-installer. Additional applications can be added or removed from your hard drive using a friendly apt-get front end via Synaptic.

6.4.3.2 Linux Mint

Linux Mint is an Ubuntu-based distribution whose goal is to provide a more complete out-of-the-box experience by including browser plugins, media codecs, support for DVD playback, Java and other components. It also adds a custom desktop and menus, several unique configuration tools, and a web-based package installation interface. Linux Mint is compatible with Ubuntu software repositories.

6.4.3.2 Sabayon

Sabayon Linux is a live DVD designed to transform a computer into a powerful Gentoo Linux system in less than 5 minutes. Gentoo Linux is a Linux distribution powered by a software install manager engine called "Portage". Besides functioning as a live DVD, Sabayon Linux can also be installed on a hard disk, acting effectively as an easy-to-use Gentoo installation disk. The live DVD includes a large range of desktop environments and open source software applications, such as KDE, GNOME, XFce, Fluxbox, KOffice, OpenOffice.org, FreeNX, amaroK, Kaffeine, etc.

6.4.3.3 Damn Small Linux

Damn Small Linux is a business card size (50MB) live CD Linux distribution. Despite its minuscule size it strives to have a functional and easy to use desktop. Damn Small Linux has a nearly complete desktop, including XMMS (MP3, and MPEG), FTP client, links-hacked web browser, spreadsheet, email, spellcheck (US English), a word-processor, three editors (Nedit, nVi, Zile [emacs clone]), Xpdf, Worker (file manager), Naim (AIM, ICQ, IRC), VNCviwer, SSH/SCP server and client, DHCP client, PPP, PPPoE, a web server, calculator, Fluxbox window manager, system monitoring apps, USB support, and soon it will have PCMCIA support as well. If you like Damn Small Linux you can install it on your hard drive. Because all the applications are small and light it makes a very good choice for older hardware.

6.4.3.4 MEPIS

MEPIS Linux is a Debian-based desktop Linux distribution designed for both personal and business purposes. It includes cutting-edge features such as a live, installation and recovery CD, automatic hardware configuration, NTFS partition resizing, ACPI power management, WiFi support, anti-aliased TrueType fonts, a personal firewall, KDE, and much more.

6.4.3.5 CentOS

CentOS as a group is a community of open source contributors and users. Typical CentOS users are organisations and individuals that do not need strong commercial support in order to achieve successful operation. CentOS is 100% compatible rebuild of the Red Hat Enterprise Linux, in full

compliance with Red Hat's redistribution requirements. CentOS is for people who need an enterprise class operating system stability without the cost of certification and support.

6.4.3.6 Puppy Linux

Puppy Linux is yet another Linux distribution. What's different here is that Puppy is extraordinarily small, yet quite full featured. Puppy boots into a 64MB ramdisk, and that's it, the whole caboodle runs in RAM. Unlike live CD distributions that have to keep pulling stuff off the CD, Puppy in its entirety loads into RAM. This means that all applications start in the blink of an eye and respond to user input instantly. Puppy Linux has the ability to boot off a flash card or any USB memory device, CDROM, Zip disk or LS/120/240 Superdisk, floppy disks, internal hard drive. It can even use a multisession formatted CD-R/DVD-R to save everything back to the CD/DVD with no hard drive required at all!

6.4.3.7 Slax

Slax is a Slackware-based bootable CD containing a Linux operating system, designed with a modular approach. Despite its small size, Slax provides a wide collection of pre-installed software for daily use, including a well-organised graphical user interface and useful recovery tools for system administrators.

6.4.3.8 gOS

gOS is an easy-to-use, Ubuntu-based distribution designed for less technical computer users. Its main features are the use of Enlightenment as the default desktop and tight integration of various Google products and services into the product.

6.4.3.9 Zenwalk

Zenwalk Linux (formerly Minislack) is a Slackware-based GNU/Linux operating system with a goal of being slim and fast by using only one application per task and with focus on graphical desktop and multimedia usage. Zenwalk features the latest Linux technology along with a complete programming environment and libraries to provide an ideal platform for application programmers. Zenwalk's modular approach also provides a simple way to convert Zenwalk Linux into a finely-tuned modern server (e.g. LAMP, messaging, file sharing).

6.4.3.10 DreamLinux

Dreamlinux is a Brazilian distribution based on Debian GNU/Linux. A live CD with a graphical hard disk installation option, it boots directly into an Xfce or GNOME desktops which provide access to a good collection of desktop applications and a central control panel for system configuration.

6.4.4 Package Management

A package management system is a collection of tools to automate the process of installing, upgrading, configuring, and removing software packages from a computer. Linux and other Unix-like systems typically manage thousands of discrete packages.

Packages are distributions of software and metadata such as the software's full name, description of its purpose, version number, vendor, checksum, and a list of dependencies necessary for the software to run properly. Upon installation, metadata is stored in a local package database.

A package management system provides a consistent method of installing software. A package management system is sometimes referred to as a package manager or a system install manager.

6.4.4.1 Functions

Package management systems are charged with the task of organizing all of the packages installed on a system and maintaining their usability. Typical functions of a package management system include:

- Verifying file checksums to ensure correct and complete packages.

- Verifying digital signatures to authenticate the origin of packages.
- Applying file archivers to manage encapsulated files.
- Upgrading software with latest versions, typically from a software repository.
- Grouping of packages by function to help eliminate user confusion.
- Managing dependencies to ensure a package is installed with all packages it requires.

Some additional challenges are met by only a few package management systems.

6.4.4.2 RPM

RPM Package Manager (originally Red Hat Package Manager, abbreviated RPM) is a package management system. The name RPM refers to two things: a software package file format, and software packaged in this format. RPM was intended primarily for Linux distributions; the file format RPM is the baseline package format of the Linux Standard Base.

Originally developed by Red Hat for Red Hat Linux, RPM is now used by many Linux distributions. It has also been ported to some other operating systems, such as Novell NetWare (as of version 6.5 SP3) and IBM's AIX as of version 5.

Every RPM package has a package label, which contains the following pieces of information:

- the software name
- the software version (the version taken from original "upstream" source of the software)
- the package release (the number of times the package has been rebuilt using the same version of the software) this field is also often used for indicating the specific distribution the package is intended for by appending strings like "mdv" (formerly, "mdk") (Mandriva Linux), "fc4" (Fedora Core 4), "rhl9" (Red Hat Linux 9), "suse100" (SUSE Linux 10.0) etc.
- the architecture the package was built for (i386, i686, athlon, ppc, etc.)

RPM file names normally have the format:

```
<name>-<version>-<release>.<arch>.rpm
```

RPM Frontends: yum, YaST, urpmi, rpmdrake, apt-rpm

6.4.4.2 Deb

deb is the extension of the Debian software package format and the most often used name for such binary packages. Like the "Deb" part of the term Debian, it originates from the name of Debra, wife of Debian's founder Ian Murdock.

Debian packages are also used in distributions based on Debian.

Debian packages are standard Unix ar archives that include two gzipped or bzipped tar archives: one that holds the control information and another that contains the data.

Deb Front-ends: Synaptic, apt-get, dpkg

6.4.4.3 Conary

Conary is a free software package management system created by rPath and distributed under the terms of the Common Public License. It focuses on installing packages through automated dependency resolution against distributed online repositories, and providing a concise and easy-to-use Python-based description language to specify how to build a package. Foresight Linux and rPath Linux use it.

Conary updates only those specific files in packages, which need to be updated; this behavior minimizes bandwidth and time requirements for updating software packages. Conary also features Rollbacks of package installation as well as derived packages.

6.4.5 Linux Directory Structure

The directory structure of Linux/other Unix-like systems is very intimidating for the new user, especially if he/she is migrating from Windows. In Windows, almost all programs install their files (all files) in the directory named: `Program Files.' Such is not the case in Linux. The directory system categorizes all installed files. All configuration files are in /etc, all binary files are in /bin or /usr/bin or /usr/local/bin. Here is the entire directory structure along with what they contain:

/ - Root directory that forms the base of the file system. All files and directories are logically contained inside the root directory regardless of their physical locations.

/bin - Contains the executable programs that are part of the Linux operating system. Many Linux commands, such as cat, cp, ls, more, and tar, are locate in /bin

/boot - Contains the Linux kernel and other files needed by LILO and GRUB boot managers.

/dev - Contains all device files. Linux treats each device as a special file. All such files are located in /dev.

/etc - Contains most system configuration files and the initialisation scripts in /etc/rc.d subdirectory.

/home - Home directory is the parent to the home directories of users.

/lib - Contains library files, including loadable driver modules needed to boot the system.

/lost+found - Directory for lost files. Every disk partition has a lost+found directory.

/media - Directory for mounting files systems on removable media like CD-ROM drives, floppy disks, and Zip drives.

/mnt - A directory for temporarily mounted filesystems.

/opt - Optional software packages copy/install files here.

/proc - A special directory in a virtual filesystem. It contains the information about various aspects of a Linux system.

/root - Home directory of the root user.

/sbin - Contains administrative binary files. Commands such as mount, shutdown, umount, reside here.

/srv - Contains data for services (HTTP, FTP, etc.) offered by the system.

/sys - A special directory that contains information about the devices, as seen by the Linux kernel.

/tmp - Temporary directory which can be used as a scratch directory (storage for temporary files). The contents of this directory are cleared each time the system boots.

/usr - Contains subdirectories for many programs such as the X Window System.

/usr/bin - Contains executable files for many Linux commands. It is not part of the core Linux operating system.

/usr/include - Contains header files for C and C++ programming languages

/usr/lib - Contains libraries for C and C++ programming languages.

/usr/local - Contains local files. It has a similar directories as /usr contains.

/usr/sbin - Contains administrative commands.

/usr/share - Contains files that are shared, like, default configuration files, images, documentation, etc.

/usr/src - Contains the source code for the Linux kernel.

/var - Contains various system files such as log, mail directories, print spool, etc. which tend to change in numbers and size over time.

/var/cache - Storage area for cached data for applications.

/var/lib - Contains information relating to the current state of applications. Programs modify this when they run.

/var/lock - Contains lock files which are checked by applications so that a resource can be used by one application only.

/var/log - Contains log files for different applications.

/var/mail - Contains users' emails.

/var/opt - Contains variable data for packages stored in /opt directory.

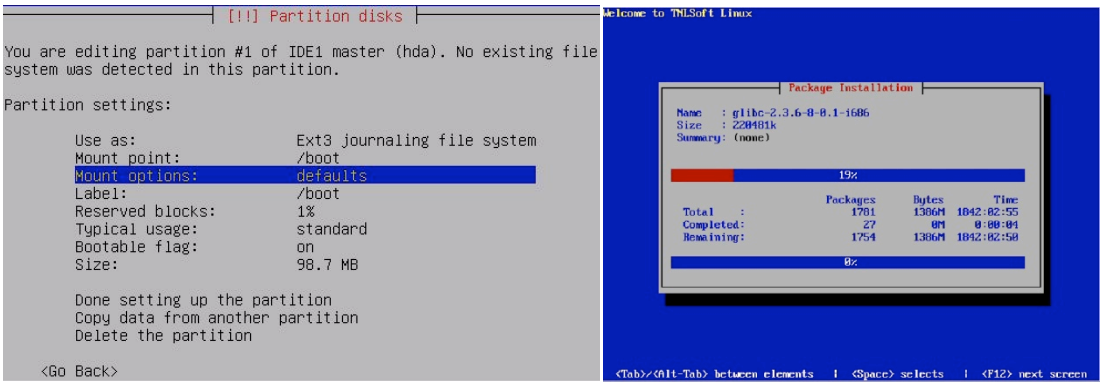
/var/run - Contains data describing the system since it was booted.

/var/spool - Contains data that is waiting for some kind of processing.

/var/tmp - Contains temporary files preserved between system reboots.

6.4.6 Linux Installer

There are different types of installers depending on its distribution. Where everything is done by texts only.



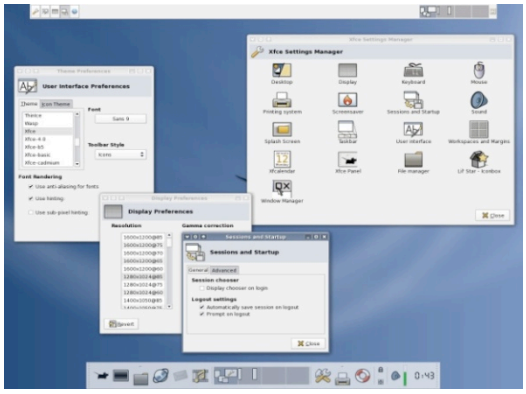
For advanced users – text based installers seem to be comfortable. On the contrary, beginners or non-technical users get scared by this text based installer.

From the beginning Linux used to come with this text-based installer. When Linux approached the general desktop users, most of the distributors started to provide GUI-based installers as well. Now, some of the distributions only have GUI-based installer, Ubuntu is an example.

GUI-based installers are more user-friendly. It gives intuitive environment for all level of users with step by step guidance to install the operating system on a computer from scratch.

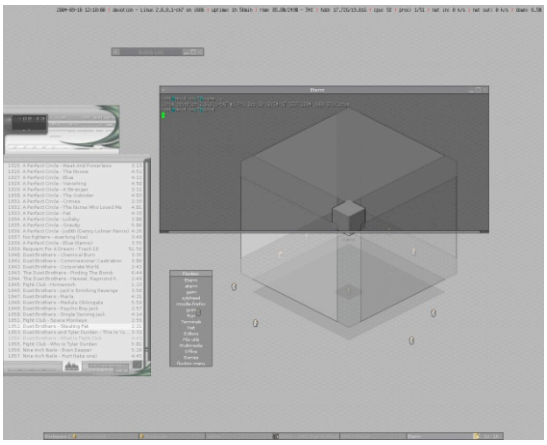
6.4.7 Desktop Environment

In graphical computing, a desktop environment (DE, sometimes desktop manager) presents a



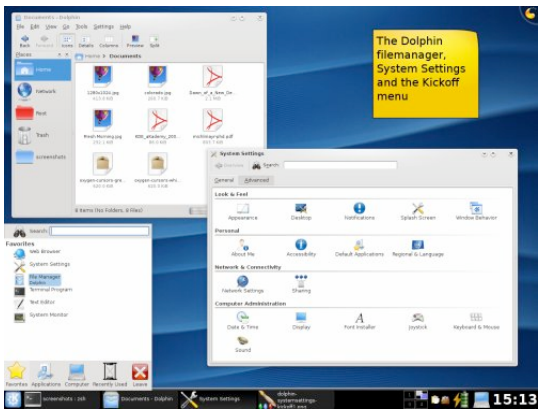
graphical user interface (GUI) to the user. The name is derived from the desktop metaphor used by most of these interfaces, as opposed to the earlier, textual command line interfaces (CLI). A desktop environment typically provides icons, windows, toolbars, folders, wallpapers, and desktop widgets. In addition, a desktop environment may offer collaboration support like drag and drop and inter-process notification. On the whole, the purpose of a desktop environment is the consistent integration of a graphical user interface and its applications to the user with a consistent look and feel.

Most of the general Linux user claims that they use Linux because of the desktop customization opportunities where other group of users blames Linux as immature for its desktop environments. It is true that till today there is no single DE for all Linux or a unified system of different DEs to make life easier.



Linux uses X windows system. All the DE Linux can use actually are made for X windows systems.An X Window System desktop environment combines a window manager with a suite of standard applications that adhere to human interface guidelines and runs under the X Window System. They are often used with operating systems such as Linux. Whereas a window manager is analogous to the Aqua user interface for OS X, a Linux desktop environment is analogous to Aqua as well as all of the default OS X graphical applications and configuration utilities. Some window managers such as IceWM, Fluxbox and Windowmaker contain rudimentary desktop environments, while others like evilwm and wmii do not. Initially, CDE was available as a proprietary solution, but was never

popular on Linux systems due to cost and licensing restrictions. In 1996 the KDE was announced, followed in 1997 by the announcement of GNOME. Xfce is a smaller project that was also founded in 1997, and focuses on speed and modularity



GNOME's graphical file manager Nautilus is very easy to use and is packed with a lot of features making it easy for new Linux users to easily pick up and understand its working. KDE's Konqueror is also very easy to use, both as a file manager and as a web browser. However this ease of use comes at a price, as both Nautilus and Konqueror are noticeably slower than lighter weight file managers. Some users object to their multifarous nature as both local file browser and remote client, fearing security issues and preferring a more minimalist approach. Both GNOME and KDE come with many pre-packaged graphical configuration tools, reducing the need to manually edit configuration files for new users. They have extensive bundled software such as graphical menu editors, text editors, audio players, and software for doing administrative work. All applications installed in most distributions are automatically added to the GNOME and KDE menus. No major configuration changes are necessary to begin working. However, by using graphical tools, the extent to which the desktops can be configured is determined by the power provided by those tools.

6.4.8 Drivers

Most of the device drivers comes with the Linux kernel. Meaning, same version of Linux kernel would support same devices regardless of the distribution. But most of the distribution adds more device drivers to support more devices. No free Linux distribution provides proprietary Linux drivers. Recently a tendency of adding proprietary device drivers is noticeable. Still most of the devices are not “properly ” supported by Linux. There are a huge number of devices which do not work at all with Linux.

Device support is the most important issue now. More general users are interested to move into Linux, but when they try to install it they realize that Linux does not support most of their hardware. There may be some workarounds, but if the user is not an advanced user – he will never use the workaround.

The most common two hardware – sound and video seemed to be the most problematic. But recently both seem to be solved. The audio drivers are provided by ALSA (Advanced Linux Sound Architecture) and OSS (Open Sound System). They seem to cover all the sound devices to support the drivers. For graphics drivers VESA (Video Electronics Standards Association) can give the minimal graphics output to work. Recently nVidia and ATI started to provide proprietary drivers, which gives full acceleration power for the respective graphics devices. Other device manufacturers are following the lead. Because of the popularity of Linux and the demand by large number of users more vendors are providing drivers for Linux. Intel and Broadcom are two good examples.

The OS distributors do not give any support for the proprietary device driver.

6.4.9 Audio/Video Codecs

Being open source and free operating system, most of the Linux do not come with non-free audio/video codecs. A common complain and one of the biggest reason of not using Linux is, users can not play mp3 audio files out of the box. Most of the users do not know, or do not care to know that mp3 licensing takes a good amount of money. The following is the figure taken from mp3 authority Thompson Consumer Electronics.

PC Software Applications

PC Software applications which incorporate mp3 / mp3PRO decoding (player, decoder) and software applications incorporating mp3 / mp3PRO encoding capabilities (encoder, ripper, recorder, jukebox).

mp3PRO patent and software license	
This **patent and software license** license covers patents and mp3PRO software (Windows, MacOS object code libraries) developed by Coding Technologies.	
Decoder	· **US$ 1.25** per unit or **US$ 90 000.00** one-time paid-up
Codec	· **US$ 5.00** per unit

mp3 patent and software license	
This **patent and software license** license covers patents and mp3 software (Windows, MacOS object code libraries) developed by Fraunhofer IIS-A.	
Decoder	· **US$ 0.75** per unit or **US$ 60 000.00** one-time paid-up
Codec	· **US$ 5.00** per unit

mp3 patent-only license	
This **patent-only** license is needed in case the mp3 software is developed in-house or licensed from a third party.	
Decoder	· **US$ 0.75** per unit or **US$ 50 000.00** one-time paid-up
Codec	· **US$ 2.50** per unit

Minimum Royalties
Annual minimum royalties are payable upon signature and each following year in January and are fully creditable against annual royalties.
· **US$ 15 000.00** per calendar year

Note: *This license does not cover the right to distribute, broadcast and/or stream mp3 / mp3PRO encoded data. These rights are covered by the licenses described under* ***Electronic Music Distribution / Broadcasting / Streaming***.

Just like the mp3 format, all other familiar and popular audio/video formats need licensing. Microsoft, IBM, Apple and SUN are licensed members of mp3. Whichever operating system they release, it is supposed to support mp3 out of the box.

There is a open source mp3 and other popular codec implementation known as Gstreamer-Ugly. Which provides support for the playback of previously non-supported formats, but in lossy mode. Recently few Linux distributors are providing mp3 support out of the box by this gstreamer codecs.

There was another practice of providing non-free codecs. For example SuSE Linux did not have any native support for non-free audio/video codecs but they used to give Real Player. SuSE had license for Real Player and Real had license of playing a number of non-free formats.

Most of the desktop computer user uses computer for music and video playback. If there is no out the box and/or native support for the popular formats, users would not feel like using Linux. There are free and open source high quality formats, but as long as the users are not aware of

these – the free formats have no use. There is another problem – the portable players. There is no portable player which can play flac or ogg format, but all the player are able to play mp3 and m4a, few of them can play expensive wma format. Users would like that format which is able to play in portable player and portable PC without any conversion.

6.4.10 Requirements

Linux gives the freedom of architecture free platform. Linux is available for x86/x86-64 PC, Apple's PPC/Intel Mac and SUN's SPARC.

Linux is able to run in lowest configured computer to latest computers with 64bit processor. A latest release of Ubuntu requirements are

6.4.10.1 Bare Minimum requirements

- 300 MHz x86 processor
- 64 MB of system memory (RAM)
- At least 4 GB of disk space (for full installation and swap space)
- VGA graphics card capable of 640x480 resolution
- CD-ROM drive or network card

6.4.10.2 Recommended minimum requirements

Ubuntu should run reasonably well on a computer with the following minimum hardware specification. However, features such as visual effects may not run smoothly.

- 700 MHz x86 processor
- 384 MB of system memory (RAM)
- 8 GB of disk space
- Graphics card capable of 1024x768 resolution
- Sound card
- A network or Internet connection

6.4.10.3 Absolute minimum requirements

- Intel 486 processor
- 32 MB of system memory (RAM)
- 300 MB of disk space

Source: https://help.ubuntu.com/community/Installation/SystemRequirements

6.5 Evaluating Linux User Experience

6.5.1 Installation

First thing I expected that installation would bother the users. But recently the installation method became so much user-friendly, so that no one actuallycomplains about the procedure. Those who have installed operating systems orany software are able to install Linux (now).Again,

I have shown text-based installation method to some users, they seemedto be lost.The major complain comes related to swap space. Users do not know and do notcare about swap space. When it comes to Linux installation. A separate swapspace is necessary - users don't seem to understand this issue.

6.5.2 Desktop Environment

Users have hard time to choose which DE to use. There are around 4/5 major DE's. All of them look nice and seem to work fine. Getting familiar and becoming stable takes some time for all the users.

Desktop Environments are highly customizable. Most of the Linux users seem to enjoy the customizations. There are huge resources available for free to customize the desktop.

6.5.3 Applications

For all jobs there is some application available. Still users seem to be disappointed when there is no Adobe Photoshop, Adobe Flash, Autodesk AutoCAD, Autodesk 3d Studio Max, SONAR, Pinnacle or any professional audio, video, 3d, CAD or animation applications. There are some open source implementations, but they are not even near to the professional versions.

Those who are sound engineers, video editors, architects and animators will never use Linux until their software are becoming available.

6.5.4 Gaming

For gamers, Linux is the ultimate disappointment. There are a good number of hardcore Linux users, who users Microsoft Windows only to play games. Linux is a stable platform. Either game developers have to start releasing Linux versions or Linux developer have to implement direct X implementations natively.

6.5.5 Audio/Video

Audio, video quality seems to be low graded on Linux. The reason is, most of the Linux codecs are open course implementation. They are intentionally low graded to avoid lawsuit by the original codec owner.

6.5.6 Drivers

Though most of the hardware works out natively but some of them do not work perfectly. This is because of the open source development. Unless the device vendor is not providing proprietary driver, we have to use the open source implementations.

6.5.7 Boot Time

Most of the desktop users boot the computer only whenever they need. Home isthe place where users do not like to turn on the computer all day long. The boottime (as well as shutdown time) is

important for home desktop users. Comparingto other OSs, Linux takes longer time to boot. It seems to bother the users.Besides, Linux shows some text (verbose mode) that does not make any sense.

6.5.8 Suspend issues

I never knew Linux had a big issue with it comes to suspend and hibernate.Specially, notebook computer users need these features. Closing the lid wouldsuspend the computer and pressing power button would hibernate. This is thegeneral idea about suspend and hibernate, but Linux has issues here.Suspend (or sleep/standby) is known as suspend to RAM (s2ram). On suspendmode peripherals does not consume electricity except the RAM, the computerenters low power state mode consuming very low amount of electricity.When suspending, some observations are:

- On some distros, closing the lid does nothing
- Pressing power button does 1) nothing, 2) shows shut down menu or 3)locks the computer
- The display goes "black" but nothing happens
- The computer enters lock mode where users have to use their passwordto unlock. After unlocking, system shows a message that the computerhad problems when sleeping
- Computer enters terminal mode and stops working
- Suspends, but never wakes up
- On resume – few devices stop working (Bluetooth, wireless device,network device, touchpad)
- Reboots.

Waking up from suspend mode is supposed to take little time. But on Linux thereare complains that it takes up to 1 minute.

6.5.9 Hibernate issues

Hibernate is the mode where all the contents of RAM is being saved in swapspace. It is known as suspend to disk (s2disk). After saving everything into swapspace the computer can be turned of. Next time when the computer wakes up, allthe contents of swap space is restored and the user can see and use from hisexact last state.Important point to be noted is, hibernation uses swap space. Users who do nothave swap space, or enough (equal or more than the RAM) swap space would notbe able to hibernate.Some hibernation issues (by pressing power button, or selecting the hibernateoption)

- Doesn't do anything
- Stops working
- Shows memory error messages
- Show permission error
- Turns off the computer immediately

6.5.10 Installation from USB issues

Recently, the trend of installing the whole operating system and running fromthe USB became popular. Another practice of installing Linux from the USB driveinstead of CD/DVD media is being practiced.But there is an issue with USB media installation as well. After the

install,USB/CD/DVD medias stop mounting. It shows an error that wrong mountoptions have been used.

6.5.11 Screen resolution issue

A common problem. Almost all users had to face this problem. After install thedesktop environment shows lowest or wrong resolution. Most of the time resolution cannot be fixed unless the user chooses VESA as his graphics driver.

6.5.12 Graphics driver problem

Until last year, vendors never provided proprietary drivers for the graphicscards. Using non-proprietary drivers led to many problems (instability, flickering, wrong resolution).

From last year, nVidia and ATI (AMD) started providing proprietary drivers,which solved many of the issues.

6.5.13 Disk mounting issues

It has been seen that, if a portable drive is not properly removed from WindowsOS, Linux has issues to mount them. It simply denies mounting them.This problem is reproducible, if the drive is NTFS and if the OS is using NTFS-3G.

WiFi and Bluetooth card issues. Most of the Linux distros has issues with WiFi and Bluetooth. Either the OScannot recognize the wireless device or it cannot connect the APN or the other Bluetooth devices.

For Bluetooth devices, even if it recognizes other Bluetooth devices, it simplydenies to transfer data.

6.5.14 SAMBA issues

SAMBA is an implementation of SMB (Server Message Block) networkingprotocol. It is an essential application to be connected with a Windows basednetwork and share data. On most of the Linux distros, SAMBA just does not work.

It shows the networked computers but would not connect to any server forunknown reason.

In the end, most of the issues mentioned here – has workarounds. But the general users ornew users do not like or prefer work around. They always look for the thingsthat “just work”. It is really important for Linux distributions to take care of thecommon issues to be a real desktop operating system.

Chapter 7: UX Development

7.1 Twelve Questions and Answers (New)

This would be a little effort to answer the twelve questions in current context which was actually asked in an interview with Bastian and Aaron J. Seigo from the KDE project and Havoc Pennington from the Gnome project by Eugenia Loli-Queruin10th Mar 2003 (published in OSNews as web article).

Here the original questions are modified to suit the research. These questions and answers are for research purpose only; this research has no affiliation with any company or group.

Question 1: *Few or infinite configuration options for application/UI?*

Answer: Now, Linux has all types of users. Some of them do not care about the options, some of the want more and more options and some users make ways of doing things. To keep all the users satisfied, a minimal standard of configuration option set is preferable. The essential things should be present, few important new things may come by default and there should be easier ways to remove them. If we consider GNOME or KDE, most of the options are modifiable from the option menu. That is absolutely fine, but now consider Aqua interface's example. New things can be added by just drag and drop, things can be removed by the same process (poof!). Still it is true that KDE, GNOME, Xfce are more configurable than Aqua or Aero. But there should be easier ways of doing regular things.

Again, for the power users, they won't mind if they need to edit a text file to edit the configuration. There can be options for them as well.

Right now, Xfce has the most minimalistic but most usable UI.

Default interface should be intuitive, usable, attractive and simple. Intuitive is the most important factor. Users won't like to use manual or ask someone for help to learn how to get back on the previous page. Usable – a Print button on a file manager sounds good, users may use it to print the window content once in a while but it doesn't have the usability. Most of the users would not find it useful. If most of the users do not use an option, the option shouldn't be there by default. Attractive part is a bit tricky. If the default interface is way attractive, users may avoid modifying it in fear of losing the attractiveness. More option means more complexity; being simple a UI can get rid of thousands of complexity.

When designing or modifying a DE/UI. It is important to learn from the users. Pre release alpha/beta versions are good ideas, pubic community forums, mailing lists are good ways of discussing the new ideas. Some Linux distribution authorities are practically doing this, and they are gaining popularity as users are getting what they are asking for. Ubuntu and Linux Mint are two prominent examples.

Question 2: *For the sake of consistency, does Linux DEs have any certification/guidelines?*

Answer: Yes, both GNOME and KDE have UI/HIG guidelines. To keep the interface consistence, following these guidelines are important. Still, there is no formal certification for Linux DEs. Being open source and community driven software, giving any sort of certification or stamp of approval seems unnecessary. But in future if there be one unified DE, the idea may become useful.

Five years ago, the guidelines were just formalities. But now it became a norm that everyone should have a guideline and developers should use the guideline. Recently KDE started to renovate their existing guideline after releasing KDE 4.0. They are calling it HIG 2.0.

Question 3: *Is FreeDesktop.org successful?*

FreeDesktop.org's target was to provide a single platform to use KDE, GNOME, Xfce, Fluxbox natively. Five years later, we can see that – no they are not successful. Only Foresight Linux uses FreeDesktop platform and it does not seem to be much appreciated.

The problem is, all users are using more or less the same idea as the FreeDesktop.org. Consider a users installed Ubuntu with only GNOME DE. As long as he is using only GNOME applications, he won't need any other library. If he needs to install a KDE application, for example AmaroK (KDE music player) he needs to install few KDE libraries and binary files for AmaroK. He does not need all the libraries of KDE why would he use FreeDesktop's platform?

Question 4: Is *hiding the directory structure from users necessary?*

Answer: If the target users are general users, yes it is necessary. Linux has been considered as programmers OS for a long time, one of the reason is this directory structure. For the general users, the structure is complicated. Mac OS X, Windows are highly considered as general users OS, they do not show the structure. Mac OS X is using UNIX underneath has the same structure of Linux. If they can hide it from the users, why Linux is not doing the same.

It is true that most of the computer user are using Windows OS. They get confused with the totally different structure. All they consider is, they could do this before, now they cannot do it anymore. For example, Windows lets the user write anything on the primary partition. Mac does more or less the same thing. But on Linux users cannot write anything anywhere except his home directory.

Linux really need to hide the directory structure. For this, the Linux kernel doesn't have to be changed. The DE can hide it from the user. Letting users to write in all partitions can be another important option.

Recently some Linux distributions started to eliminate the "root" user idea, which was required for the general users. Most of the desktop users use the computer alone. They are the root user and the only user. Keeping two accounts, one for root another for general use doesn't fit the user's expectation. Windows and Mac OS has the same idea, but they hide it from the user.

Again, asking password for everything the user is doing is an annoyance. There should be options to disable it.

Question 5: *Disliking of Windows XP interface?*

Answer: I personally found Windows XP interface nice. It does the job. The thing I do not like is, it doesn't give options for customizations.

Question 6: *Better file management approaches (by file system or by applications)?*

Answer: The choice depends on the users. By default the file manager should be able to organize the files properly. There should be easier and faster ways of previews of the contents. There can be applications which can use it's database to organize the files in different ways. For example, AmaroK, the KDE music player can organize all media files with SQL Lite database with search and play facility. The F-spot a GNOME photo manager application became enough powerful to organize all the photos with attractive options (for example: timeline). Users can use both file manager or the applications to organize and use their files.

Question 7: *Is there any way to integrate X11 DE with underlying system to learn the overall system?*

Answer: Till today, the answer is no. The one possible solution is to use only one configuration tool all the time. Another way could be a unified application which can notify both types of DEs about the system.

Question 8: *Large number of bundle application, acceptable or not?*

For better integration, all sort of application comes in bundle. But practically most of the applications are useless. The current GNOME approach can be considered as standard. Now GNOME is providing one software for one purpose method. It ensures that the system is not going to be blotted. Recently KDE is following the GNOME's lead, which is appreciable.

Question 9: *Is there any chance of common package management?*

A common package management is one of the most anticipated thing for the Linux users. But till today, there is none. Some package manager can use multiple formats, for example KDE's Adept (deb, rpm, bsd). Recently APT (Advanced Packaging Tool) became popular. It is originally made for Debian's .deb format, but a modified version apt-rpm can work with RPMs. APT is available for MacOS X (by Fink) and Solaris as well. This APT thing can be treated as common package manager, which solves a lot of problems.

Question 10: *File managers with lot of functionalities or let individual apps do the job?*

Answer: A noticeable innovation is, Mac OS X Leopard's Quick Look. Which shows preview of audio, video, document, presentation and file/folder information in a pop-up window by pressing space-bar or a single click on Quick Look icon. It is really appreciated by all Mac OS X users. The same implementation can be done on Linux as well.

Konqueror uses to integrate web browser, document editor and image viewer with the file manager. Which is alright, but users of other operating systems do not seem to like the idea (as they are more familiar with using different application for different purposes).

Question 11: *One DE. Does unification sounds too extreme?*

FreeDesktop.org is a failure. KDE, GNOME and Xfce will never be unified. Users won't use only one DE based applications only. Therefore, unification sounds impossible. Multiple DE libraries using at the same time increases complexity. Still today, dependency problem won't let old applications. Right now, users have to use any of the DEs ar primary DE and if he needs applications of different DE – he have to install libraries of that DE as well.

Question 12: *What major change we may expect in next 5 year?*

Rise of Xfce. A complete different DE may arise and may become the user's choice. More graphics intensive and interactive UI may come.

7.2 Key factors to gain higher Linux Experience

7.2.1 Distribution

A complete different Linux implementation with completely different name.

The trend of Linux implementation must change. From the beginning of the Linux history, all Linux distributions are more or less the same. This trend needs to change.

gOS might be a good example. gOS is a Ubuntu derived Linux operating system, though it doesn't mention that it is Linux. It uses Enlightment desktop environment which is completely different from usual Linux DE's. It's target users are web 2.0 users. gOS doesn't show the directory structure or terminal unless users want to dig inside it. Another interesting implementation is, using guided commands without using the terminal at all. As their target users are non-technical naive users, these implementations are appreciable.

MacOS X is another example. MacOS X is based on UNIX but it hides its architecture from the users and gives a friendly interface to the users. There's a rumor that next MacOS X (10.6) is going to use Linux kernel instead of UNIX.

7.2.2 Boot/Shutdown Time

Unlike other operating systems, Linux based operating systems take more time to boot up and shutdown. The main reason is monolithic kernel. If the kernel is stripped down according to the target machine, the boot/shutdown time will dramatically decrease.

7.2.3 Proprietary Drivers

More vendors have to provide Linux drivers with their product or at least they must support open source driver developers so that they can implement drivers for their products.

7.2.4 Audio/Video Codecs

Desktop users use their computer as multimedia station. They will need audio video codecs either ways. Providing out of the box codecs help gaining better user experience.

7.2.5 Single Desktop Environment

Only one desktop environment with associated library. For example: Mac OS X user UNIX underneath with a stable desktop environment. Linux can follow the lead.

7.2.6 Easier Package Management

Instead of depending software repositories and command line package managers, there should be more easier way of installing newer or older software. General user must not need to be aware of any dependency issue.

7.2.7 No Terminal

Never ask a general user to use terminal or any command line tool.

7.2.8 Cost

Linux implementations do not have to be essentially free. If the service is reliable users would rather like to pay for the operating system. As some codecs costs money, developers can assign costs for the distribution.

7.3 New Ideas

According to the evaluated user experience on this part I'd suggest some new ideas of a conceptual Linux implementation.

1. Kernel: Stripped down Linux kernel. (Latest Stable)

2. Desktop Environment: A new DE made only for this OS or GNOME.

3. Application framework: Basically GTK2+, will support other libraries as well.

4. Package Management: APT for Deb and RPM. Only one add/remove utility for overall system.

5. OS Installer: GUI Based

6. File structure: Native Linux structure, but will be hidden from the user(unveiled by terminal only.

7. Applications:

Category	Suggested application
Office Suite	OpenOffice.org
Audio Player	AmaroK
Video Player	VLC
Photo Manager	F-Spot
Email Client	Mozilla Thunderbird
Bittorrent client	Transmission
Graphics editor	GIMP
Web browser	Mozilla Firefox
Web 2.0 Applications	Flickr uploader, Blogger uploader etc.

8. Java6, GCC, common graphics drivers including nVidia and ATI, Wireless adapter's driver should be provided to work out of the box.

9. One control panel based system-wide system settings tool.

10. Application installer: Bundle structured applications.

This application installer needs to be described a bit. Linux application installers have a problem from the beginning – dependency problem. A newer version of the application needs newer dependency where an older version of the same application needs older dependency. Even if the user needs both versions of the same application, he cannot do it because of the dependency problem.

If the dependency was shared the problem becomes more critical. For example old A and old B needs dependency old X. New B needs new X. The user installed new B so the old X is replaced by new X. Now the old A will stop working. If the user wants to replace the new X with old X, he'll get notification that he's trying to replace with older version and it will break this and that dependencies so he is not allowed to do so. Then if the user manually replaces the new X with old X, then new B will stop working.

Bundle Application structure seems to be the solution of these problems. This would solve the problems of the desktop environment libraries as well. It means, a GNOME desktop environment will have only GNOME libraries installed and nothing else yet would be able to run KDE and Xfce applications natively.

Before getting into the bundle structure, let me define local libraries and global libraries. Local libraries are the libraries which has explicit dependencies for the application. The libraries which are globally common and there is no dependency regarding versions are the global libraries.

A bundle application would have everything in a single package except the global libraries. Only global libraries will be the shared resource for this type of applications. When the application will be executed, it will use global resources and local resources together to run the application. When the program will be terminated, the application will release the local resources completely so that would be no memory footprint for the non-native libraries. This solution would also solve the problem of memory wastage because of unnecessary static libraries.

These applications do not need any installation process. Direct execution of the program is the only way to use the program. There is no need of uninstallation as well, deleting the application would do the job.

7.4 Recommendations

Recommendation for the existing Linux developers and the new developers is to involve the users from the first step of the development. Let users decide what they want. Survey the users, show them demos, ask them for the desktop theme ideas. Try to provide according to what they want as much as possible.

Already some of the distro developers are using this method. Linux Mint already have set an example. Ubuntu is a community driven Linux distribution, which comes with no non-free codec and basic GNOME. Linux Mint is based on Ubuntu but adds different interface, few different Mint tools and audio, video player with required codecs. Mint uses the Ubuntu repository as well. Mint's best quality is, they develop according to what users ask for them on their forum. If a user suggests a logical and useful option, on the next version Mint literally implements that option. That boosted the demand of the Linux Mint and within short time Linux Mint reached #4 of distrowatch.org's Linux ranking. Mint's GNOME version is just one example; there are Mint's KDE, Xfce and Fluxbox versions as well. Linux Mint is successful because they listen to what users want. This example should be followed.

Conclusion

This research only shows what is user experience, how it relates with software engineering. Operating System is an important software. As users have to use it all the time, an operating system has to be most usable. On the second part of this research we have seen what is an operating system and few functions of it. Then the research focused on Linux and how UX relates with it. Then we evaluated some issues of Linux and finally recommended some ideas.

Evaluation of user experience is necessary for operating system development – for both old and new operating systems. Linux for desktop computer users is relatively new idea. If developers want to make it highly usable, they have to involve the users in the development process. In every cycle of the software development process developers must use the valuable feedbacks of the users. Thus maximum usability can be provided.

References

Book References:

[1] *Roger S. Pressman (2005).* **Software Engineering: A Practitioner's Approach**, 6th Edition, Avenue of the Americas, NY: McGraw-Hill

[2]*Roger S. Pressman (2001).* **Software Engineering: A Practitioner's Approach**,5th Edition, Avenue of the Americas, NY: McGraw-Hill

[3] **The Elements of User Interface Design** by *T. Mandel* (1997), Upper Saddle River, NJ: John Wiley & Sons

[4] **Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design** by *Larry L. Constantine, Lucy A.D. Lockwood*

[5]*Deitel, Harvey M.; Deitel, Paul; Choffnes, David* (2004).**Operating Systems**. Upper Saddle River, NJ: Pearson/Prentice Hall.

[6]*Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg* (2004).**Operating System Concepts**. Hoboken, NJ: John Wiley & Sons.

[7]*Tanenbaum, Andrew S.; Woodhull, Albert S.* (2006).**Operating Systems. Design and Implementation**. Upper Saddle River, N.J.: Prentice Hall.

[8]*Tanenbaum, Andrew S.* (2001). **Modern Operating Systems**. Upper Saddle River, N.J.: Prentice Hall.Bic, *Lubomur F.; Shaw, Alan C.* (2003).**Operating Systems**.. Pearson: Prentice Hall.

[9]*Stallings* (2005).**Operating Systems, Internals and Design Principles**. Pearson: Prentice Hall.

Web References:

[1] An Evolving Glossary of Experience Design: http://www.nathan.com/ed/glossary/index.html(Last accessed: October 15, 2007)

[2] Glossary user experience: http://www.simply.com.au/glossary.php?alphabet=U(Last accessed: October 15, 2007)

[3]Glossary user experience: http://www.intersect.co.nz/info-glossary.html(Last accessed: October 15, 2007)

[4] Talking bout the Elements of User Experiencehttp://www.webword.com/interviews/jjg.html(Last accessed: October 15, 2007)

[5] About user experience: http://www.nngroup.com/about/userexperience.html(Last accessed: October 15, 2007)

[6] How to quantify user experience: http://www.sitepoint.com/article/quantify-user-experience(Last accessed: October 16, 2007)

[7] Definition of UX: http://www.nnyman.com/personal/2006/03/03/definition-of-user-experience-second-version/(Last accessed: October 16, 2007)

[8] Donald Norman (Wikipedia): http://en.wikipedia.org/wiki/Donald_Norman(Last accessed: October 16, 2007)

[9] User Experience (InfoDesign): http://www.informationdesign.org/archives/cat_user_experience.php(Last accessed: October 16, 2007)

[10] UX pioneers: http://www.adlininc.com/uxpioneers/(Last accessed: October 16, 2007)

[11] Designing of UX: http://www.dux2007.org/(Last accessed: October 17, 2007)

[12] UX Magazine article: http://www.uxmag.com/h(Last accessed: October 17, 2007)

[13] Interaction Design Association: http://www.ixda.org/en/(Last accessed: October 17, 2007)

[14] UI design: http://www.usernomics.com/user-interface-design.html(Last accessed: October 17, 2007)

[15] Microsoft Usability: http://www.microsoft.com/usability/default.mspx(Last accessed: October 17, 2007)

[16] Article 1:**Why Features Don't Matter Anymore, The New Laws of Digital Technology**(Author: Andreas Pfeiffer) http://www.acm.org/ubiquity/views/v7i07_pfeiffer.html, *(posted January 19, 2006, last accessed: October 18, 2007)*

[17] Article 2:**User Experience Research** (Author: Andreas Pfeiffer) Source: http://www.acm.org/ubiquity/views/v7i43_pfeiffer.html, *(posted March 16, 2006, last accessed: October 18, 2007)*

[18] Agile software development: http://en.wikipedia.org/wiki/Agile_software_development (Last accessed: November 19, 2007)

[19] New methodology of software development: http://martinfowler.com/articles/newMethodology.html (Last accessed: November 19, 2007)

[20] Bridging the distance: http://www.ddj.com/architect/184414899 (Last accessed: November 19, 2007)

[21] The agile Manifesto: http://www.ambysoft.com/essays/agileManifesto.html (Last accessed: November 19, 2007)

[22] Agility (Wikipedia): http://en.wikipedia.org/wiki/Agility (Last accessed: November 19, 2007)

[23] Agile modeling: http://www.agilemodeling.com/ (Last accessed: November 19, 2007)

[24] The Agile Alliance: http://www.agilealliance.org/ (Last accessed: November 19, 2007)

[25] Agile Usability: http://www.agilemodeling.com/essays/agileUsability.htm (Last accessed: November 19, 2007)

[26] Usability first: http://www.usabilityfirst.com/ (Last accessed: November 21, 2007)

[27] Usability (Wikipedia): http://en.wikipedia.org/wiki/Usability (Last accessed: November 21, 2007)

[28] UI design: http://www.uidesign.net/1999/papers/UIA1.html (Last Accessed: November 21, 2007)

[29] KDE (Wikipedia): http://en.wikipedia.org/wiki/KDE

[30] UI design: http://www.ambysoft.com/essays/userInterfaceDesign.html (Last Accessed: November 21, 2007)

[31] OS (Wikipedia): http://en.wikipedia.org/wiki/Operating_system (Last accessed: March 18, 2008)

[32] Windows Aero (Wikipedia): http://en.wikipedia.org/wiki/Windows_Aero (Last accessed March 20, 2008)

[33] Aqua (Wikipedia):http://en.wikipedia.org/wiki/Aqua_%28user_interface%29 (Last accessed March 20, 2008)

[34] What is KDE: http://www.kde.org/whatiskde/ (Last accessed March 20, 2008)

[36] Windows Project: http://techbase.kde.org/index.php?title=Projects/KDE_on_Windows (Last accessed March 20, 2008)

[37] Mac OS X KDE Project: http://techbase.kde.org/index.php?title=Projects/KDE_on_Mac_OS_X (Last accessed March 20, 2008)

[38] Windows KDE Project: http://techbase.kde.org/index.php?title=Talk:Projects/KDE_on_Windows/Installation (Last accessed March 20, 2008)

[39] GNOME (Wikipedia): http://en.wikipedia.org/wiki/GNOME (Last accessed: March 20, 2008)

[40] GNOME: The Free Software Desktop Project: http://www.gnome.org/ (Last accessed March 20, 2008)

[41] GNOME 2.22 Release Notes: http://library.gnome.org/misc/release-notes/2.22/#rnusers (Last accessed March 20, 2008)

[42] GNOME Human Interface Guidelines 2.0: http://library.gnome.org/devel/hig-book/stable/ (Last accessed March 20, 2008)

[43] Comparison of X Window System desktop environments: http://en.wikipedia.org/wiki/Comparison_of_X_Window_System_desktop_environments (Last accessed March 20, 2008)

[44] Free software UI: http://ometer.com/free-software-ui.html (Last accessed March 20, 2008)

[45] About Xfce: http://www.xfce.org/about/ (Last accessed March 20, 2008)

[46] Memory usage: http://thunar.xfce.org/wiki/memory_usage (Last accessed March 20, 2008)

[47] Wikipedia Xfce: http://en.wikipedia.org/wiki/Xfce (Last accessed March 20, 2008)

[48] Xfce 4.4.0 and so far: http://foo-projects.org/pipermail/xfce4-dev/2007-January/022198.html (Last accessed March 20, 2008)

[49] Definition of hci: http://sigchi.org/cdg/cdg2.html (Last Accessed: March 25, 2008)

[50] IA (Wikipedia):http://en.wikipedia.org/wiki/Information_architecture (Last Accessed: March 25, 2008)

[51] The many faces of information architecture: http://www.steptwo.com.au/papers/kmc_iafaces/index.html (Last Accessed: March 25, 2008)

[52] Interaction design (Wikipedia): http://en.wikipedia.org/wiki/Interaction_design (Last Accessed: March 25, 2008)

[53] Interaction design patterns: http://www.interaction-design.org/encyclopedia/interaction_design_patterns.html (Last Accessed: March 25, 2008)

[54] Introducing interaction design pattern: http://www.boxesandarrows.com/view/introducing_interaction_design (Last Accessed: March 25, 2008)

[55] Interface design: http://en.wikipedia.org/wiki/Interface_design (Last Accessed: March 25, 2008)

[56] Interface design principals: http://www.sylvantech.com/~talin/projects/ui_design.html (Last Accessed: March 25, 2008)

[57] Tips and techniques: http://www.ambysoft.com/essays/userInterfaceDesign.html (Last Accessed: March 25, 2008)

[58] Usability (Wikipedia): http://en.wikipedia.org/wiki/Usability (Last Accessed: March 25, 2008)

[59] Introduction to usability: http://www.useit.com/alertbox/20030825.html (Last Accessed: March 25, 2008)

[60] UCD: http://en.wikipedia.org/wiki/User-centered_design (Last Accessed: March 26, 2008)

[61] UCD Phases: http://www.usabilityprofessionals.org/usability_resources/about_usability/what_is_ucd.html (Last Accessed: March 26, 2008)

[62] Guideline: http://en.wikipedia.org/wiki/Guideline(Last Accessed: March 26, 2008)

[63] HIG guideline: http://en.wikipedia.org/wiki/Human_interface_guidelines(Last Accessed: March 26, 2008)

[64] Apple guidelines: http://developer.apple.com/documentation/UserExperience (Last accessed: March 27, 2008)

[65] Microsoft guidelines: http://msdn2.microsoft.com/en-us/library/aa185848.aspx (Last accessed: March 27, 2008)

[66] Microsoft guidelines: http://msdn2.microsot.com/en-us/library/aa511258.aspx (Last accessed: March 27, 2008)

[67] KDE4 guidelines: http://wiki.openusability.org/guidelines/index.php/Main_Page (Last accessed: March 27, 2008)

[68] GNOME HIG 2.0: http://library.gnome.org/devel/hig-book/stable/ (Last accessed: March 27, 2008)

[69] Old GNOME HIG: http://developer.gnome.org/projects/gup/hig/ (Last accessed: March 27, 2008)

[70] OSX Human interface guideline: http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/XHIGIntro/chapter_1_section_1.html (Last accessed: December 4, 2007)

[71] OSX technology overview: http://developer.apple.com/documentation/MacOSX/Conceptual/OSX_Technology_Overview/index.html#//apple_ref/doc/uid/TP40001067 (Last accessed: December 4, 2007)

[72] Apple User Experience: http://developer.apple.com/ue (Last accessed: December 4, 2007)

[73] Apple Human Interface Guidelines, Apple Developer Connection Publication (Last Revision: 2006-10-03) Source: ADC Official site: http://developer.apple.com (Last accessed: December 4, 2007)

[74] OSNews hosted 12 usability questions http://www.osnews.com/story/2997 (Last accessed: March 27, 2008)

[75] Package Management System (Wikipedia): http://en.wikipedia.org/wiki/Package_management_system(Last accessed: March 27, 2008)

[76] RPM Official Websitehttp://www.rpm.org/(Last accessed: March 27, 2008)

[77]RPM Package Manager (Wikipedia): http://en.wikipedia.org/wiki/RPM_Package_Manager(Last accessed: March 27, 2008)

[78] DEB file format (Wikipedia): http://en.wikipedia.org/wiki/Deb_%28file_format%29(Last accessed: March 27, 2008)

[79] Conary Package Manager: http://en.wikipedia.org/wiki/Conary_%28package_manager%29(Last accessed: March 27, 2008)

[80] Linux kernel: http://tldp.org/LDP/tlk/tlk.html(Last accessed: March 27, 2008)

[81] Linux kernel anatomy: http://www.ibm.com/developerworks/linux/library/l-linux-kernel/(Last accessed: March 27, 2008)

[82] Linux kernel virtual machine: http://www.ibm.com/developerworks/linux/library/l-linux-kvm/(Last accessed: March 27, 2008)

[83] What is Linux (Wikipedia): http://en.wikipedia.org/wiki/Linux(Last accessed: March 28, 2008)

[84] What is Linux (Linux.com): http://www.linux.com/whatislinux/(Last accessed: March 28, 2008)

[85] Desktop Linux (Wikipedia): http://en.wikipedia.org/wiki/Desktop_Linux(Last accessed: March 28, 2008)

[86] What is Linux (Linux.org): http://www.linux.org/info/(Last accessed: March 28, 2008)

[87] Linux History: http://en.wikipedia.org/wiki/History_of_Linux(Last accessed: March 28, 2008)

[88] Linux information: http://searchenterpriselinux.techtarget.com/sDefinition/0,,sid39_gci212482,00.html(Last accessed: March 28, 2008)

[89] Linux is not Unix: http://www.linuxdevcenter.com/pub/a/linux/2005/10/06/what-is-linux.html(Last accessed: March 28, 2008)

[90] Installing Linux: http://www.linuxdevcenter.com/pub/a/linux/2005/10/06/what-is-linux.html?page=2(Last accessed: March 28, 2008)

[91] Why Linux (article): http://computer.howstuffworks.com/question246.htm(Last accessed: March 25, 2008)

[92] Why Linux (article)http://www.seul.org/docs/whylinux.html(Last accessed: March 25, 2008)

[93] Linux as desktop OS: http://www.desktoplinux.com/articles/AT5836989728.html(Last accessed: March 25, 2008)

[94] Why Linux (article): http://www.linuxlinks.com/local/why.shtml(Last accessed: March 25, 2008)

[95] Open Source (Wikipedia): http://en.wikipedia.org/wiki/Open_source(Last accessed: March 25, 2008)

[96] Open source documentation: http://opensource.org/docs/osd(Last accessed: March 25, 2008)

[97] Linux directory structure: http://slashmedia.wordpress.com/2007/12/23/linux-directory-structure/(Last accessed: March 27, 2008)

[98] Linux directory structure: http://www.tuxfiles.org/linuxhelp/linuxdir.html(Last accessed: March 27, 2008)

[99] Desktop environment: http://en.wikipedia.org/wiki/Desktop_environment(Last accessed: March 27, 2008)

[100] X window DEshttp://en.wikipedia.org/wiki/Comparison_of_X_Window_System_desktop_environments(Last accessed: March 27, 2008)

[101] MP3 (Wikipedia) http://en.wikipedia.org/wiki/MP3(Last accessed: March 27, 2008)

[102] MP3 Royalty rate http://www.mp3licensing.com/royalty/software.html(Last accessed: March 27, 2008)

[103] MP3 Licensing: http://www.mp3licensing.com/licensees/index.asp(Last accessed: March 27, 2008)

[104] Gstreamer: http://www.gstreamer.net/(Last accessed: March 27, 2008)

[105] System requirements: https://help.ubuntu.com/community/Installation/SystemRequirements(Last accessed: March 27, 2008)

[106] Slackware: www.slackware.com(Last accessed: March 27, 2008)

[107] Debian: www.debian.org(Last accessed: March 27, 2008)

[108] Fedora: www.fedoraproject.org(Last accessed: March 27, 2008)

[109] Mandriva: www.mandriva.com(Last accessed: March 27, 2008)

[110] Ubuntu: www.ubuntu.com(Last accessed: March 27, 2008)

[111] SuSE: www.opensuse.org(Last accessed: March 27, 2008)

[112] Gentoo: www.gentoo.org(Last accessed: March 27, 2008)

[123] Linux Mint: www.linuxmint.com(Last accessed: March 27, 2008)

[124] PCLinuxOS: www.pclinuxos.com(Last accessed: March 27, 2008)

[125] Sabayon: www.sabayonlinux.org(Last accessed: March 27, 2008)

[126] CentOS: www.centos.org(Last accessed: March 27, 2008)

[127] MEPIS: www.mepis.org(Last accessed: March 27, 2008)

[128] DSL: www.damnsmalllinux.org(Last accessed: March 27, 2008)

[129] Zenwalk: www.zenwalk.org(Last accessed: March 27, 2008)

[130] Slax: www.slax.org(Last accessed: March 27, 2008)

[131] DreamLinux: www.dreamlinux.com.br(Last accessed: March 27, 2008)

[132] gOS: www.thinkgos.com(Last accessed: March 27, 2008)

[133] Linux distribution: http://www.linux.org/dist/(Last accessed: March 27, 2008)

[134] Distrowatch major distros: http://distrowatch.com/dwres.php?resource=major(Last accessed: March 27, 2008)

[135] Distrowatch popular distro list: http://distrowatch.com/stats.php?section=popularity(Last accessed: March 27, 2008)

[136] Next Mac will use Linux kernel: http://www.linux.org/news/LO2007/mac_kernel.html(Last accessed: March 27, 2008)

[137] Next Mac will use Linux kernel: http://plug.org.in/pipermail/plug-mail/2007-April/001945.html(Last accessed: March 27, 2008)

[138] Apple Application Bundle Anatomy: http://developer.apple.com/documentation/CoreFoundation/Conceptual/CFBundles/Concepts/BundleAnatomy.html(Last accessed: March 27, 2008)

[139] Apple Application Bundle tricks: http://www.mactipsandtricks.com/articles/Wiley_HT_appBundles2.lasso(Last accessed: March 27, 2008)

[140] Canonical official website: http://www.canonical.com(Last accessed: March 8, 2008)

[141] Microsoft User Experience: http://www.microsoft.com/college/ip_userexp.mspx(Last accessed: March 8, 2008)

[142] Apple User Experience: http://developer.apple.com/ue/(Last accessed: March 8, 2008)

[143] IBM USER:http://www.almaden.ibm.com/cs/disciplines/user(Last accessed: March 8, 2008)

Forum Thread References:

Note: There are lot of user comments and suggestion on the real threads. As the threads are modified to better presentation, visiting the real threads is suggested.

Links of the threads: (Created on January 26, 2008)

a. **UbuntuForums:**http://ubuntuforums.org/showthread.php?t=678856
b. **JCXP Forums:**http://www.jcxp.net/forums/index.php?showtopic=25580&st=0
c. **TunesBDForums:**http://www.tunesbd.net/forum/showthread.php?p=311696
(Last accessed: January 28, 2008)

Screenshot on page 47: http://www.gnome-look.org/CONTENT/content-pre1/45837-1.jpg (Last accessed: January 28, 2008

[END OF DOCUMENT]