Development of Computer and Software Engineering

Nathan Finch

DEVELOPMENT OF COMPUTER AND SOFTWARE ENGINEERING

DEVELOPMENT OF COMPUTER AND SOFTWARE ENGINEERING

Nathan Finch

BIBLIOTEX
Digital Library

Development of Computer and Software Engineering
by Nathan Finch

www.bibliotex.com

Ebook ISBN: 9781984664044

Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

1

Software Engineering

PROCESS

Software engineering process and practices are the structures imposed on development of a software product. There are different models of software process (software lifecycle is a synonym) used in different organizations and industries. RAL has identified three levels of software process for its projects. These levels balance the different needs of different types of projects. Scaling the process to the project is vital to its success, too much process can be as problematic as too little; too much process can slow down a purely R&D exploration, too little process can slow down a large development project with hard deliverables.

HIGH QUALITY SOFTWARE

Developing high quality software is hard, especially when

the interpretation of term "quality" is patchy based on the environment in which it is used. In order to know if quality has been achieved, or degraded, it has to be measured, but determining what to measure and how is the difficult part. Software Quality Attributes are the benchmarks that describe system's intended behaviour within the environment for which it was built.

The quality attributes provide the means for measuring the fitness and suitability of a product. Software architecture has a profound affect on most qualities in one way or another, and software quality attributes affect architecture. Identifying desired system qualities before a system is built allows system designer to mold a solution (starting with its architecture) to match the desired needs of the system within the context of constraints (available resources, interface with legacy systems, etc). When a designer understands the desired qualities before a system is built, then the likelihood of selecting or creating the right architecture is improved.

STATEMENTS

Both statements are useless as they provide no tangible way of measuring the behaviour of the system. The quality attributes must be described in terms of scenarios, such as "when 100 users initiate 'complete payment' transition, the payment component, under normal circumstances, will process the requests with an average latency of three seconds." This statement, or scenario, allows an architect to make quantifiable arguments about a system.

A scenario defines the source of stimulus (users), the actual stimulus (initiate transaction), the artifact affected (payment component), the environment in which it exists (normal

operation), the effect of the action (transaction processed), and the response measure (within three seconds). Writing such detailed statements is only possible when relevant requirements have been identified and an idea of components has been proposed.

Qualities

Scenarios help describe the qualities of a system, but they don't describe how they will be achieved. Architectural tactics describe how a given quality can be achieved. For each quality there may be a large set of tactics available to an architect. It is the architect's job to select the right tactic in light of the needs of the system and the environment.

For example, a performance tactics may include options to develop better processing algorithms, develop a system for parallel processing, or revise event scheduling policy. Whatever tactic is chosen, it must be justified and documented.

Software Qualities

It would be naïve to claim that the list below is as a complete taxonomy of all software qualities – but it's a solid list of general software qualities compiled from respectable sources. Domain specific systems are likely to have an additional set of qualities in addition to the list below. System qualities can be categorized into four parts: runtime qualities, non-runtime qualities, business qualities, and architecture qualities.

Each of the categories and its associated qualities are briefly described below. Other articles on this site provide more information about each of the software quality attributes

listed below, their applicable properties, and the conflicts the qualities.

Types of softare qualities

It defines six software quality attributes, also called quality characteristics:

1. *Functionality*: Are the required functions available, including interoperabilithy and security
2. *Reliability*: Maturity, fault tolerance and recoverability
3. *Usability*: How easy it is to understand, learn, operate the software system
4. *Efficiency*: Performance and resource behaviour
5. *Maintainability*: How easy is it to modify the software
6. *Portability*: Can the software easily be transferred to another environment, including installability

Product Revision

The product revision perspective identifies quality factors that influence the ability to change the software product, these factors are:

- Maintainability, the ability to find and fix a defect.
- Flexibility, the ability to make changes required as dictated by the business.
- Testability, the ability to Validate the software requirements.

Product Transition

The product transition perspective identifies quality factors that influence the ability to adapt the software to new environments:

- Portability, the ability to transfer the software from one environment to another.

- Reusability, the ease of using existing software components in a different context.
- Interoperability, the extent, or ease, to which software components work together.

Software Engineering Process

The elements of a software engineering process are generally enumerated as:

- Marketing Requirements
- System-Level Design
- Detailed Design
- Implementation
- Integration
- Field Testing
- Support

No element of this process ought to commence before the earlier ones are substantially complete, and whenever a change is made to some element, all dependent elements ought to be reviewed or redone in light of that change. It's possible that a given module will be both specified and implemented before its dependent modules are fully specified — this is called advanced development or research.

It is absolutely essential that every element of the software engineering process include several kinds of *review:* peer review, mentor/management review, and cross-disciplinary review. Software engineering elements (whether documents or source code) must have version numbers and auditable histories. "Checking in" a change to an element should require some form of review, and the depth of the review should correspond directly to the scope of the change.

Marketing Requirements

The first step of a software engineering process is to create a document which describes the target customers and their reason for needing this product, and then goes on to list the features of the product which address these customer needs. The Marketing Requirements Document (MRD) is the battleground where the answer to the question "What should we build, and who will use it?" is decided.

In many failed projects, the MRD was handed down like an inscribed stone tablet from marketing to engineering, who would then gripe endlessly about the laws of physics and about how they couldn't actually build that product since they had no ready supply of Kryptonite or whatever. The MRD is a joint effort, with engineering not only reviewing but also writing a lot of the text.

System-Level Design

This is a high-level description of the product, in terms of "modules" (or sometimes "programmes") and of the interaction between these modules. The goals of this document are first, to gain more confidence that the product could work and could be built, and second, to form a basis for estimating the total amount of work it will take to build it. The system-level design document should also outline the system-level testing plan, in terms of customer needs and whether they would be met by the system design being proposed.

Detailed Design

The detailed design is where every module called out in the system-level design document is described in detail. The

interface (command line formats, calling API, externally visible data structures) of each module has to be completely determined at this point, as well as dependencies between modules. Two things that will evolve out of the detailed design is a PERT or GANT chart showing what work has to be done and in what order, and more accurate estimates of the time it will take to complete each module.

Every module needs a unit test plan, which tells the implementor what test cases or what kind of test cases they need to generate in their unit testing in order to verify functionality. Note that there are additional, nonfunctional unit tests which will be discussed later.

Implementation

Every module described in the detailed design document has to be implemented. This includes the small act of coding or programming that is the heart and soul of the software engineering process. It's unfortunate that this small act is sometimes the only part of software engineering that is taught (or learned), since it is also the only part of software engineering which can be effectively self-taught.

A module can be considered implemented when it has been created, tested, and successfully used by some other module (or by the system-level testing process). Creating a module is the old edit-compile-repeat cycle. Module testing includes the unit level functional and regression tests called out by the detailed design, and also performance/stress testing, and code coverage analysis.

Integration

When all modules are nominally complete, system-level

integration can be done. This is where all of the modules move into a single source pool and are compiled and linked and packaged as a system. Integration can be done incrementally, in parallel with the implementation of the various modules, but it cannot authoritatively approach "doneness" until all modules are substantially complete.

Integration includes the development of a system-level test. If the built package has to be able to install itself (which could mean just unpacking a tarball or copying files from a CD-ROM) then there should be an automated way of doing this, either on dedicated crash and burn systems or in containerized/simulated environments. Sometimes, in the middleware arena, the package is just a built source pool, in which case no installation tools will exist and system testing will be done on the as-built pool. Once the system has been installed (if it is installable), the automated system-level testing process should be able to invoke every public command and call every public entry point, with every possible reasonable combination of arguments.

If the system is capable of creating some kind of database, then the automated system-level testing should create one and then use external (separately written) tools to verify the database's integrity. It's possible that the unit tests will serve some of these needs, and all unit tests should be run in sequence during the integration, build, and packaging process.

Field Testing

Field testing usually begins internally. That means employees of the organization that produced the software package will run it on their own computers. This should

ultimately include all "production level" systems — desktops, laptops, and servers.

The statement you want to be able to make at the time you ask customers to run a new software system (or a new version of an existing software system) is "we run it ourselves." The software developers should be available for direct technical support during internal field testing. Ultimately it will be necessary to run the software externally, meaning on customers' (or prospective customers') computers. It's best to pick "friendly" customers for this exercise since it's likely that they will find a lot of defects — even some trivial and obvious ones — simply because their usage patterns and habits are likely to be different from those of your internal users.

The software developers should be close to the front of the escalation path during external field testing. Defects encountered during field testing need to be triaged by senior developers and technical marketers, to determine which ones can be fixed in the documentation, which ones need to be fixed before the current version is released, and which ones can be fixed in the next release (or never).

Support

Software defects encountered either during field testing or after the software has been distributed should be recorded in a tracking system. These defects should ultimately be assigned to a software engineer who will propose a change to either the definition and documentation of the system, or the definition of a module, or to the implementation of a module. These changes should include additions to the unit and/or system-level tests, in the form of a regression test to

show the defect and therefore show that it has been fixed (and to keep it from recurring later).

Just as the MRD was a joint venture between engineering and marketing, so it is that support is a joint venture between engineering and customer service. The battlegrounds in this venture are the bug list, the categorization of particular bugs, the maximum number of critical defects in a shippable software release, and so on.

Function Point Based Measure

Introduction

Function Points and the Function Point Model are measurement tools to manage software. Function Points, with other business measures, become Software Metrics. Function Points measure Software size. Function Points measure functionality by objectively measuring functional requirements.

Function Points quantify and document assumptions in Estimating software development. Function Points and Function Point Analysis are objective; Function Points are consistent, and Function Points are auditable. Function Points are independent of technology.

Function Points even apply regardless of design. But Function Points do not measure people directly. Function Points is a macro tool, not a micro tool. Function Points are the foundation of a Software Metrics programme. Software Metrics include Function Points as a normalizing factor for comparison. Function Points in conjunction with time yield Productivity Software Metrics. Function Points in conjunction with defects yield Quality Software Metrics. Function Points

with costs provide Unit Cost, Return on Investment, and Efficiency Software Metrics, never before available.

Function Points connect Software Metrics to measure Risk. Function Points can verify Staffing metrics. Function Points can evaluate Build, Buy and/or Outsource decisions. Function Points combine with SEI CMM measures, TQM measures, Baldrige measures, ISO and/or other software and business measures to prove overall status and value.

Doing The Right Things!

Function Points and Usage or Volume measures create Software Metrics that demonstrate an organization's ability to Leverage software's business impact. The Leverage of E Commerce is obvious, but until now unmeasured. Function Points support Customer Satisfaction measures to create Value Software Metrics. Function Points and Skill measures provide Software Metrics for Employee Service Level Agreements to meet current and future company skill needs. Function Points can even measure the Corporate Vision and generate Software Metrics to report progress towards meeting it.

Function Points, Function Point Analysis, the Function Point Model, Supplemental Software Measures, and the Software Metrics they generate, are only the third measure that transcend every part of every organization. (The other two are time and money.) Without them your organization is only two thirds whole.

Definition

A function point is a unit of measurement to express the amount of business functionality an information system

provides to a user. Function points are the units of measure used by the IFPUG Functional Size Measurement Method.

The IFPUG FSM Method is an ISO recognized software metric to size an information system based on the functionality that is perceived by the user of the information system, independent of the technology used to implement the information system.

Function points were defined in 1979 in New Way of Looking at Tools by Allan Albrecht at IBM. The functional user requirements of the software are identified and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces.

Once the function is identified and categorized into a type, it is then assessed for complexity and assigned a number of function points. Each of these functional user requirements maps to an end-user business function, such as a data entry for an Input or a user query for an Inquiry.

This distinction is important because it tends to make the functions measured in function points map easily into user-oriented requirements, but it also tends to hide internal functions (*e.g.* algorithms), which also require resources to implement.

Over the years there have been different approaches proposed to deal with this perceived weakness, however there is no ISO recognized FSM Method that includes algorithmic complexity in the sizing result. The variations of the Albrecht based IFPUG method designed to make up for this (and other weaknesses) include:

- *Early and easy function points*: Adjusts for problem and data complexity with two questions that yield a

somewhat subjective complexity measurement; simplifies measurement by eliminating the need to count data elements.

- *Engineering function points*: Elements (variable names) and operators (*e.g.*, arithmetic, equality/inequality, Boolean) are counted. This variation highlights computational function.
- *Bang measure*: Defines a function metric based on twelve primitive (simple) counts that affect or show Bang, defined as "the measure of true function to be delivered as perceived by the user." Bang measure may be helpful in evaluating a software unit's value in terms of how much useful function it provides, although there is little evidence in the literature of such application. The use of Bang measure could apply when re-engineering (either complete or piecewise) is being considered, as discussed in Maintenance of Operational Systems—An Overview.

Five Components of Function Points

Data Functions

- Internal Logical Files
- External Interface Files

Transactional Functions

- External Inputs
- External Outputs
- External Inquiries

Internal Logical Files

The first data function allows users to utilize data they are

responsible for maintaining. For example, a pilot may enter navigational data through a display in the cockpit prior to departure.

The data is stored in a file for use and can be modified during the mission. Therefore the pilot is responsible for maintaining the file that contains the navigational information. Logical groupings of data in a system, maintained by an end user, are referred to as Internal Logical Files (ILF).

External Interface Files

The second Data Function a system provides an end user is also related to logical groupings of data. In this case the user is not responsible for maintaining the data. The data resides in another system and is maintained by another user or system.

The user of the system being counted requires this data for reference purposes only. For example, it may be necessary for a pilot to reference position data from a satellite or ground-based facility during flight.

The pilot does not have the responsibility for updating data at these sites but must reference it during the flight. Groupings of data from another system that are used only for reference purposes are defined as External Interface Files (EIF).

The remaining functions address the user's capability to access the data contained in ILFs and EIFs.

This capability includes maintaining, inquiring and outputting of data. These are referred to as Transactional Functions.

External Input

The first Transactional Function allows a user to maintain Internal Logical Files (ILFs) through the ability to add, change and delete the data. For example, a pilot can add, change and delete navigational information prior to and during the mission.

In this case the pilot is utilizing a transaction referred to as an External Input (EI). An External Input gives the user the capability to maintain the data in ILF's through adding, changing and deleting its contents.

External Output

The next Transactional Function gives the user the ability to produce outputs. For example a pilot has the ability to separately display ground speed, true air speed and calibrated air speed.

The results displayed are derived using data that is maintained and data that is referenced. In function point terminology the resulting display is called an External Output (EO).

External Inquiries

The final capability provided to users through a computerized system addresses the requirement to select and display specific data from files. To accomplish this a user inputs selection information that is used to retrieve data that meets the specific criteria.

In this situation there is no manipulation of the data. It is a direct retrieval of information contained on the files. For example if a pilot displays terrain clearance data that was

previously set, the resulting output is the direct retrieval of stored information. These transactions are referred to as External Inquiries (EQ).

In addition to the five functional components described above there are two adjustment factors that need to be considered in Function Point Analysis.

Functional Complexity

The first adjustment factor considers the Functional Complexity for each unique function.

Functional Complexity is determined based on the combination of data groupings and data elements of a particular function.

The number of data elements and unique groupings are counted and compared to a complexity matrix that will rate the function as low, average or high complexity.

Each of the five functional components (ILF, EIF, EI, EO and EQ) has its own unique complexity matrix. The following is the complexity matrix for External Outputs.

	1-5 DETs	**6 - 19 DETs**	**20+ DETs**
0 or 1 FTRs	L	L	A
2 or 3 FTRs	L	A	H
4+ FTRs	A	H	H

Complexity	**UFP**
L (Low)	4
A (Average)	5
H (High)	7

Using the examples given above and their appropriate complexity matrices, the function point count for these functions would be:

Function Name	Function Type	Record Element Type	Data Element Type	File Types Referenced	Unadjusted FPs
Navigational data	ILF	3	36	n/a	10
Positional data	EIF	1	3	n/a	5
Navigational data - add	EI	n/a	36	1	4
Navigational data - change	EI	n/a	36	1	4
Navigational data - delete	EI	n/a	3	1	3
Ground speed display	EO	n/a	20	3	7
Air speed display	EO	n/a	20	3	7
Calibrated air speed display	EO	n/a	20	3	7
Terrain clearance display	EQ	n/a	1	1	3
Total unadjusted count					50 UFPs

All of the functional components are analysed in this way and added together to derive an Unadjusted Function Point count.

Value Adjustment Factor

The Unadjusted Function Point count is multiplied by the second adjustment factor called the Value Adjustment Factor. This factor considers the system's technical and operational characteristics and is calculated by answering 14 questions. The factors are:

Data Communications

The data and control information used in the application are sent or received over communication facilities.

Distributed Data Processing

Distributed data or processing functions are a characteristic of the application within the application boundary.

Performance

Application performance objectives, stated or approved by the user, in either response or throughput, influence (or will influence) the design, development, installation and support of the application.

Heavily Used Configuration

A heavily used operational configuration, requiring special design considerations, is a characteristic of the application.

Transaction Rate

The transaction rate is high and influences the design, development, installation and support.

On-line Data Entry

On-line data entry and control information functions are provided in the application.

End -User Efficiency

The on-line functions provided emphasize a design for end-user efficiency.

On-line Update

The application provides on-line update for the internal logical files.

Complex Processing

Complex processing is a characteristic of the application.

Reusability

The application and the code in the application have been specifically designed, developed and supported to be usable in other applications.

Installation Ease

Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase.

Operational Ease

Operational ease is a characteristic of the application. Effective start-up, backup and recovery procedures were provided and tested during the system test phase.

Multiple Sites

The application has been specifically designed, developed and supported to be installed at multiple sites for multiple organizations.

Facilitate Change

The application has been specifically designed, developed and supported to facilitate change. Each of these factors is scored based on their influence on the system being counted. The resulting score will increase or decrease the Unadjusted Function Point count by 35%. This calculation provides us with the Adjusted Function Point count.

Approach to Counting Function Points

There are several approaches used to count function points. Q/P Management Group, Inc. has found that a structured workshop conducted with people who are knowledgeable of the functionality provided through the application is an efficient, accurate way of collecting the necessary data. The workshop approach allows the counter

to develop a representation of the application from a functional perspective and educate the participants about function points. Function point counting can be accomplished with minimal documentation. However, the accuracy and efficiency of the counting improves with appropriate documentation. Examples of appropriate documentation are:

- Design specifications
- Display designs
- Data requirements (Internal and External)
- Description of user interfaces

Function point counts are calculated during the workshop and documented with both a diagram that depicts the application and worksheets that contain the details of each function discussed.

Benefits of Function Point Analysis

Organizations that adopt Function Point Analysis as a software metric realise many benefits including: improved project estimating; understanding project and maintenance productivity; managing changing project requirements; and gathering user requirements. Each of these is discussed below. Estimating software projects is as much an art as a science. While there are several environmental factors that need to be considered in estimating projects, two key data points are essential. The first is the size of the deliverable. The second addresses how much of the deliverable can be produced within a defined period of time.

Size can be derived from Function Points, as described above. The second requirement for estimating is determining

how long it takes to produce a function point. This delivery rate can be calculated based on past project performance or by using industry benchmarks.

The delivery rate is expressed in function points per hour (FP/Hr) and can be applied to similar proposed projects to estimate effort (*i.e.* Project Hours = estimated project function points FP/Hr). Productivity measurement is a natural output of Function Points Analysis.

Since function points are technology independent they can be used as a vehicle to compare productivity across dissimilar tools and platforms. More importantly, they can be used to establish a productivity rate (*i.e.* FP/Hr) for a specific tool set and platform. Once productivity rates are established they can be used for project estimating as described above and tracked over time to determine the impact continuous process improvement initiatives have on productivity.

In addition to delivery productivity, function points can be used to evaluate the support requirements for maintaining systems. In this analysis, productivity is determined by calculating the number of function points one individual can support for a given system in a year (*i.e.* FP/FTE year). When compared with other systems, these rates help to identify which systems require the most support. The resulting analysis helps an organization develop a maintenance and replacement strategy for those systems that have high maintenance requirements.

Managing Change of Scope for an in-process project is another key benefit of Function Point Analysis. Once a project has been approved and the function point count has been established, it becomes a relatively easy task to identify, track

and communicate new and changing requirements. As requests come in from users for new displays or capabilities, function point counts are developed and applied against the rate. This result is then used to determine the impact on budget and effort. The user and the project team can then determine the importance of the request against its impact on budget and schedule.

At the conclusion of the project the final function point count can be evaluated against the initial estimate to determine the effectiveness of requirements gathering techniques. This analysis helps to identify opportunities to improve the requirements definition process.

Communicating Functional Requirements was the original objective behind the development of function points. Since it avoids technical terminology and focuses on user requirements it is an excellent vehicle to communicate with users. The techniques can be used to direct customer interviews and document the results of Joint Application Design (JAD) sessions. The resulting documentation provides a framework that describes user and technical requirements.
In conclusion, Function Point Analysis has proven to be an accurate technique for sizing, documenting and communicating a system's capabilities. It has been successfully used to evaluate the functionality of real-time and embedded code systems, such as robot based warehouses and avionics, as well as traditional data processing.

2

Development of Computer

Computer Types

A computer is one of the most brilliant inventions of mankind. Thanks to the computer technology, we were able to achieve an efficient storage and processing of data; we could rest our brains by employing computer memory capacities for storage of the information.

Owing to computers, we have been able speed up daily work, carry out critical transactions and achieve accuracy and precision in work output. The computers of the earlier years were of the size of a large room and were required to consume huge amounts of electric power. However, with the advancing technology, computers have shrunk to the size of a small watch.

Depending on the processing powers and sizes of computers, they have been classified under various types. Let us look at the classification of computers. Based on the

operational principle of computers, they are categorized as analog computers and hybrid computers.

Analog Computers

These are almost extinct today. These are different from a digital computer because an analog computer can perform several mathematical operations simultaneously. It uses continuous variables for mathematical operations and utilizes mechanical or electrical energy.

Hybrid Computers

These computers are a combination of both digital and analog computers. In this type of computers, the digital segments perform process control by conversion of analog signals to digital ones.

Mainframe Computers

Large organizations use mainframes for highly critical applications such as bulk data processing and ERP. Most of the mainframe computers have the capacities to host multiple operating systems and operate as a number of virtual machines and can thus substitute for several small servers.

Microcomputers

A computer with a microprocessor and its central processing unit is known as a microcomputer. They do not occupy space as much as mainframes. When supplemented with a keyboard and a mouse, microcomputers can be called as personal computers. A monitor, a keyboard and other similar input output devices, computer memory in the form of RAM and a power supply unit come packaged in a microcomputer. These computers can fit on desks or tables

and serve as the best choices for single-user tasks. Personal computers come in a variety of forms such as desktops, laptops and personal digital assistants. Let us look at each of these types of computers.

Desktops

A desktop is intended to be used on a single location. The spare parts of a desktop computer are readily available at relative lower costs. Power consumption is not as critical as that in laptops. Desktops are widely popular for daily use in workplaces and households.

Laptops

Similar in operation to desktops, laptop computers are miniaturized and optimized for mobile use. Laptops run on a single battery or an external adapter that charges the computer batteries. They are enabled with an inbuilt keyboard, touch pad acting as a mouse and a liquid crystal display. Its portability and capacity to operate on battery power have served as a boon for mobile users.

Personal Digital Assistants (PDAs)

It is a handheld computer and popularly known as a palmtop. It has a touch screen and a memory card for storage of data. PDAs can also be effectively used as portable audio players, web browsers and smart phones. Most of them can access the Internet by means of Bluetooth or Wi-Fi communication.

Minicomputers

In terms of size and processing capacity, minicomputers lie in between mainframes and microcomputers.

Minicomputers are also called mid-range systems or workstations. The term began to be popularly used in the 1960s to refer to relatively smaller third generation computers. They took up the space that would be needed for a refrigerator or two and used transistor and core memory technologies. The 12-bit PDP-8 minicomputer of the Digital Equipment Corporation was the first successful minicomputer.

Supercomputers

The highly calculation-intensive tasks can be effectively performed by means of supercomputers. Quantum physics, mechanics, weather forecasting, molecular theory are best studied by means of supercomputers. Their ability of parallel processing and their well-designed memory hierarchy give the supercomputers, large transaction processing powers.

Wearable Computers

A record-setting step in the evolution of computers was the creation of wearable computers. These computers can be worn on the body and are often used in the study of behaviour modeling and human health. Military and health professionals have incorporated wearable computers into their daily routine, as a part of such studies.

When the users' hands and sensory organs are engaged in other activities, wearable computers are of great help in tracking human actions. Wearable computers are consistently in operation as they do not have to be turned on and off and are constantly interacting with the user.

These were some of the different types of computers available today. Looking at the rate of the advancement in

technology, we can definitely look forward to many more types of computers in the near future.

Computer Systems Architecture

The discipline that defines the conceptual structure and functional behaviour of a computer system. It is analogous to the architecture of a building, determining the overall organization, the attributes of the component parts, and how these parts are combined.

It is related to, but different from, computer implementation. Architecture consists of those characteristics which affect the design and development of software programs, whereas implementation focuses on those characteristics which determine the relative cost and performance of the system.

The architect's main goal has long been to produce a computer that is as fast as possible, within a given set of cost constraints. Over the years, other goals have been added, such as making it easier to run multiple programs concurrently or improving the performance of programs written in higher-level languages.

A computer system consists of four major components: storage, processor, peripherals, and input/output (communication). The storage system is used to keep data and programs; the processor is the unit that controls the operation of the system and carries out various computations; the peripheral devices are used to communicate with the outside world; and the input/output system allows the previous components to communicate with one another.

Storage

The storage or memory of a computer system holds the data that the computer will process and the instructions that indicate what processing is to be done. In a digital computer, these are stored in a form known as binary, which means that each datum or instruction is represented by a series of bits. Bits are conceptually combined into larger units called bytes (usually 8 bits each) and words (usually 8 to 64 bits each). A computer will generally have several different kinds of storage devices, each organized to hold one or more words of data. These types include registers, main memory, and secondary or auxiliary storage.

Registers are the fastest and most costly storage units in a computer. Normally contained within the processing unit, registers hold data that are involved with the computation currently being performed.

Main memory holds the data to be processed and the instructions that specify what processing is to be done. A major goal of the computer architect is to increase the effective speed and size of a memory system without incurring a large cost penalty. Two prevalent techniques for increasing effective speed are interleaving and caching, while virtual memory is a popular way to increase the effective size. Interleaving involves the use of two or more independent memory systems, combined in a way that makes them appear to be a single, faster system. With caching, a small, fast memory system contains the most frequently used words from a slower, larger main memory.

Virtual memory is a technique whereby the programmer is given the illusion of a very large main memory, when in

fact it has only a modest size. This is achieved by placing the contents of the large, "virtual" memory on a large but slow auxiliary storage device, and bringing portions of it into main memory, as required by the programs, in a way that is transparent to the programmer.

Auxiliary memory (sometimes called secondary storage) is the slowest, lowest-cost, and highest-capacity computer storage area. Programs and data are kept in auxiliary memory when not in immediate use, so that auxiliary memory is essentially a long-term storage medium. There are two basic types of secondary storage: sequential and direct-access. Sequential-access secondary storage devices, of which magnetic tape is the most common, permit data to be accessed in a linear sequence. A direct-access device is one whose data may be accessed in any order. Disks and drums are the most commonly encountered devices of this type.

Memory mapping is one of the most important aspects of modern computer memory designs. In order to understand its function, the concept of an address space must be considered. When a program resides in a computer's main memory, there is a set of memory cells assigned to the program and its data. This is known as the program's logical address space. The computer's physical address space is the set of memory cells actually contained in the main memory. Memory mapping is simply the method by which the computer translates between the computer's logical and physical address spaces. The most straightforward mapping scheme involves use of a bias register. Assignment of a different bias value to each program in memory enables the programs to coexist without

interference. Another strategy for mapping is known as paging. This technique involves dividing both logical and physical address spaces into equal-sized blocks called pages. Mapping is achieved by means of a page map, which can be thought of as a series of bias registers.

Processing

A computer's processor (processing unit) consists of a control unit, which directs the operation of the system, and an arithmetic and logic unit, which performs computational operations. The design of a processing unit involves selection of a register set, communication paths between these registers, and a means of directing and controlling how these operate. Normally, a processor is directed by a program, which consists of a series of instructions that are kept in main memory.

Although the process of decoding and executing instructions is often carried out by logic circuitry, the complexity of instruction sets can lead to very large and cumbersome circuits for this purpose. To alleviate this problem, a technique known as microprogramming was developed.

With microprogramming, each instruction is actually a macrocommand that is carried out by a microprogram, written in a microinstruction language. The microinstructions are very simple, directing data to flow between registers, memories, and arithmetic units.

It should be noted that microprogramming has nothing to do with microprocessors. A microprocessor is a processor implemented through a single, highly integrated circuit.

Peripherals and Communication

A typical computer system includes a variety of peripheral devices such as printers, keyboards, and displays. These devices translate electronic signals into mechanical motion or light (or vice versa) so as to communicate with people.

There are two common approaches for connecting peripherals and secondary storage devices to the rest of the computer: The channel and the bus. A channel is essentially a wire or group of wires between a peripheral device and a memory device. A multiplexed channel allows several devices to be connected to the same wire. A bus is a form of multiplexed channel that can be shared by a large number of devices. The overhead of sharing many devices means that the bus has lower peak performance than a channel; but for a system with many peripherals, the bus is more economical than a large number of channels.

A computer controls the flow of data across buses or channels by means of special instructions and other mechanisms. The simplest scheme is known as program-controlled input/output (I/O). Direct memory access I/O is a technique by which the computer signals the device to transmit a block of data, and the data are transmitted directly to memory, without the processor needing to wait.

Interrupts are a form of signal by which a peripheral device notifies a processor that it has completed transmitting data. This is very helpful in a direct memory access scheme, for the processor cannot always predict in advance how long it will take to transmit a block of data. Architects often design elaborate interrupt schemes to simplify the situation where several peripherals are active simultaneously.

Digital Computer

A device that processes numerical information; more generally, any device that manipulates symbolic information according to specified computational procedures. The term digital computer—or simply, computer—embraces calculators, computer workstations, control computers (controllers) for applications such as domestic appliances and industrial processes, data-processing systems, microcomputers, microcontrollers, multiprocessors, parallel computers, personal computers, network servers, and supercomputers.

A digital computer is an electronic computing machine that uses the binary digits (bits) 0 and 1 to represent all forms of information internally in digital form. Every computer has a set of instructions that define the basic functions it can perform. Sequences of these instructions constitute machine-language programs that can be stored in the computer and used to tailor it to an essentially unlimited number of specialized applications. Calculators are small computers specialized for mathematical computations. General-purpose computers range from pocket-sized personal digital assistants (notepad computers), to medium-sized desktop computers (personal computers and workstations), to large, powerful computers that are shared by many users via a computer network. The vast majority of digital computers now in use are inexpensive, special-purpose microcontrollers that are embedded, often invisibly, in such devices as toys, consumer electronic equipment, and automobiles.

The main data-processing elements of a computer reside in a small number of electronic integrated circuits (ICs) that form a microprocessor or central processing unit (CPU). Electronic technology allows a basic instruction such as "add two numbers" to be executed many millions of times per second. Other electronic devices are used for program and data storage (memory circuits) and for communication with external devices and human users (input-output circuits). Nonelectronic (magnetic, optical, and mechanical) devices also appear in computers. They are used to construct input-output devices such as keyboards, monitors (video screens), secondary memories, printers, sensors, and mechanical actuators.

Information is stored and processed by computers in fixed-sized units called words. Common word sizes are 8, 16, 32, and 64 bits. Four-bit words can be used to encode the first 16 integers. By increasing the word size, the number of different items that can be represented and their precision can be made as large as desired. A common word size in personal computers is 32 bits, which allows 2^{32} = 4,294,967,296 distinct numbers to be represented.

Computer words can represent many different forms of information, not just numbers. For example, 8-bit words called characters or bytes are used to encode text symbols (the 10 decimal digits, the 52 upper-and lowercase letters of the English alphabet, and punctuation marks). A widely used code of this type is ASCII (American Standard Code for Information Interchange). Visual information can be reduced to black and white dots (pixels) corresponding to 0's and 1's. Audio information can be digitized by mapping

a small element of sound into a binary word; for example, a compact disk (CD) uses several million 16-bit words to store an audio recording. Logical quantities encountered in reasoning or decision making can be captured by associating 1 with true and 0 with false. Hence, most forms of information are readily reduced to a common, numberlike binary format suitable for processing by computer.

Logic Components

The operation of a digital computer can be viewed at various levels of abstraction, which are characterized by components of different complexity. These levels range from the low, transistor level seen by an electronic circuit designer to the high, system level seen by a computer user. A useful intermediate level is the logic level, where the basic components process individual bits. By using other basic components called gates, logic circuits can be constructed to perform many useful operations.

System Organization

An accumulator is a digital system that constitutes a simple processor capable of executing a few instructions. By introducing more data-processing circuits and registers, as well as control circuits for a larger set of instructions, a practical, general-purpose processor can be constructed. Such a processor forms the "brain" of every computer, and is referred to as its central processing unit. A CPU implemented on a single integrated-circuit chip is called a microprocessor.

A typical computer program is too large to store in the CPU, so another component called the main memory is

used to store a program's instructions and associated data while they are being executed (Fig. 1). Main memory consists of high-speed integrated circuits designed to allow storage and retrieval of information one word at a time. All words in main memory can be accessed with equal ease; hence this is also called a random-access memory (RAM).

A computer program is processed by loading it into main memory and then transferring its instructions and data one word (or a few words) at a time to the CPU for processing. Hence, there is a continual flow of instructions and data words between the CPU and its main memory. As millions of words must be transferred per second, a high-speed communication link is needed between the CPU and main memory. The system bus fills this role.

A computer has input-output (I/O) control circuits and buses to connect it to external input-output devices (also called peripherals). Typical input-output devices are a keyboard, which is an input device, and a printer, which is an output device. Because most computers need more storage space than main memory can supply, they also employ secondary memory units which form part of the computer's input-output subsystem. Common secondary memory devices are hard disk drives, flexible (floppy) disk drives, and magnetic tape units. Compared to main memory, secondary memories employ storage media (magnetic disks and tapes) that have higher capacity and lower cost. However, secondary memories are also significantly slower than main memory.

No explicit instructions are needed for input-output operations if input-output devices share with main memory

the available memory addresses. This is known as memory-mapped input-output, and allows load and store instructions to be used to transfer data between the CPU and input-output devices.

In general, a computer's instruction set should include a selection of instructions of the following three types: (1) Data-transfer instructions that move data unchanged between the CPU, main memory, and input-output devices. (2) Data-processing instructions that perform numerical operations such as add, subtract, multiply, and divide, as well as nonnumerical (logical) operations, such as NOT, AND, EXCLUSIVE-OR, and SHIFT. (3) Program-control instructions that can change the order in which instructions are executed, for example branch, branch-on-zero, call procedure, and return from procedure.

The instruction unit (I unit) of a CPU, also called the program control unit, is responsible for fetching instructions from main memory, using the program counter as the instruction address register. The opcode of a newly fetched instruction *I* is placed in the instruction register. The opcode is then decoded to determine the sequence of actions required to execute *I*.

These may include the loading or storing of data assigned to main memory, in which case the I unit computes all needed addresses and issues all needed control signals to the CPU and the system bus. Data are processed in the CPU's execution unit (E unit), also called the datapath, which contains a set of registers used for temporary storage of data operands, and an arithmetic logic unit (ALU), which contains the main data-processing circuits.

Computer Performance

Improving computer performance

Many computer problems can be solved with free or low-cost products or just by using a few common sense tips to improve performance and keep your PC running for a long time. Computers often freeze or crash when one needs them the most; in the middle of an important presentation, a term paper that's due the next day, or while updating our financial software. Many computer problems can be solved with free or low-cost products or just by using a few common sense tips to improve performance and keep your PC running for a long time.

- *Virus Scan Programme*: The most essential thing to have is a virus scan that is run weekly. Most new computers come with a virus scan already installed. If the computer you are using doesn't have a virus scan there are free scans available online, but one really should be installed on your computer if you spend any time at all online. Spend the time learning how to use your virus scanner. Find out how it's updated-most update automatically-and use the options to set it up to run automatically at a set time every week. Most computer problems can be prevented just by having a virus scanner installed.
- *Run the Scandisk programme*: At least every two months you should run a programme called Scandisk. It is automatically on your computer. Scandisk actually scans your files and even your hard drive and can let you know of any problems it discovers.

To run Scandisk, first make sure everything running in the background on your PC is turned off. To do this press "Ctrl-Alt-Delete" and your close programme box will appear. Highlight each item EXCEPT "explorer" and "systray". Click on "End Task" to close the programme. Then go to your start button and choose "Programmes". Choose "Accessories" at the top of the list. Move your cursor down to "SystemTools" and Choose "Scandisk". A box will appear giving your choices of what you want your computer to scan. You can have it scan your files only or your entire hard drive. Scanning your hard drive will take longer.

- *Run the Defragmenter programme*: Another programme that you should run about every two months is called Disk Defragmenter. Defrag will arrange your files better so your PC can access them faster. It's best to run this programme after the Scandisk programme is finished.

To run Disk Defragmenter go back to the "System Tools" and choose "Disk Defragmenter". It's best to run this programme overnight as it takes a long time. You should not be using these programmes while using other programmes on your computer. If the Scandisk or Defrag programme keeps starting over you may need to run the programmes in Safe Mode, a special diagnostic mode. Read the manual for your PC to find out how to put your PC into Safe Mode.

- *Use a Firewall programme*: For anyone on a DSL or cable connection a personal firewall keeps viruses, hijackers and hackers from your computer. Since you are constantly connected to the internet by using these connections there is a constant threat that

others may try to access your computer. There are free firewall programmes available for download.

- *Run a Spyware programme*: Anyother important rogramme is a spyware search programme. When you download something from the internet sometimes other software is included. This software is called "spyware" and it can do many things to harm your computer, including letting someone from another website see what websites you surf in order to send you advertising. Other spyware tries to steal your passwords or other personal information. You should run a spyware checker at least once a month or at any time you are experiencing problems with your connection or computer speed.
- *Run a hi-jacker search programme*: Another important programme to have is a hi-jacker search programme. A hi-jacker changes your homepage to a different search engine page and can also block links to common search pages. They are trying to force you to use their homepage. Some of the hi-jacker's homepages are not pages you would like your children to see. You should use a hi-jacker search programme at least once a month or whenever you are having a problem with your homepage.
- *Clear your temporary files*: Sometimes your PC's virtual memory will become full. When this happens your computer may run slow, give you error messages or freeze during a programme. This happens because everything you do on your PC-every picture on every webpage, every document you type,

every photo you change-is saved somewhere on your PC, even after you have left that page and closed the document. In order to erase this memory you need to do the following steps:

- Open your internet browser. Go to "Tools", "Internet Options".
- A box will pop open. It should be open to the "General" tab.
- In the centre of the box is a section marked "Temporary Internet Files".
- Click on "Delete Cookies" and "Delete Files" one at a time. This will delete the temporary files that are clogging the PC's memory.

- *Change your history options*: Another way to clean out your computer's memory is to change how many days it saves visited pages. You will find a "history" box in your internet options. It asks how many days you wish to keep pages in history. This is the part of the computer that saves all pages you've visited for as many days as you wish. By saving these pages for a smaller number of days you can clear up more of your PC's memory.
- *Clean out old files*: Periodically going through the files in My Documents or other folders you have set up and deleting or archiving them on a CD can also help your computer's memory and performance. Do you really need to keep a saved copy of last semester's English term paper? If not, delete it. You can also go through the programmes on your PC from the control panel and delete those you no longer

use. Don't delete any of the shared files, though, if it asks, because that can cause more problems.

- *Reboot the PC occasionally*: The last thing that is very important to do but seems simple is just to turn your computer off periodically. Your PC needs to be restarted in order to reset itself after new programmes or equipment is installed. It also erases the virtual memory from the last session and goes through a self-diagnostic to look for any problems. It sounds like a simple solution, but most problems can be fixed simply by restarting your PC.

Computer's Performance

Get Organized

It's likely that more time is wasted in business trying to hunt down a file or e-mail than anything else. There are several reasons for this, and each has a simple fix. First is poor folder structure. If you can't make out your wallpaper image because you have so many files on your desktop, you've got a problem. It's time to put those files in their proper folders.

Think about how you would organize these files if they were paper. You might structure them by project, project type, date, client or process of completion, for example. Don't be afraid to nest your folders as deep as they need to be to make things easier to find. Also, if you find old files that you don't expect to use anytime soon, burn them onto a disc and delete them from your hard drive.

This will save hard drive space and ultimately improve the performance of your computer. A simple trick if you have a bunch of folders or files and you want a particular folder to

always appear at the top is to start the folder name with an underscore. For example, _Projects would appear before Invoices. Another trick is to improve the metadata for quicker searching in the future. Both Mac OSX and Windows Vista offer useful systemwide search, which you can improve by right-clicking on any file, then clicking on Properties (for Windows) or Get Info (for Macs).

Here, you'll want to add useful tags that you might search by later, such as an individual's name, client name, project phase and so on. One final organization trick is to use Smart Folders (Macs) or Search Folders (PCs). You create a smart folder identifying the criteria of the files you want included. This may mean your folder contains files opened within the past five days, files of a certain file type, files modified by a specific user, and so on. The criteria can be filtered by themselves or combined to offer targeted results.

Back it Up

Beyond dumping old files, back up your system in case you accidentally delete something or face a more serious catastrophe. Macs and PCs both have backup software built in. For PCs, it's called the Backup and Restore Centre; for Macs, it's called Time Machine. For both of these to work, you'll need to buy an external hard drive so you'll have someplace to store the data. For most personal computers, a 500GB to 1TB drive should offer plenty of room, and can be had for about $100.

Slow Computer

A slow computer can make you want to pull your hair out. The easiest way to speed it up is to reduce the number of

applications running simultaneously. Applications running in the background sap power, so turn on your radio instead of running your music player; close your calendar app if you're not using it. The next step is to defragment your hard drive. (This is a Windows-only step since Mac OSX operates differently and doesn't need defragging.) Go to Accessories, then System Tools, click Disk Defragmenter, and click Defragment Now. This takes little bits of data scattered all over your computer's hard drive and reorganizes them so your computer can find them faster.

Widgets

If you really need to look at a calendar, use Widgets (for Macs) or Gadgets (for PCs). Widgets are small, typically single-purpose applications that provide useful information at a glance. So instead of firing up your Web browser to find stock quotes, your Google AdSense balance and news headlines, use Widgets. Mac OSX even offers a quick tool built in to the Safari browser that allows you create a Widget by clicking on the Open in Dashboard button (next to Forward and Back buttons), then selecting the part of the Web page you want to track. For example, clip part of a page on eBay to track bidding, and you'll never have to open your browser to keep an eye on things.

Parts of Computer

Introduction

Look inside any computer and you will see items of hardware. There is the motherboard, processors, disks drives, power supply and the memory chips. Many of these items

are interchangeable which mean that the computer can be easily upgraded with additional items of hardware, which just plug into the main system.

Description

Opening the computer exposes both it and you to a myriad of hazards. From nasty cuts from very sharp edges on the sheet metal and circuit cards, to potentially lethal high voltage inside the power supply and its power switch, there are numerous ways to hurt or kill yourself. Over 300 volts can be found inside the power supply of most computers. Most colour monitors have areas of over 20,000 volts inside, *even when they are turned off* and unplugged. Other computer peripherals such as printers, scanners, etc. can also have hazardous areas inside them.

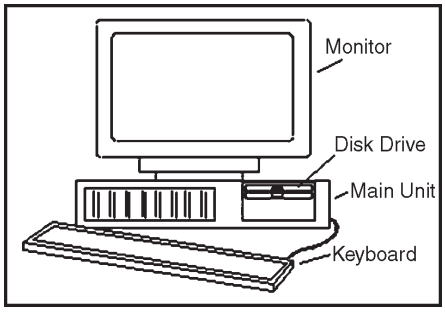
Even with all the dangers to you, it is far more likely you will hurt the computer, than it will hurt you. Improperly handling or installing a card in the computer can cause damage to it or other components. Misalignment of a card in its socket can cause the card or motherboard, or both, to expire the instant power is applied. Simply mishandling an internal component can cause problems. Fingerprints on connectors can cause intermittent trouble, simply touching certain cards, after walking across the floor can transfer enough static electricity to ruin or prematurely age the component. Static electricity, even so low that you can't feel it, can wreck havoc with today's large-scale microelectronics.

All computer boards should be left in their anti-static bags until just before you install them in the computer. You should insure your body is connected to a good ground through a Megohm resistance. Use a professional wrist type grounding-

strap, not a homemade device. *Improper grounding of you could cause a lethal shock hazard.*

In 20 years plus, in the computer business I have seen thousands of computer problems created by the improper handling and installation of circuit boards, hard drives, CD-ROMs, modems, and other devices. Many *computer professionals* don't even fully understand the proper care and handling of the parts inside a computer, and by their ignorance, mishandle a part that should run for over ten years and reduce its life span to less than one year.

Inside the computer,It may look like a complicated mess of wires, and chips, but really it's quite simple.



Computer Hardware Parts

Disks Drives

Disk drives provide a means of storing work, or data. *Floppy disks* are transportable from PC to PC and come in two sizes,

31/2"and 51/4"diameter. Floppy disks provide a means of storing work. You can *write* information onto the disk and *read* information from the disk. Floppy disks are used by inserting them into a disk drive.

A light on the disk drive indicates when the disk is being accessed. Removing or inserting a disk while the drive light is illuminated may cause damage to the disk and is likely to result in the loss of data stored on the disk.

Hard Disks

Hard disks (or fixed disks) work on the same principle as floppy disks but are fixed inside the PC in a sealed unit. They can store a great deal more information than floppy disks and range in capacity from 10Mb to several hundred Mb. Access times (*i.e.* the time taken to read and write information) for hard disks are much faster than for floppy disks. Manufacturers often quote access times as well as capacities for hard disks. Information is stored on disk in the form of *files*. A file might be a programme or data such as a word processor document. Files can be grouped together on disk in *directories*. Hard disks are contained in an air-sealed unit and are thus less liable to physical problems than floppies. However the consequences of any disk failure are much higher than for floppies so some form of backup must be carried out regularly.

CD ROM Drives

Today, many software programmes come n CD instead of floppy dosks. The programmes for newer software applications are quite karge. These large programmes, which may take as many as 20 floppy disks, can fit just fine on a

single CD, which reduces costs and makes installation much easier. To distinguish them from music CD's the computer CD's are called CD-ROMS. This stands for Compact Disk-Read Only Memory.That is, a consumer is unable to erase or record information or programmes onto a CD. If you want to advantage of recording, deleting and re-recording, then you use a a rewritable disk,(CR-RW).

Ports and Expansion Slots

Located at the rear of the PC are various sockets or ports. These allow the keyboard and monitor to be connected. Most PCs include a *parallel* port (usually used for printers) and two *serial* ports (usually used for communications with other computer systems or connecting mice and plotters).

Also included inside the system box are *expansion slots*. These allow extra hardware to be added to the PC using printed circuit boards, or *cards*, plugged into the expansion slots. The slots are connected to the CPU via the *bus*. The bus is a set of wires which transfers data. Think of it as a freeway.

Motherboards

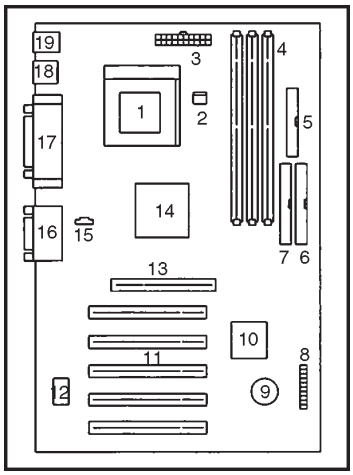
If the CPU is the brain of the computer, the motherboard and its components are the major systems this brain uses to control the rest of the computer. It is possibly the most important part of the computer. Having a good understanding of how the motherboard and its contained subsystems work is probably the most critical part of getting a good understanding of how PCs work. Here you can see the connection from the power supply to the motherboard. The power supply serves two functions. First of all it draws power

from the outlet (usually 120V AC) and translates it into voltages the computer can use (usually 12V DC). It also runs the main cooling fan which prevents the computer from overheating which can be detrimental to the integrity of your data.

The motherboard is the foundation of any PC. All the critical subsystems, including the CPU, the system chipset, the memory, the system I/O, the expansion bus, and other critical components run directly off the motherboard.

Likewise, the interconnections among these components are built into the motherboard itself. It manages all transactions of data between the CPU and the peripherals.

It houses the CPU and its second level cache, the chipset, the BIOS, the main memory, the I/O chips, the port for the keyboard, the serial I/O, the parallel I/O, the disks, and the plug-in cards.



Components of a Motherboard

- CPU Socket (for processor)
- CPU fan connector
- ATX power supply connector (for plug from power supply in computer case)

- Memory Slots (for memory modules-new mainboards use DDR modules, previous generation uses SDRAM modules)
- Floppy Drive Connector (for data cable to floppy drive)
- Primary IDE Connector (data cable for hard drive, cd-drive-up to two devices)
- Secondary IDE Connector (data cable for hard drive, cd-drive-up to two devices)
- Panel for connecting power switch, reset switch, hard drive light, etc.
- CMOS Battery (keeps current BIOS Setup information intact)
- Chipset (controls specific functions)
- PCI Slots (for plugin cards such as modems, network cards, pci video cards, sound cards, etc.)
- BIOS chip
- AGP Slot (for Accelerated Graphics Port video card, today's standard)
- Chipset (controls specific functions)
- Connector for CDROM audio cable (for sound already built onto main board)
- On-board Audio Jacks (line in, line out, microphone jacks)
- Parallel Port and Serial Ports (parallel printer port plus one or two serial ports which are hidden from view)
- USB Ports (new mainboards use USB 2.0, older use USB 1.1)
- Keyboard and Mouse Connectors (PS2 connectors, one for keyboard, one for mouse)

Many mainstream PC's today, especially lower priced ones, have the AGP Video controller, the Audio controller, Modem and NIC (Network Interface Card) built onto the mainboard rather than on separate plug-in cards. On-board video uses a portion of the memory provided by the plug-in memory modules which is typically reserved for the rest of the system.

This type of video does not perform well when used for high-end graphics such as games. It is possible that, even though the video chip is built on board, the manufacturer may have supplied an AGP slot so you can install your own Video card. Today's video cards contain at least 32mb of their own memory, and may go up to as high as 256mb. The on-board video would first need to be disabled in the BIOS/ CMOS setup routine before installing the card. If the modem, sound, or network interface fail, or you want to upgrade them, they can usually be disabled in the BIOS settings also, allowing for installation of a plug-in replacement card.

Connections

Ports

- *Parallel*: This port is commonly used to connect a printer.
- *Serial*: This port is typically used to connect an external modem.
- *Universal Serial Bus (USB)*: Quickly becoming the most popular external connection, USB ports offer power and versatility and are incredibly easy to use.
- *Firewire (IEEE 1394)*: FireWire is a very popular method of connecting digital-video devices, such as camcorders or digital recorders, to your computer.

Internet/Network

- *Modem:* This is the standard method of connecting to the Internet..
- *Local Area Network LAN Card*: This is used by many computers, particularly those in an Ethernet office network, to connected to each other.
- *Cable Modem:* Some people now use the cable-television system in their home to connect to the Internet.
- Digital Subscriber Line DSL Modem. This is a high-speed connection that works over a standard telephone line.
- *Very high bit rate DSL (VDSL)*: A newer variation of DSL, VDSL requires that your phone line have fibre optic cables.

The Main Unit

The main unit contains the Central Processing Unit (CPU) and various supporting Integrated Circuits (or *chips*) all of which are fixed to a printed circuit board (PCB) called the *motherboard*.

The main unit also houses disk drives, expansion slots and the power supply.

The CPU

The CPU (Central Processing Unit) or processor is the 'brain' of any computer system. In the PC it is contained on a single Integrated Circuit or 'chip'. The CPU processes all instructions and data. The CPU is driven by an internal clock. Simply speaking, every time the clock pulses the CPU processes one instruction. Thus, the faster the clock the quicker the CPU

processes its instructions. Clock speed is measured in MegaHertz (MHz). All the real computing work is done by the CPU.

Memory

All PCs are fitted with a certain amount of workspace memory called *Random Access Memory* (RAM).

This memory is used for storing running software and data which the software requires, as well as the*operating system*. The contents of RAM are lost when the PC is switched off.

This in known as volitile because it is tempory in nature.For example, if you are running word processing software then the software and the document you are working on are stored in RAM. To save the document, after making any changes you wish to keep, it is necessary to copy the document from RAM onto *disk*.

RAM

Chips can be individually installed directly on th motherboard or in the banks of several chips on a small board that plugs into the motherboard.

The most common types of boards that hold memory chips are called SIMMS (single inline memory modules), DIMMS (duel inline memory modules), andRIMMS (memory modules manufctured by Rambus, Inc.)

By contrast, another kind of memory holds its data permanently, even when the power is turned off. This type of memory is called non-volitile and is called ROM (read only memory).

ROM (Read-only Memory)

They contains the commands the computer needs to activate itself. Instructions in ROM let the computer start when the power is turned on, and, unlike RAM, its contents are retained even when the power is off. A Sound Card is an expansion card that lets a computer produce sound. Examples of practical uses for sound capabilities include games, music applications, and interactive educational software.

The Expansion Card is a circuit board that slides into an expansion slot. It is used to add peripherals, such as a sound card or modem, to the PC. An Expansion Slot is an opening on the motherboard into which a board or card can be inserting, expanding the capability of the computer. The Power Supply is the vehicle through which electricity is regulated and sent to the various components of a computer.

BIOS or CMOS setup

The motherboard contains a special chip called the BIOS, which is short for Basic Input/Output System. Burnt onto this chip are software instructions on how to load basic computer hardware. Many PCs have what is called a flash BIOS, allowing the user to update the BIOS if necessary. The BIOS also includes a sort of diagnostic routine known as the POST (Power On Self Test). This test ensures that the computer meets the necessary requirements to boot up correctly. If the computer doesn't pass the POST you will hear a pattern of beeps that indicate the problem encountered. CMOS (Complementary Metal Oxide Semiconductor) is a chip on the motherboard which stores

system information and computer settings such as date, time, hard drive settings, boot sequence, parallel port settings, on-board audio and video, etc. This information can be accessed and changed through the BIOS/CMOS setup programme which is available as the computer begins to boot up. As the computer boots, there will typically be some text on the screen such as "Press Del to enter Setup". Depending on the manufacturer, the key required to enter the BIOS setup may be F1, F2, Del, or Esc.

Unlike earlier generations of PC's, the user is no longer required to go into the BIOS Setup and enter new information such as the number of cylinders, heads, sectors, etc. when changing IDE hard drives, for instance. These and some other settings are now detected automatically. If the CMOS battery dies, any changes made in the the BIOS Setup will be lost. After replacing the battery, the user will need to re-enter the Setup programme and make the changes again.

Driver

After the computer boots up, the operating system begins the process of loading into memory. Unlike the information burnt into the BIOS and CMOS chips, Windows must detect the hardware and load device specific files into memory that allow the operating system to understand and communicate with the hardware devices. As part of the Windows operating system, Microsoft includes a great many device drivers (for the more popular devices and peripherals). These load automatically as the hardware is detected. However, you may have a particular modem or sound card that isn't included in Windows database. While the operating system may detect the device's presence, it can't assign a driver file to it, since

their isn't one in it's database. This is where you come in. You will need to locate the correct driver for the version of Windows you are using. If you have a disk that came with the device you'll need to use it to install the driver, provided it was created for your version of Windows. A driver written for Windows '98 won't work with XP, so you'll have to get on the internet and find a current one. A good site to look for the driver is DriverGuide.com, but you'll need to know something about the device such as the manufacturer, model number, etc. to get the proper driver.

3

Development of Computer Networks

A computer network is two or more computers connected together using a telecommunication system for the purpose of communication and sharing resources". Ask any computer network expert to simplify this definition to you and you will start a debate on how it should not be just two computers but three. Simply put a network is a means of communication between computers. Within a given network, computers can send files, e-mails and other correspondence to each other. Even things like instant messaging, is set up within a computer's network. There are two basic types of computer Networks. Lan and Wan.

LAN

LAN or Local Area Network is the most common kind of network set up. There are two ways to connect a LAN network.

The simplest and easiest way is the peer-to-peer connection network. This is when two or more computers are directly connected to each other. For example if there were four computers in the network, computer 1 would be connected to computer 2, computer 2 would be connected to computer 3 and computer 3 would be connected to computer 4.

This means each computer is dependent on the other. And if there were a network problem with any one computer, all of them would be affected. The other type if the client server connection. This is the type of connection where all the computers in a given network are connected to one central computer. This is a more complicated network but one that is much more efficient that peer-to-peer.

A local area network, or LAN, is a network of connected computers in a room, building, or set of buildings. Local area networks have been around since the beginning of computer use. A LAN is defined as a user network whereby data is sent at high rates between people located relatively close to each other. LANs do not usually make use of leased communication lines, but only means of communication that are provided by the installer of the network.

The Internet is a wide area network, or WAN, which is distinct from a LAN. In contrast to the term *Internet*, local area networks are often called *intranets*, though sometimes this term refers to a cluster of LANs associated with a particular company or organization but not connected to the larger Internet. A local area network uses a hub or router to connect computers together.

The means of communication is the omnipresent Ethernet cable or wireless wi-fi technology. These technologies offer

data transfer rates running between 10 to 10000 Mbit/s. Larger, more important LANs have redundant lines or other backup protocols. In networked computers, the most popular communication protocol is TCP/IP. Smaller LANs may be temporary and used between friends to play computer games over the network.

Over a network, users can share files, view files, make changes to data on other computers if permitted, play movies or music on multiple computers at once, chat with instant messaging, send e-mails to each other, play games, and so on. All the advantages of the Internet apply, although they only include others on the LAN, and the data transfer rates are high.

Perhaps the most frequently employed use of a LAN is to connect users to the Internet with only one connected router. In modern times, we use broadband cable or DSL modems to connect to the Internet, and it would be clumsy to have a modem associated with every computer, so we simply plug the modem into a router and link the router to computers with Ethernet cables.

Configuring a LAN can be intimidating at first, but contemporary operating systems have programmes that do most of the necessary configurations automatically, so setting up a local area network is pretty easy. A local area network (LAN) supplies networking capability to a group of computers in close proximity to each other such as in an office building, a school, or a home. A LAN is useful for sharing resources like files, printers, games or other applications. A LAN in turn often connects to other LANs, and to the Internet or other WAN.

Most local area networks are built with relatively inexpensive hardware such as Ethernet cables, network adapters, and hubs. Wireless LAN and other more advanced LAN hardware options also exist.

Examples

The most common type of local area network is an Ethernet LAN. The smallest home LAN can have exactly two computers; a large LAN can accommodate many thousands of computers. Many LANs are divided into logical groups called subnets. An Internet Protocol (IP) "Class A" LAN can in theory accommodate more than 16 million devices organized into subnets.

High Performance Network

The HPN is used for the exchange of large amounts of operational data. Two Force10 E600 Routers, interconnected via 4-way 10-Gigabit Ethernet aggregated links, provide connectivity between the High Performance Computing Facility (HPCF) and the Data-Handling System (DHS). The HPCF network nodes are connected via 10-Gigabit Ethernet and all DHS nodes via Gigabit Ethernet aggregated links.

General Purpose Network

The GPN is used for all other traffic. It has at its core two Foundry BigIron RX-16 routers and at the edge seven Foundry Super-X switches. The core routers are interconnected via 4-way 10-Gigabit Ethernet aggregated links and have multiple Gigabit Ethernet uplinks to the edge routers. The core also includes two further Super-X switches that are dual-attached to the RX-16s via 10-Gigabit Ethernet.

The GPN provides connectivity to:

- The HPCF, the DHS and additional servers via Gigabit Ethernet ports in the core.
- The user desktops and laptops via Gigabit Ethernet ports on the edge switches.
- The firewalls (for the Wide Area Network and the Demilitarized Zone (DMZ) via Gigabit Ethernet ports in the core. The DMZ includes ECaccess, web servers, the mail gateway and DNS (Domain Name Servers).

The Hardware

Both the Force10 E600 chassis are populated with 24 10-Gigabit-Ethernet and 144 Gigabit Ethernet ports. For resiliency there are four power supplies, two CPU modules and nine switching fabric modules. Both the RX-16 chassis are populated with 8 10-Gigabit Ethernet ports and 144 Gigabit Ethernet ports. For resiliency there are seven power supplies, two CPU modules and four switching fabric modules. The Super-X chassis each contain up to 156 Gigabit Ethernet ports and dual power-supplies.

WAN

Definition

The wide area network, often referred to as a WAN, is a communications network that makes use of existing technology to connect local computer networks into a larger working network that may cover both national and international locations. This is in contrast to both the local area network and the metropolitan area network, which provides communication within a restricted geographic area. Here is how the wide area network functions, and why it is

so important to communications today. The concept of linking one computer network with another is often desirable, especially for businesses that operate a number of facilities. Beginning with the local area network and going up to the wide area network, this is most easily accomplished by using existing telephony technology. Essentially, fibre optics are used to create the link between networks located in different facilities.

Often, this means using standard phone lines, referred to as POTS, or employing PSTN (public switched telephone network) technology. During the 1990s, a third option, that of ISDN (integrated services digital network) solutions for creation a wide area network gained a great deal of popularity, mainly because the concept made it more cost effective to extend the network beyond national boundaries.

With coverage in a broad area, a wide area network allows companies to make use of common resources in order to operate. For example, many retail drugstores make use of a wide area network as part of their support to customers who fill prescriptions with one of their stores. Once in the common customer database for the pharmacy, the client is free to fill a prescription at any of the company's locations, even while vacationing in another state.

Companies also make good use of the wide area network as well. Internal functions such as sales, production and development, marketing and accounting can also be shared with authorized locations through this sort of broad area network application. The concept of a wide area network as a means of taking individual location based computer networks and using them to create a unified computer

network for the entire corporation means that employees can work from just about anywhere. Should one facility be damaged or rendered inaccessible due to natural disaster, employees simply move to another location where they can access the unified network, and keep on working.

Overview

WAN or Wide Area Network is when several LANs or independent computers are connected to a single, wider network. The Internet is the perfect example of WAN. E-mails, Chat Rooms and IMs all connect to the WAN of the Internet. WAN is much more complex and requires connecting devices or hubs from all over the world. The term Wide Area Network (WAN) usually refers to a network which covers a large geographical area, and use communications circuits to connect the intermediate nodes.

A major factor impacting WAN design and performance is a requirement that they lease communications circuits from telephone companies or other communications carriers. Transmission rates are typically 2 Mbps, 34 Mbps, 45 Mbps, 155 Mbps, 625 Mbps (or sometimes considerably more). Numerous WANs have been constructed, including public packet networks, large corporate networks, military networks, banking networks, stock brokerage networks, and airline reservation networks.

Some WANs are very extensive, spanning the globe, but most do not provide true global coverage. Organizations supporting WANs using the Internet Protocol are known as Network Service Providers (NSPs). These form the core of the Internet. Wide Area Networks, or WANs, connect a

geographically diverse group of computers within a state, country, or even across several states or countries. WANs typically are connected by telephone lines, other types of communication lines, or radio waves.

Quite often, smaller local area networks (LANs) are linked together to form a WAN. This is accomplished via dedicated private lines, leased from telecommunications firms like Sprint and ATandT, or by Switched Multi-Megabit Data Services (SMDS) technology, developed in 1995 to eliminate the need for a leased line. WAN technology has been refined over a period of several decades. It first emerged in the mid-twentieth century with the advent of networks like ARPAnet. Developed in 1969 by the Department of Defence, ARPAnet and several other networks eventually evolved into the Internet, the largest WAN in the world.

The packet switching technology most commonly used with WANs surfaced in the 1960s, and standard packet switching protocol, known as X.25, was developed in 1976. To increase network speed, packet switching allows for the parceling of data into smaller chunks, known as packets, prior to transmission. These packets can travel independently via alternate routes, and they are reassembled once they reach their target.

Although X.25 remained the most popular WAN packet switching protocol for years, other packet switching protocols used with increasing frequency by WAN developers and administrators include the Internet standard, Transmission Control Protocol/Internet Protocol (TCP/IP), and Frame Relay, used most often by WANs connected via high speed T-1 and T-3 lines.

WANs are used for a variety of purposes. A corporation with offices in several locations may use a WAN to form an intranet. Quite often, the individual offices will use their own LANs for things like internal messaging, data processing functions, and hardware and software sharing. When these LANs are joined together to form a WAN, similar data sharing and messaging capabilities become possible across a much broader geographic area.

Businesses wanting to link up with their suppliers or distributors may create a WAN as a means of establishing an extranet. For example, an extranet could provide a sales representative with electronic access to information in about the time it might take to deliver a product, or the availability of a product.

Some WANs bring together various types of communications, such as data, video, and voice. Some organizations, including companies, universities, research centres, hospitals, and libraries, use WANs to connect to the Internet. By connecting the NSP WANs together using links at Internet Packet Interchanges (sometimes called "peering points") a global communication infrastructure is formed.

NSPs do not generally handle individual customer accounts (except for the major corporate customers), but instead deal with intermediate organizations whom they can charge for high capacity communications.

They generally have an agreement to exchange certain volumes of data at a certain "quality of service" with other NSPs.

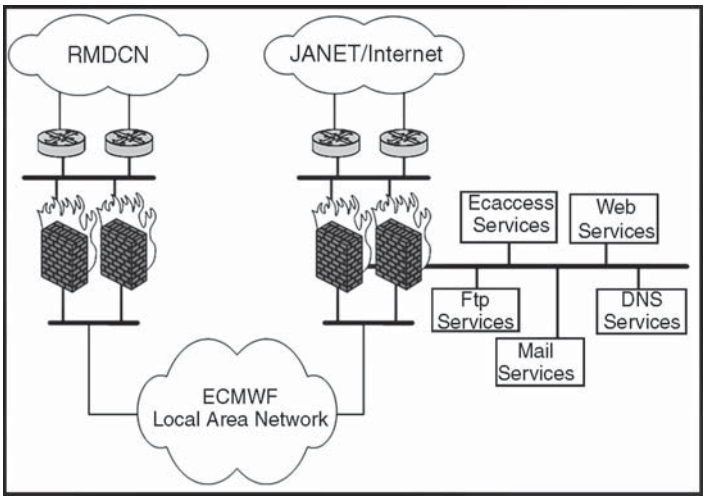
So practically any NSP can reach any other NSP, but may require the use of one or more other NSP networks to reach

the required destination. NSPs vary in terms of the transit delay, transmission rate, and connectivity offered.

The characteristics of the transmission facilities lead to an emphasis on efficiency of communications techniques in the design of WANs.

Controlling the volume of traffic and avoiding excessive delays is important. Since the topologies of WANs are likely to be more complex than those of LANs, routing algorithms also receive more emphasis.

Many WANs also implement sophisticated monitoring procedures to account for which users consume the network resources. This is, in some cases, used to generate billing information to charge individual users.



The size of a network is limited due to size and distance constraints. However networks may be connected over a high-speed communications link (called a WAN link) to link them together and thus becomes a WAN. WAN links are usually:

- Dial up connection
- Dedicated connection-It is a permanent full time connection. When a dedicated connection is used,

the cable is leased rather than a part of the cable bandwidth and the user has exclusive use.

- Switched network-Several users share the same line or the bandwidth of the line.

  There are two types of switched networks:

  1. *Circuit switching*: This is a temporary connection between two points such as dial-up or ISDN.
  2. *Packet switching*: This is a connection between multiple points. It breaks data down into small packets to be sent across the network. A virtual circuit can improve performance by establishing a set path for data transmission. This will shave some overhead of a packet switching network. A variant of packet switching is called cell-switching where the data is broken into small cells with a fixed length.

Connection Technologies

- X.25-This is a set of protocols developed by the CCITT/ITU which specifies how to connect computer devices over an internet work. These protocols use a great deal of error checking for use over unreliable telephone lines. They establish a virtual communication circuit. It uses a store and forward method which can cause about a half second delay in data reception when two way communications are used. Their speed is about 64Kbps. Normally X.25 is used on packed switching PDNs (Public Data Networks). A line must be leased from the LAN to a PDN to connect to an X.25 network. A PAD (packet

assembler/disassembler) or an X.25 interface is used on a computer to connect to the X.25 network. CCITT is an abbreviation for International Telegraph and Telephone Consultative Committee. The ITU is the International Telecommunication Union.

- Frame Relay-devices at both sides of the connection handle Error checking. Frame relay uses frames of varying length and it operates at the data link layer of the OSI model. A permanent virtual circuit (PVC) is established between two points on the network. Frame relay speed is between 56Kbps and 1.544Mbps. Frame relay networks provide a high-speed connection up to 1.544Mbps using variable-length packet switching over digital fibre-optic media. Frame relay does not store data and has less error checking than X.25.
- Switched Multi-megabit Data Service (SMDS)-Uses fixed length cell switching and runs at speeds of 1.533 to 45Mbps. It provides no error checking and assumes devices at both ends provide error checking.
- Telephone connections
  - Dial up
  - *Leased lines*: These are dedicated analog lines or digital lines. Dedicated digital lines are called digital data service (DDS) lines. A modem is used to connect to analog lines, and a Channel Service Unit/Data Service Unit or Digital Service Unit(CSU/DSU) is used to connect to digital lines. The DSU connects to the LAN and the CSU connects to the line.

- *T Carrier lines*: Multiplexors are used to allow several channels on one line. The T1 line is basic T Carrier service. The available channels may be used separately for data or voice transmissions or they may be combined for more transmission bandwidth.
- T1 and T3 lines are the most common lines in use today. T1 and T2 lines can use standard copper wire. T3 and T4 lines require fibre-optic cable or other high-speed media. These lines may be leased partially called fractional T1 or fractional T3, which means a customer, can lease a certain number of channels on the line. A CSU/DSU and a bridge or router is required to connect to a T1 line.
- Integrated Services Digital Network (ISDN)-Comes in two types and converts analog signals to digital for transmission. It is a dial up service

  a. Basic Rate ISDN (BRI)-Two 64Kbps B-channels with one 16Kbps D channel. The D-channel is used tor call control and setup.

  b. Primary Rate ISDN (PRI)-23 B-channels and one D channel. A device resembling a modem (called an ISDN modem) is used to connect to ISDN. The computer and telephone line are plugged into it.
- *Switched-56*: A switched line similar to a leased line where customers pay for the time they use the line. Speed is 56Kbps. It is not dedicated and will not work to connect a WAN.

- *Asynchronous Transfer Mode (ATM)*: May be used over a variety of media with both baseband and broadband systems. It is used for audio, video, and data. It uses fixed length data packets of 53 8 bit bytes called cell switching. 5 bytes contain header information. The cell contains path information that the packet is to use. It uses hardware devices to perform the switching of the data. Speeds from 155Mbps to 622 Mbps are achieved. Error checking is done at the receiving device, not by ATM. A permanent virtual connection or circuit (PVC) is established. It may also use a switched virtual circuit (SVC). *Service classes*:
  - Constant bit rate for data.
  - Variable bit rate for audio or video.
  - Connection less for data.
  - Connection oriented for data.

  ATM can be embedded in other protocols such as ATM-25, T1, T3, OC-1, OC-3, OC-12, and OC-48.

  Some ATM technologies include:
  - ATM-25-25Mbps speed.
  - STS-3-155Mbps on fibre or category 5 cable.
  - STS-12-620 Mbps on fibre cable for campus wide network.
  - STS-48-2.2 Gbps on fibre cable on a MAN.
  - STS-192-8.8 Gbps on fibre cable on intercity long distance. Phone companies normally use this.

Synchronous Optical Network (SONET)-A physical layer standard that defines voice, data, and video delivery methods over fibre optic media. It defines data rates in terms of optical

carrier (OC) levels. The transmission rate of OC-1 is 51.8 Mbps. Each level runs at a multiple of the first. The OC-5 data rate is 5 times 51.8 Mbps which is 259 Mbps.

SONET also defines synchronous transport signals (STS) for copper media which use the same speed scale of OC levels. STS-3 runs at the same speed of OC-3. Mesh or ring topology is used to support SONET. SONET uses multiplexing. The ITU has incorporated SONET into their Synchronous Digital Hierarchy (SDH) recommendations.

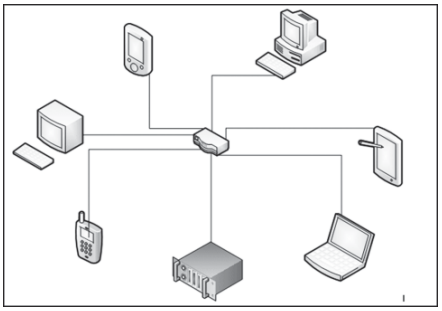
WAN Technology Comparisons

Terms

- *Circuit switching*: Physical switched connection.
- *Message switching*: A store and forward mechanism where messages are treated as individual units.
- *Packet switching*: Messages are broken down into smaller packets with individual destination information. Independent routing is used which allows the packets to use any route between the source send destination. Much RAM and processing power is required to support this switching type.
- *Data gram packet switching*: Uses independent paths.
- *Virtual circuit packet switching*: This is used for audio and video streaming. A set path is established between the source and destination and a connection-oriented service is made.

Network Distribution

Building a network consists partly of connecting the computers:

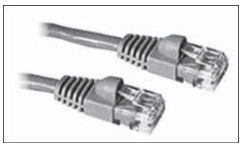


Besides the computers, you will use other objects.

Network Cables

Cable is used to connect computers. Although we may use wireless networking, you should always have cables with you. The most commonly used cable is referred to as Category 5 cable RJ-45.

The ends of the cable appear as follows:



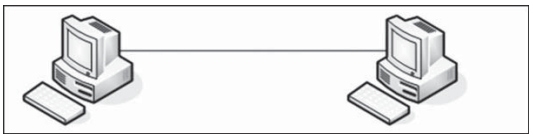
They can be in different colours:

You can purchase this cable from a general store, a computer store, or web store on the Internet. When purchasing it, get one with at least 6ft.

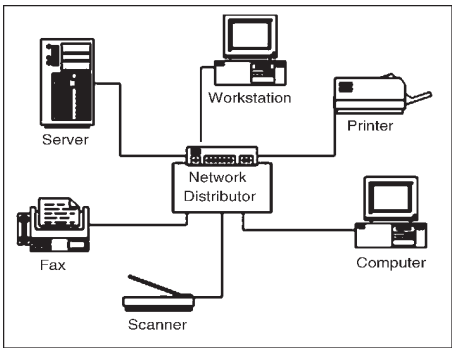
Introduction to Network Distributors

We mentioned that you could connect one computer to another.

This can be done using their serial ports:



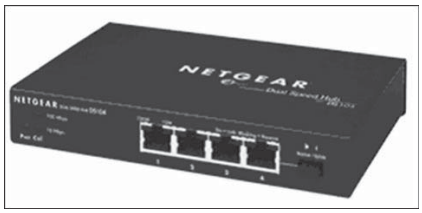
This is possible because almost every computer has a serial port. If you have to connect many computers to produce a network, this serial connection would not be practical. The solution is to use a central object that the computers and other resources can connect to, and then this object becomes responsible to "distribute" or manage network traffic:



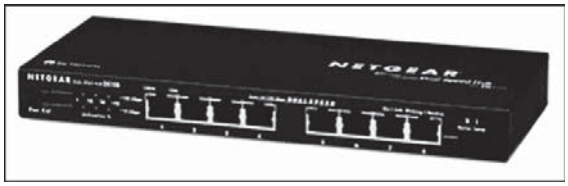
The most regularly used types of network distributors are the hub, the router, and the switch.

Hub

A hub is rectangular box that is used as the central object on which computers and other devices are connected. To make this possible, a hub is equipped with small holes called ports. Here is an example of a hub:



Although this appears with 4 ports, depending on its type, a hub can be equipped with 4, 5, 12, or more ports. Here is an example of a hub with 8 ports:



When configuring it, you connect an RJ-45 cable from the network card of a computer to one port of the hub. In most cases for a home-based or a small business network, you may not need (or shouldn't use) a hub.

Routers: Wired or Wireless

Like a hub, a router is another type of device that acts as the central point among computers and other devices that are part of a network. Here is an example of a wired router:

A router functions a little differently than a hub. In fact, a router can be considered a little "intelligent" than the hub. Like a hub, the computers and other devices are connected to a router using network cables. To make this possible, a router is equipped with holes, called ports, in the back.

Here is an example:

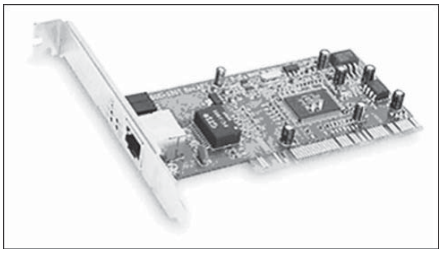
Based on advances in the previous years from IEEE and other organizations or research companies, there are wireless routers. With this type, the computers and devices connect to the router using microwaves (no physical cable).

Wired Network Cards: Internal

In order to connect to a network, a computer must be equipped with a device called a network card. A network card, or a network adapter, also called a network interface card, or NIC, allows a computer to connect to the exterior. If you buy a computer from one of those popular stores or big companies on the Internet, most of their computers have a network card tested and ready. You can reliably use it. If you go to a store that sells or manufactures computers, you can ask them to install or make sure that the computer has a network card. If you have a computer that doesn't have a network card, you can install one. If you have a computer that already has a network card, you can still replace it.

When it comes to their installation, there are roughly two categories of network cards: internal and external.

An internal network card looks like a printed circuit board with some objects "attached" or "glued" to it and it appears as follows:



What this card looks like may not be particularly important and it may depend on the manufacturer but some of its

aspects are particularly important. To start, there are two types of cards and you should know which one is suited (or which one you want to use) for your computer. One type of NICs uses a peripheral component interconnect (PCI) connection. Another type uses industry standard architecture (ISA).

There are two primary ways you replace an internal network card. In most cases, you will remove the card your computer already has and install a new one. In some other cases, you will only add a new card but you cannot replace the existing one because it is part of the motherboard. The area where you add a network card is called a slot. To proceed, you must find out what your computer has to offer when it comes to network cards. To do this, you have three main alternatives. You can open the computer and examine the available slots of your computer. They are usually located inside of what would be considered as the back wall of the computer. If you know where you connect the monitor, you should be able to locate the area that has the slots. Unfortunately, unless you have experience with this, simply looking at the slots will not tell you what type of connection you are dealing with.

The second alternative is to open the manual that came with your computer (provided you haven't thrown it away). The manual usually lists the (types of) slots that your computer provides and where they are located. The last alternative to knowing the types of slots that your computer provides is to contact the company that sold you the computer. They usually know, as long as you give them the model of the computer.

Once you know the type of slot available to you, you can go on the Internet or to a computer store and buy an appropriate network card. One of the most important characteristics of a network card is the speed it can use to carry information (data). The speeds are either 10 or 100Mbps (megabits per second). When buying a network card, you should pay attention to this. Your computer manufacturer also may sell network cards intended for your computer.

After buying a network card intended for internal installation, you can/must install it. The network card should come with a manual and all (easy to follow) instructions. You can also install the network card after setting up the computer.

Wired Network Cards: External

We mentioned that a network card could also be used or installed externally. This can be done using USB. Before using it, you can purchase it from a computer store or a web store.

The device may look like this:

Here is another example:

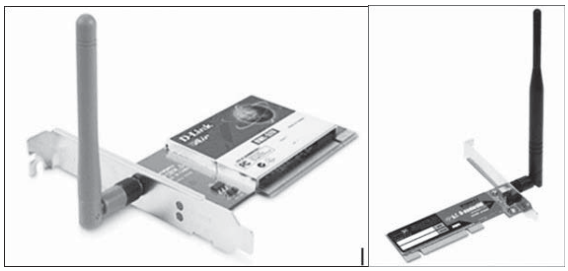
If you buy one of these objects, its documentation will guide you.

Wireless Network Cards

Depending on your network budget or your customer's, instead of using wired network cards, you can use wireless ones. Most laptops already have a wireless card built-in so you may not have to acquire one. Many new desktop computers (from HP) now have built-in wireless capability.

A wireless NIC appears as its wired counterpart.

Here are two examples:



Overall, the physical installation of a wireless network card follows the same rules as that of a wired NIC. They normally come with easy to follow instructions but it may be a good idea to install the wireless network adapters after installing the wireless router. Also, it may be a good idea to purchase the network cards and the wireless router from the same manufacturer.

Most desktop computers (workstations) come without a wireless network card. If you purchase a computer from one of the big companies on the Internet, you can choose to have it shipped with a wireless NIC. Some companies may propose to install it before shipping the computer. If you buy a computer from a store and if you want to use wireless

networking, you can buy a wireless network card separately. As stated already, a wireless network card is not particularly difficult to install.

Besides the wireless network cards that can be installed inside the computer, you can use external cards. These are installed using a USB port. Here is an example of a USB adapter:

Here is another example:

These adapters, like most USB objects, are easy to connect and use. Like the other hardware parts, when you connect these, the computer detects them and helps you get them ready for use. Unlike desktop computers, most laptops nowadays come equipped with a wireless network card (in fact most laptops today ship with both a wired and a wireless adapters).

This means that, after purchasing or acquiring a laptop, you should simply check whether it has a wireless adapter. The way you check this depends on the laptop. Therefore, check its documentation.

Network Accessories

Printers

If you attach a printer to one computer and share it, when that computer is off, nobody can print. An alternative is to purchase a network printer. That is, a printer that will directly connect to the network and people can print to it any time.

There are many types of printers in this case:

- Some printers come equipped with a network card. In this case, you can use an RJ-45 cable to connect it to a router or a hub
- Most printers nowadays have a USB port that can be used to connect them to a router
- Many printers come equipped with wireless capability. This means that the computers can connect to the printer without using a wire and they can print.

If you are using a wireless network and your printer doesn't have wireless capabilties, you can purchase a wireless print server. This allows you to connect almost any type of printer, with or without a network card, to the network. You can purchase a wireless print server from a computer store or from a web store. It is usually easy to install as it comes with easy-to-follow instructions.

Internet Service Provider (ISP)

An Internet Service Provider (ISP) is a company that serves as the intermediary between your network (or you) and the Internet. If you plan to give access to the Internet to the members of your network, you may need this type of company. You can start by checking with your local telephone company or your local TV cable company.

Firewall

Firewall is a security measure that consists of protecting your network from intruders. This is primarily important if you plan to connect your network to the Internet. There are two types of firewalls: hardware and software. For a small network, when buying a router, you can inquire as to whether it has a built-in firewall. Many of them do. Alternatively, you can use or configure one of the computers of your network as a firewall.

Networking Topologies

Definition

Network topology refers to the way that your computer network is arranged. The network can have a physical or a logical topology. The physical topology describes the layout of computers and where the workstations are positioned. The logical network topology describes how the information flows through the network.

Choosing your physical topology is important because if it is not chosen correctly, this could cause your network to not operate properly. There are several terms that describe the type of physical topology that a network can have. The most common topologies are bus, ring, star, and mesh. In a bus topology, all of the computers are attached to a single cable using terminators.

The terminators work to absorb the energy from the signals in the network. The bus topology is easy to install, but it is not reliable because a single default can bring down your network.

In a ring topology, each computer is connected directly to two other computers on the network. As with a bus topology, a single fault can disrupt a ring network. This type of network does have advantages, however, and does not require a network server.

In a star topology, each computer is connected using its own separate cable. This set up is more reliable than a bus topology because its design makes it fault tolerant and susceptible to errors. The information on the network is transmitted from one system to another and the data flows in one single direction. The topology is expensive to maintain and is not reliable because removing one computer can disrupt your entire network.

In a mesh topology, a path is present from one computer to another computer in the network. The mesh topology is usually used in internet structure. The mesh topology can be complicated to construct because it has multiple connection between locations. For every computer that you have you will need at least one and a half connections for each one. This contributes to the expensive structure.

In choosing a network topology, understand that each one has its advantages and disadvantages. Research each one and see which one fits within your budget and which one fits your network layout. Also consider the amount of maintenance required in each topology. When setting up a network at home, consider a network topology that is simple and easy to maintain. You will also want a topology that is inexpensive to set up. If you are choosing a topology for a business, you will want to consider a topology that is reliable and resistant to errors. Your customers will need a network

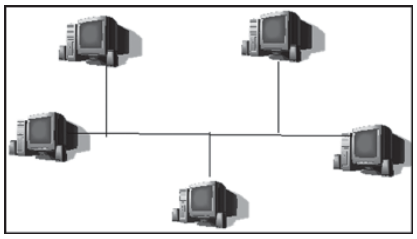
that is free of downtime; therefore, be sure that your topology is one that withstands disruption.

Types of Networking Topologies

Bus Topology

A bus topology is a method of transmission on networks that uses a common vehicle for transmissions and thus it is categorized as shared communication. Imagine a bus picking up various people from one stop and dropping of people as it travels and then picking a few more.

That is what happens in a bus network exactly. A bus network uses a multi-drop transmission medium, all node on the network share a common bus and thus share communication. This allows only one device to transmit at a time. A distributed access protocol determines which station is to transmit. Data frames contain source and destination addresses, where each station monitors the bus and copies frames addressed to itself.



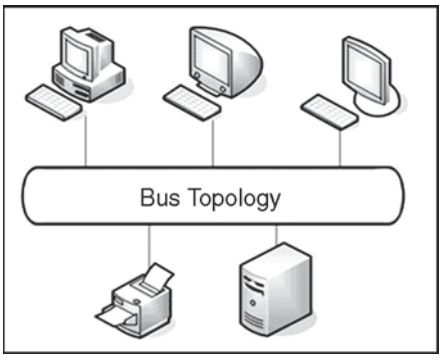
A bus topology connects each computer (nodes) to a single segment trunk (a communication line, typically coax cable, that is referred to as the 'bus'. The signal travels from one end of the bus to the other. A terminator is required at each to absorb the signal so as it does not reflect back across the bus. A media access method called CSMA/MA is used to handle the collision that occur when two signals placed on

the wire at the same time. The bus topology is passive. In other words, the computers on the bus simply 'listen' for a signal; they are not responsible for moving the signal along.

However in a Bus topology only one device is allowed to transmit at a given point of time. The DAP or the Distribute Access Protocol has the information about which station has to transmit the data. The data that is being transmitted have frames that will have the source name and the network address.

Function of Bus Topology

The bus topology connects each computer on the network into something called the segment trunk. A bus is usually referred to a cable that connects end to end and this is used to transmit the signals from one end to the other end. At every end a terminator is placed so that it understands in which direction the data is traveling and also the terminator is used to absorb the signals. If the terminator does not absorb the signal then the same signal is reflected back to the bus confusing the whole data flow. The bus is considered to be a passive network because the computers are largely dependant on the signal that is being transmitted.



Advantages

The advantage of the Bus network is that if one computer fails in the network the others are still not affected and they continue to work. Bus network is very simple and easy to set up. If there is an urgent need to set up a network and perhaps be used on a short term basis then the Bus network is the best possibility. Bus networks use the least amount of cable to set up making it cost effective.

Limitations

In the bus network you will need a network connection in order to determine if the data is being transferred between two nodes. If the data transfer rate is high then the Bus network does not perform very well because the data travels in a stream and cannot be overloaded. The bus network is a bit challenging when you have to troubleshoot the problems. Bus networks are not suitable as large networks as there are limitations to the number of nodes you can set up for one single cable. As the number of computers increase in the network the data transfer rate goes down accordingly and noticeably.

Conclusion

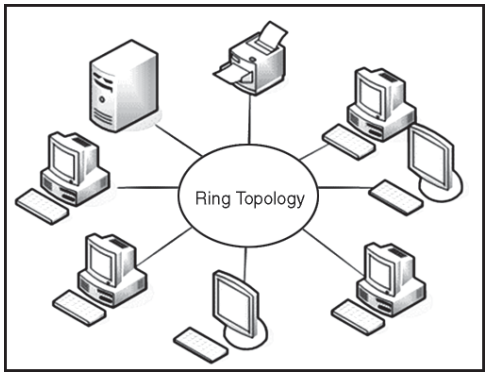
The bus networks in spite of its limitations is considered to be the easiest and the fastest network that can be set up compared to the other kinds of network. In bus networks there is a collision handling system which ensures that data travels without errors and data is delivered correctly. There is a bus master on the network which ensures that data is flowing in the right direction and order. All the computers

on the bus network however listen for the signals and they do not hold the responsibility to move the signal forward.

The signal carries forward on its own. So if one computer is not receiving any signals the signal still carries forward without stopping at the computer that has failed. These days bus networks are less common due to the advancement of networks and there are much lesser complicated networks that are easy to operate and efficient. However even these newer technologies derive their basics from older technologies like the Bus Topology.

Ring Topology

Definition

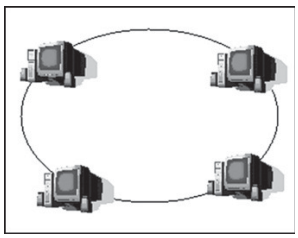


A ring topology is a network topology or circuit arrangement in which each network device is attached along the same signal path to two other devices, forming a path in the shape of a ring. Each device in the network that is also referred to as node handles every message that flows through the ring. Each node in the ring has a unique address. Since in a ring topology there is only one pathway between any two nodes, ring networks are generally disrupted by the failure of a single link.

The redundant topologies are used to eliminate network downtime caused by a single point of failure. All networks need redundancy for enhanced reliability. Network reliability is achieved through reliable equipment and network designs that are tolerant to failures and faults. The FDDI networks overcome the disruption in the network by sending data on a clockwise and a counterclockwise ring. In case there is a break in data flow,the data is wrapped back onto the complementary ring before it reaches the end of the cable thereby maintaining a path to every node within the complementary ring.

Ring network connects computers in a circle of point-to-point connections, with no central server, such as a series of desktop computers in an office.

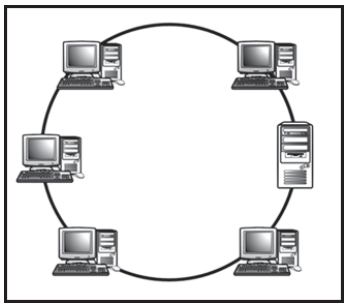
Each node handles its own applications and also shares resources over the entire network. If one node becomes inoperative, the other nodes are still able to maintain contact with one another. Such a network is best for decentralized systems in which no priorities are required



Under the network, a signal is transferred sequentially via a “token” fro one station to the next. When a station wants to transmit, it “grabs” the token, attaches data and an address to it, and then sends it around the ring.

The token travels along the ring until it reaches the destination address. The receiving computer acknowledges receipt with a return message to the sender. The sender then

releases the token for the token for use by another computer. Each station on the ring has equal access but only one station can talk at a time. To allow an orderly access to the ring, a single electronic token passes from one computer to the next around the ring as seen in (token ring). A computer can only transmit dat



In a true ring topology, if a single computer or section of cable fails, there is an interruption in the signal. The entire network becomes inaccessible. Network disruption can also occur when computers are added or removed from the network, making it an impractical network design in environments where there is constant change to the network.

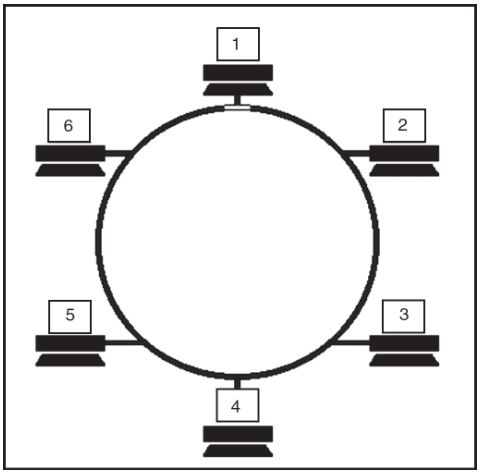
Ring networks are most commonly wired in a star configuration. In a Token Ring network, a multistation access unit (MSAU) is equivalent to a hub or switch on an Ethernet network. The MSAU performs the token circulation internally. To create the complete ring, the ring in (RI) port on each MSAU is connected to the ring out (RO) port on another MSAU. The last MSAU in the ring is then connected to the first, to complete the ring.

Token Ring Topology

Unlike Ethernet, Token Ring uses a ring topology whereby the data is sent from one machine to the next and so on around the ring until it ends up back where it started. It also

uses a token passing protocol which means that a machine can only use the network when it has control of the Token, this ensures that there are no collisions because only one machine can use the network at any given time.

Figure below shows the basic of a Token Ring Topology. Although 16Mbps is the standard ring speed these days (and Fast Token Ring is being developed) we will consider a 4Mbps Token Ring in this tutorial to explain the basic concepts.



At the start, a free Token is circulating on the ring, this is a data frame which to all intents and purposes is an empty vessel for transporting data. To use the network, a machine first has to capture the free Token and replace the data with its own message. In the example above, machine 1 wants to send some data to machine 4, so it first has to capture the free Token. It then writes its data and the recipient's address onto the Token.

The packet of data is then sent to machine 2 who reads the address, realises it is not its own, so passes it on to machine 3. Machine 3 does the same and passes the Token on to machine 4. This time it is the correct address and so number 4 reads the message. It cannot, however, release a

free Token on to the ring, it must first send the message back to number 1 with an acknowledgement to say that it has received the data (represented by the purple flashing screen).

The receipt is then sent to machine 5 who checks the address, realises that it is not its own and so forwards it on to the next machine in the ring, number 6. Machine 6 does the same and forwards the data to number 1, who sent the original message. Machine 1 recognizes the address, reads the acknowledgement from number 4 (represented by the purple flashing screen) and then releases the free Token back on to the ring ready for the next machine to use.

Token Ring Self Maintenance

When a Token Ring network starts up, the machines all take part in a negotiation to decide who will control the ring, or become the 'Active Monitor' to give it its proper title. This is won by the machine with the highest MAC address who is participating in the contention procedure, and all other machines become 'Standby Monitors'.

The job of the Active Monitor is to make sure that none of the machines are causing problems on the network, and to re-establish the ring after a break or an error has occurred. The Active Monitor performs Ring Polling every seven seconds and ring purges when there appears to be a problem.

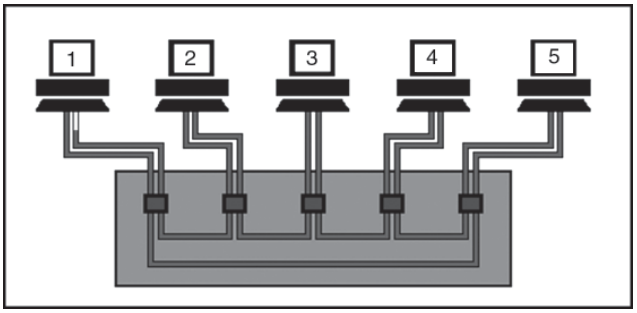
The ring polling allows all machines on the network to find out who is participating in the ring and to learn the address of their Nearest Active Upstream Neighbour (NAUN). Ring purges reset the ring after an interruption or loss of data is reported. Each machine knows the address of its Nearest Active Upstream Neighbour.

This is an important function in a Token Ring as it updates the information required to re-establish itself when machines enter or leave the ring. When a machine enters the ring it performs a lobe test to verify that its own connection is working properly, if it passes, it sends a voltage to the hub which operates a relay to insert it into the ring.

If a problem occurs anywhere on the ring, the machine that is immediately after the fault will cease to receive signals. If this situation continues for a short period of time it initiates a recovery procedure which assumes that its NAUN is at fault, the outcome of this procedure either removes its neighbour from the ring or it removes itself.

Token Ring Operation Using a Hub

A Token Ring hub simply changes the topology from a physical ring to a star wired ring. The Token still circulates around the network and is still controlled in the same manner, however, using a hub or a switch greatly improves reliability because the hub can automatically bypass any ports that are disconnected or have a cabling fault.



Further advancements have been made in recent years with regard to Token Ring technology, such as early Token release and Token Ring switching but as this site is primarily concerned with cabling issues we will not go into any more detail here.

Star Topology

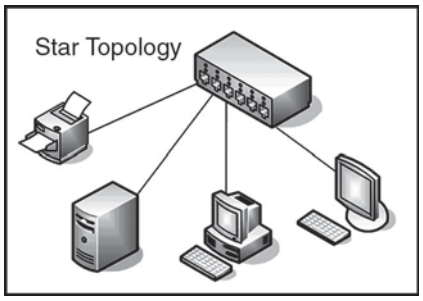
Star topology is the most commonly used physical technology in Ethernets LANS, when installed, the star topology resembles spokes I n a bicycle wheel. The star topology is made up of a central connection point that is a device such as a hub, switch, or router, where all the cabling segments meet. Each host in the network is connected to the central devices with its own cables although a physical star topology costs more to implement than the physical bus topology , the advantage of star topology make it worth the additional cost.

Because each host it connected to the central device with its own cable, when that cable has a problem, only that host is affected, the rest of the network remains operational this benefit is extremely important and is why virtually every newly designed Ethernet a central connection point might be desirable for security or restricted access, but this is also a main disadvantage of a star topology.if the central device fails, the whole network become disconnected.when a star network is expanded to include an additional networking device that is connected for the main networking device, it is called an extended star topology.

Description

A star topology is designed with each node (file server, workstations, and peripherals) connected directly to a central network hub or concentrator. Data on a star network passes through the hub or concentrator before continuing to its destination. The hub or concentrator manages and controls all

functions of the network. It also acts as a repeater for the data flow. This configuration is common with twisted pair cable; however, it can also be used with coaxial cable or fibre optic cable. Many businesses and home networks use the star topology. A star network features a central connection point called a "hub" that may be an actual hub or a switch as shown below:



Devices typically connect to the hub with Unshielded Twisted Pair (UTP) Ethernet cables also known as RJ45 cables. Compared to the bus topology, a star network generally requires more cable, but a failure in any star network cable will only take down one computer's network access and not the entire LAN.

Star topology has the following advantages:

- Easy to manage
- Easy to locate problems (cable/workstations)
- Easier to expand than a bus or ring topology
- Easy to Install and Wire
- No disruptions to the network then connecting or removing devices.
- Easy to detect faults and to remove parts.

Star topology has following disadvantages:

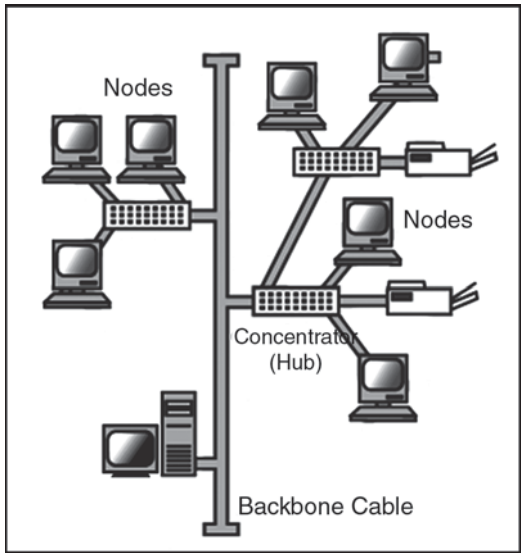
- It is susceptible to a single point of failure be it the server or hub/switch

- With Increased devices and traffic can make the network slow
- Requires more Cable Length.
- If the hub or concentrator fails, nodes attached are disabled.
- More expensive than linear bus topologies because of the cost of the concentrators.

Tree Topology

Among all the Network Topologies we can derive that the Tree Topology is a combination of the bus and the Star Topology. The tree like structure allows you to have many servers on the network and you can branch out the network in many ways.

This is particularly helpful for colleges, universities and schools so that each of the branches can identify the relevant systems in their own network and yet connect to the big network in some way.



A Tree Structure suits best when the network is widely spread and vastly divided into many branches. Like any other topologies, the Tree Topology has its advantages and disadvantages. A Tree Network may not suit small networks and it may be a waste of cable to use it for small networks. Tree Topology has some limitations and the configuration should suit those limitations.

Benefits

A Tree Topology is supported by many network vendors ad even hardware vendors. A point to point connection is possible with Tree Networks. All the computers have access to the larger and their immediate networks. Best topology for branched out networks.

Limitations

In a Network Topology the length of the network depends on the type of cable that is being used. The Tree Topology network is entirely dependant on the trunk which is the main backbone of the network. If that has to fail then the entire network would fail. Since the Tree Topology network is big it is difficult to configure and can get complicated after a certain point.

The Tree Topology follows a hierarchical pattern where each level is connected to the next higher level in a symmetrical pattern. Each level in the hierarchy follows a certain pattern in connecting the nodes. Like the top most level might have only one node or two nodes and the following level in the hierarchy might have few more nodes which work on the point to point connectivity and the third level also has asymmetrical node to node pattern and each of these

levels are connected to the root level in the hierarchy. A Tree Structured network is very similar to this and that is why it is called the Tree Topology.

Features

There will be at least three levels of hierarchy in the Tree Network Topology and they all work based on the root node. The Tree Topology has two kinds of topology integral in it, the star and the linear way of connecting to nodes. The Tree Topology functions by taking into account the total number of nodes present in the network. It does not matter how many nodes are there on each level. Nodes can be added to any level of the hierarchy and there are no limitations a far as the total number of nodes do not exceed. The higher levels in the hierarchy are expected to perform more functions than the lower levels in the network.

Advantages of a Tree Topology

- Point-to-point wiring for individual segments.
- Supported by several hardware and software venders.

Disadvantages of a Tree Topology

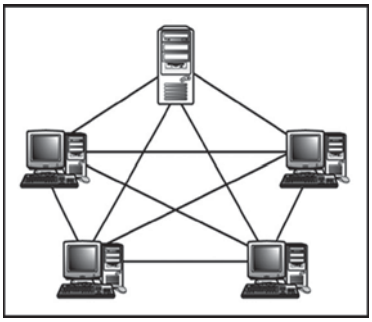
- Overall length of each segment is limited by the type of cabling used.
- If the backbone line breaks, the entire segment goes down.
- More difficult to configure and wire than other topologies.

Mesh Topology

A type of network setup where each of the computers and network devices are interconnected with one another,

allowing for most transmissions to be distributed, even if one of the connections go down. This type of topology is not commonly used for most computer networks as it is difficult and expensive to have redundant connection to every computer. However, this type of topology is commonly used for wireless networks. The *mesh* topology incorporates a unique network design in which each computer on the network connects to every other, creating a point-to-point connection between every device on the network.

The purpose of the mesh design is to provide a high level of redundancy. If one network cable fails, the data always has an alternative path to get to its destination.

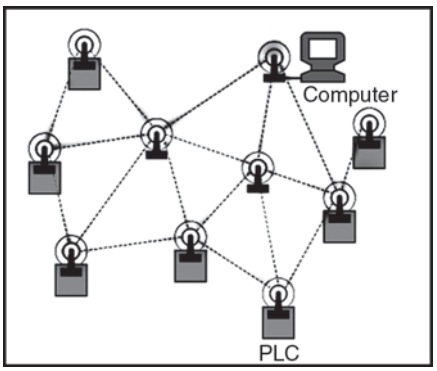


As seen in the above figure, the wiring for a mesh network can be very complicated. Further, the cabling costs associated with the mesh topology can be high, and troubleshooting a failed cable can be tricky. Because of this, the mesh topology is rarely used. A variation on a true mesh topology is the hybrid mesh. It creates a redundant point-to-point network connection between only specific network devices.

Wireless Mesh Networks

Wireless Mesh Networks work based on the radio frequencies and was originally developed by the army to be

able to communicate. The reliability factor is high in any kind of Mesh Network. There are three types of wireless Mesh Topologies.



- Fixed Wireless Connections
- Peer to Peer or Adhoc Networks
- Node to Node

Fixed Mesh Networks

The fixed Mesh Networks will work only in the specified location and they are not mobile networks. They are meant to be used in a limited surrounding with boundaries. The location of nodes in affixed Mesh Network is all pre determined and they are not interchangeable. The fixed Mesh Network does not work on line of sight like the other types of Mesh Networks. The total number of hops in a fixed Mesh Network is usually fixed and also short. There may not be many nodes as this kind of Mesh Networks exist within an office or building. More often than not the data travels ion a specific direction.

Peer to Peer Mobile Networks

In a peer to peer mobile network the individual devices connect to each other using the Mesh Network. The peer

does not require connecting to the main node and they can still communicate from one device to another device by taking the shortest possible data transfer route. However many experts believe that in the peer to peer Mesh Networks the problems with scalability in terms of time taken for data transfer is questionable. The device has to know to transmit the data in the most optimal path and the entire data transfer or depends on this single factor. If the device is incapable then the whole purpose of using it in a peer to peer connection is lost.

4

Software Testing

Introduction

Establishing Software testing Objectives is a critical part of planning the Software testing process. Defining testing objectives is also one of the most difficult test planning activities. It is difficult because humans frequently do not have a clear idea of what they want to do until they begin to do it.

This means the best laid test plans change during test process execution. This is a problem without a solution, but there are some actions testers can take which will improve test planning. The establishment of clear testing objectives goes a long way towards offsetting future execution problems. Before the tester can do this s/he must understand what we mean by the word objective.

An objective is a testing "goal." It is a statement of what the tester wants to accomplish when implementing a specific

testing activity. Each testing activity may have several objectives and there are two levels of objective specification.

A test plan should contain both high-level general objectives in the overview section, and specific low-level "provable" objectives for each particular type of testing being implemented. The latter kind being operational goals for specific testing tasks. A good set of operational objectives can intuitively explain why we are executing a particular step in the testing process.

Inputs

- System Requirements Document
- Software Design Description Document
- Risk Score Analysis Results (Task II.II)

Three methods can be used to specify test objectives. The first is brainstorming. The test team uses "creative" interaction to construct a list of test objectives. The second approach is to identify "key" System functions. Next, specify test objectives for each function. The third method is to identify business transactions and base objectives on them. This can also be thought of as Scenario-based as business cycles could be used to drive the process.

Output

Statement of Test Objectives - The statement of the test objectives is really a statement of the test requirements. It can be created using any word processing package or spread sheet. It can also be implemented with automated testing tools. As an example, in SQA's Manager product the test objects/requirements are input as a test requirements

hierarchy and are stored in the test repository. Each branch within the requirements tree can have sub-branches, and sub-branches can also have sub-branches. SQA is only one example. Other automated testing tools will have their own type of test objectives/requirements documentation.

Unit testing

These type of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Testing in software

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.This testing mode is a component of software development that takes a meticulous approach to building a product by means of continual testing and revision.

Once all of the units in a programme have been found to be working in the most efficient and error-free manner possible, larger components of the programme can be evaluated by means of integration testing. Unit testing must be done with an awareness that it may not be possible to test a unit for every input scenario that will occur when the programme is run in a real-world environment.

Rules

- Write the test first
- Never write a test that succeeds the first time
- Start with the null case, or something that doesn't work
- Don't be afraid of doing something trivial to make the test work
- Loose coupling and testability go hand in hand
- Use mock objects. A mock object is an object that pretends to be a particular type, but is really just a sink, recording the methods that have been called on it
- A test is not a pure unit test if: It talks to the database
  - It communicates across the network
  - It touches the file system
  - It can't run at the same time as any of your other unit tests
  - You have to do special things to your environment (such as editing config files) to run it. Tests that do these things should be kept aside from the regular unit test suit to run the test cases faster whenever we make changes.

Unit testing deals with testing a unit as a whole. This would test the interaction of many functions but confine the test within one unit. The exact scope of a unit is left to interpretation. Supporting test code, sometimes called scaffolding, may be necessary to support an individual test. This type of testing is driven by the architecture and implementation teams. This focus is also called black-box

testing because only the details of the interface are visible to the test. Limits that are global to a unit are tested here.

In the construction industry, scaffolding is a temporary, easy to assemble and disassemble, frame placed around a building to facilitate the construction of the building. The construction workers first build the scaffolding and then the building. Later the scaffolding is removed, exposing the completed building. Similarly, in software testing, one particular test may need some supporting software.

This software establishes an environment around the test. Only when this environment is established can a correct evaluation of the test take place. The scaffolding software may establish state and values for data structures as well as providing dummy external functions for the test. Different scaffolding software may be needed from one test to another test. Scaffolding software rarely is considered part of the system. Sometimes the scaffolding software becomes larger than the system software being tested. Usually the scaffolding software is not of the same quality as the system software and frequently is quite fragile. A small change in the test may lead to much larger changes in the scaffolding.

Internal and unit testing can be automated with the help of coverage tools. A coverage tool analyses the source code and generates a test that will execute every alternative thread of execution. It is still up to the programmer to combine these test into meaningful cases to validate the result of each thread of execution. Typically, the coverage tool is used in a slightly different way.

First the coverage tool is used to augment the source by placing informational prints after each line of code. Then

the testing suite is executed generating an audit trail. This audit trail is analysed and reports the per cent of the total system code executed during the test suite. If the coverage is high and the untested source lines are of low impact to the system's overall quality, then no more additional tests are required.

The idea behind unit testing is elegant and simple, but can be expanded to enable sophisticated series of tests for code validation and regression testing. A unit test is strictly something that 'exercises' or runs the code under test. Many developers manually perform unit testing on a regular basis in the course of working on a segment of code. In other words, it can be as simple as *'I know the code should perform this task when I supply this input; I'll try it and see what happens.'* If it doesn't behave as expected, the developer would likely modify the code and repeat this iterative process until it works.

The problem with doing this manually is that it can easily overlook large ranges of values or different combinations of inputs and it offers no insight into how much of the code was actually executed during testing. Additionally, it does not help us with the important task of proving to someone else that it worked and that it worked *correctly*.

The cost and time required is compounded by the reality that one round of testing is rarely enough; besides fixing bugs, any changes that are made to code later in the development process may require additional investment of time and resources to ensure it's working properly.

Large projects typically augment manual procedures with tools such as the Framework to automate and improve this

process. Automation mitigates risk of undetected errors, saves costs by detecting problems early, and saves time by keeping developers focused on the task of writing the software, instead of performing the tests themselves.

The idea behind unit testing is that once you have a unit that you think works, you set up a test case where you specify some input to the unit and compare the result of your unit with your expected result. You know about the expected result, because you know what your unit is doing (or you should know it).

Well, you better know what your unit is supposed to do, or else you should not do programming in the first place...;) Then you run the tests and the CakePHP/«insert your framework here» test suite tells you if they passed or if not, with some graceful message where the error occurred.

Now you have that testcase. Now you add testcases for t h a t input and this one. The advantage of this is that once you wrote the tests down they are there (cool, huh?) and you can hold on to them. There is no need anymore for you to open the browser and test everything manually again when you change your system. Instead, you add functionality, run the automated unit tests again, if they pass you are good to go, if they don't pass you broke something. Well, what if you broke something but your tests don't catch it? That's something that UT cannot do for you. You must make sure you have a good test coverage

Typically, the order of the running of the tests should not matter. There might be special cases, but in well over 90% it does not. This should also be your goal, too, to have two different problems if two test cases fail. Keep them all isolated

and you will sleep well. For most tests there is also not much configuration to be done. You specify the input, your expected result, crank the handle and evaluate how well you have done. You should typically be able to group tests together, too. When you run these groups you can get a good overview over large components of your system.

Testing Phase

The first test in the development process is the unit test. The source code is normally divided into modules, which in turn are divided into smaller units called units. These units have specific behaviour. The test done on these units of code is called unit test. Unit test depends upon the language on which the project is developed. Unit tests ensure that each unique path of the project performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Table. The Testing Phase: Improve Quality.

Phase	Deliverable
Testing	Regression Test
	Internal Testing
	Unit Testing
Application Testing	
	Stress Testing

Simply stated, quality is very important. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level. It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality. A customer satisfied with the quality of a product will remain

loyal and wait for new functionality in the next version. Quality is a distinguishing attribute of a system indicating the degree of excellence.

In many software engineering methodologies, the testing phase is a separate phase which is performed by a different team after the implementation is completed. There is merit in this approach; it is hard to see one's own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-read the material many times.

Unfortunately, delegating testing to another team leads to a slack attitude regarding quality by the implementation team. Alternatively, another approach is to delegate testing to the the whole organization. If the teams are to be known as craftsmen, then the teams should be responsible for establishing high quality across all phases. Sometimes, an attitude change must take place to guarantee quality.

Regardless if testing is done after-the-fact or continuously, testing is usually based on a regression technique split into several major focuses, namely internal, unit,application, and stress.The testing technique is from the perspective of the system provider.

Because it is nearly impossible to duplicate every possible customer's environment and because systems are released with yet-to-be-discovered errors, the customer plays an important, though reluctant, role in testing. As will be established later in the thesis, in the Water Sluice methodology this is accomplished in the alpha and beta release of the system.

Uses

Forget for a moment that there is something called XP (Extreme Programming) that coined the Unit Test term. The most of the projects developed today are always under tight development schedules and usually have only its developers as the tester of their code. By writing the unit tests themselves they can have a head start towards bug-free and quality code.

One will argue that if the developer is writing all the unit tests, it is quite possible to get the set of unit tests that are passable, because these unit tests are developed based either on the foreknowledge of application code or the assumptions made in the application code. However, do not be fooled with this, imagine what will happen if developer decides to change the application, her old test cases will break. That will force her to either re-think her changes or re-write the unit tests.

The application architect or analyst can write all the unit test cases upfront (Not what XP recommend, but we are not worried about it) and test the developed code against these cases and functionalities. The advantage is well defined deliverable for the developer and more quantifiable progress. A developer can also use this to disciple their work habits *e.g.* she can write a set of unit test that she wants to accomplish in a days work. Once tests ready, she can start developing the application and check her progress against the unit test. Now she has a metre to check her progress.

NUnit Framework

NUnit framework is port of JUnit framework from java and Extreme Programming (XP). This is an open source product. You can download it from http://www.nunit.org. The NUnit

framework is developed from ground up to make use of.NET framework functionalities. It uses an Attribute based programming model. It loads test assemblies in separate application domain hence we can test an application without restarting the NUnit test tools. The NUnit further watches a file/assembly change events and reload it as soon as they are changed. With these features in hand a developer can perform develop and test cycles sides by side.

Need for Software testing

A primary purpose for testing is to detect software failures so that defects may be uncovered and corrected. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the quality aspects of code: does it do what it is supposed to do and do what it needs to do. We test software because developers are unable to build defect free software. If the development processes were perfect, meaning no defects were produced, testing would not be necessary. Testing by the individual who developed the work has not proven to be a substitute to building and following a detailed test plan.

The disadvantages of a person checking their own work using their own documentation are as follows:

- Misunderstandings will not be detected, because the checker will assume that what the other individual heard from him was correct.
- Improper use of the development process may not be detected because the individual may not understand the process.

- The individual may be "blinded" into accepting erroneous system specifications and coding because he falls into the same trap during testing that led to the introduction of the defect in the first place.
- Information services people are optimistic in their ability to do defect-free work and thus sometimes underestimate the need for extensive testing.
- Without a formal division between development and test, an individual may be tempted to improve the system structure and documentation, rather than allocate that time and effort to the test.

Testing unveils design defects as well as data defects of any product. All testing focuses on discovering and eliminating defects or variances from what is expected.

Testers need to identify these two types of defects:

1. *Variance from Specifications*: A defect from the perspective of the builder of the product.
2. *Variance from what is Desired*: A defect from a user (or customer) perspective.

Background and Objectives

Software testing is an integral and important activity in every software development environment. Software seems to have has permeated almost every equipment that we use in our daily lives. Companies that produce embedded systems for use in health care, transportation, and other critical segments of our society have embraced model based software testing by integrating them into their development environments.

- Software Testing is designed to establish that the software is working satisfactorily as per the requirements.

- Software Testing is a process designed to prove that the programme is error free.
- Software The job of testing is to certify that the software does its job correctly and can be used in production.

Because, with these as the guidelines, one would tend to operate the system in a normal manner to see if it works and one would unconsciously choose such normal/correct test data as would prevent the system from failing. Besides, it is any way not possible to certify that a software has no errors, simply because it is almost impossible to detect all errors. In a way, we can say that software testing is basically a task of locating errors. From the objective point of view, testing can be done in two ways:

Positive Testing

Operate application or software as it should be operated. Use proper variety of test data, including data values at boundries to test if it fails.

Check actual test results with the expected and see:

- Does it behave normally?
- Are results correct?
- Does the software function correctly?

Negative Testing

Test for abnormal operations. Test with illegal/ abnormal test data. Intentionally attempt to make things go wrong and to discover/ detect and see

- Does the system fail/ crash?
- Does the programme do what it should not?
- Does it fail to do what it should?

Positive view of Negative Testing

The job of testing is to discover errors before the user does. A good tester is one who is successfull in making the system fail. Mentality of the tester has to be destructive – opposite to that of the creator/ developer which should be constructive.

This chapter is designed to enable a clear understanding and knowledge of the foundations, techniques, and tools in the area of software testing and its practice in the industry. The course will prepare students to be leaders in software testing. Whether you are a developer or a tester, you must test software. This course is a unique opportunity to learn strengths and weaknesses of a variety of software testing techniques.

Applications of testing techniques in health care industry (*e.g.* pacemaker), nuclear industry (*e.g.* plant control), aerospace industry (*e.g.* Mars Polar Lander), security (*e.g.* smart card), automobile industry (*e.g.* automotive control systems), and others will be considered.

The chapter will focus on:

- Test process and continuous quality improvement
- Test generation from requirements
- Modeling techniques: UML: FSM and Statecharts, Combinatorial design; and others.
- Test generation from models.
- Test adequacy assessment.
- Industrial applications.

Discussion oriented lectures by the instructor, in-class group presentations by teams, laboratory exercises using advanced testing tools, and invited talks by experts from the

industry will be the primary mechanisms for learning and the dissemination of knowledge.

Chapter description

Fundamentals of software testing; software test proces and continuous quality improvement; Test generation using finite state models,

Combinatorial design, and others; Test adequacy assessment using black box and white box criteria; Industrial applications of model based testing. Students will be required to form small teams of three or four, preferably interdisciplinary, and make presentations to the class.The work of each team will be reviewed by the instructor and other teams.

Test Approach

Inclusions

The contents of this release are as follows:

Phase 1 Deliverables

- New and revised Transaction Processing with automated support
- New Customer Query Processes and systems
- Revised Inter-Office Audit process
- Relocate Exceptions to Head Office
- New centralised Agency Management system
- Revised Query Management process
- Revised Retrievals process
- New International Reconciliation process
- New Account Reconciliation process

Exclusions

When the scope of each Phase has been agreed and signed off, no further inclusions will be considered for inclusion in this release, except:

- Where there is the express permission and agreement of the Business Analyst and the System Test Controller;
- Where the changes/inclusions will not require significant effort on behalf of the test team (*i.e.* requiring extra preparation - new test conditions etc.) and will not adversely affect the test schedule.

Specific Exclusions

- Cash management is not included in this phase
- Sign On/Sign Off functions are excluded - this will be addressed by existing processes
- The existing Special Order facility will not be replaced
- Foreign Currency Transactions
- International Data Exchanges
- Accounting or reporting of Euro transactions

Testing Process

- Organise Project involves creating a System Test Plan, Schedule and Test Approach, and requesting/ assigning resources.
- Design/Build System Test involves identifying Test Cycles, Test Cases, Entrance and Exit Criteria, Expected Results, etc. In general, test conditions/ expected results will be identified by the Test Team in conjunction with the Project Business Analyst or Business Expert. The Test Team will then identify Test Cases and the Data required. The Test

conditions are derived from the Business Design and the Transaction Requirements Documents

- Design/Build Test Procedures includes setting up procedures such as Error Management systems and Status reporting, and setting up the data tables for the Automated Testing Tool.
- Build Test Environment includes requesting/building hardware, software and data set-ups.
- Execute Project Integration Test - See Section 3 - Test Phases and Cycles
- Execute Operations Acceptance Test - See Section 3 - Test Phases and Cycles
- Signoff - Signoff happens when all pre-defined exit criteria have been achieved. See Section 2.4.

Exclusions

SQA will not deal directly with the business design regarding any design/ functional issues/ queries. The development team is the supplier to SQA - if design/ functional issues arise they should be resolved by the development team and its suppliers.

Testing Scope

Outlined below are the main test types that will be performed for this release. All system test plans and conditions will be developed from the functional specification and the requirements catalogue.

Functional Testing

The objective of this test is to ensure that each element of the application meets the functional requirements of the business as outlined in the:

- Requirements Catalogue
- Business Design Specification
- Year 2000 Development Standards
- Other functional documents produced during the course of the project *i.e.* resolution to issues/change requests/feedback.

This stage will also include Validation Testing - which is intensive testing of the new Front end fields and screens. Windows GUI Standards; valid, invalid and limit data input; screen and field look and appearance, and overall consistency with the rest of the application.

The third stage includes Specific Functional testing - these are low-level tests which aim to test the individual processes and data flows.

Integration Testing

This test proves that all areas of the system interface with each other correctly and that there are no gaps in the data flow. Final Integration Test proves that system works as integrated unit when all the fixes are complete.

Business (User) Acceptance Test

This test, which is planned and executed by the Business Representative(s), ensures that the system operates in the manner expected, and any supporting material such as procedures, forms etc. are accurate and suitable for the purpose intended. It is high level testing, ensuring that there are no gaps in functionality.

Performance Testing

These tests ensure that the system provides acceptable response times.

Regression Testing

A Regression test will be performed after the release of each Phase to ensure that:

- There is no impact on previously released software, and
- To ensure that there is an increase in the functionality and stability of the software.

The regression testing will be automated using the automated testing tool.

Bash and Multi-User Testing

Multi-user testing will attempt to prove that it is possible for an acceptable number of users to work with the system at the same time. The object of Bash testing is an ad-hoc attempt to break the system.

Technical Testing

Technical Testing will be the responsibility of the Development Team.

Operations Acceptance Testing (OAT)

This phase of testing is to be performed by the Systems Installation and Support group, prior to implementing the system in a live site. The SIS team will define their own testing criteria, and carry out the tests.

System Test Entrance/Exit Criteria

Entrance Criteria

The Entrance Criteria specified by the system test controller, should be fulfilled before System Test can commence. In the

event, that any criterion has not been achieved, the System Test may commence if Business Team and Test Controller are in full agreement that the risk is manageable.

- All developed code must be unit tested. Unit and Link Testing must be completed and signed off by development team.
- System Test plans must be signed off by Business Analyst and Test Controller.
- All human resources must be assigned and in place.
- All test hardware and environments must be in place, and free for System test use.
- The Acceptance Tests must be completed, with a pass rate of not less than 80%.

Acceptance Tests

25 test cases will be performed for the acceptance tests. To achieve the acceptance criteria 20 of the 25 cases should be completed successfully - *i.e.* a pass rate of 80% must be achieved before the software will be accepted for System Test proper to start. This means that any errors found during acceptance testing should not prevent the completion of 80% of the acceptance test applications.

Resumption Criteria

In the event that system testing is suspended resumption criteria will be specified and testing will not re-commence until the software reaches these criteria.

Exit Criteria

The Exit Criteria detailed below must be achieved before the Phase 1 software can be recommended for promotion to

Operations Acceptance status. Furthermore, I recommend that there be *a minimum 2 days effort Final Integration testing AFTER the final fix/change has been retested.*

- All High Priority errors from System Test must be fixed and tested
- If any medium or low-priority errors are outstanding - the implementation risk must be signed off as acceptable by Business Analyst and Business Expert
- Project Integration Test must be signed off by Test Controller and Business Analyst.
- Business Acceptance Test must be signed off by Business Expert.

Completion Criteria

A completion criterion is the standard by which a test objective is measured. Completion criteria can be either quantitative or qualitative. The important point is that the test team must some how be able to determine when a test objective has been satisfied. One or more completion criteria must be specified for each test objective.

Output

Statement of Objective Completion Criteria - The important consideration is that each requirement and how it is validated is documented. Test requirements are completely useless unless they can be satisfied. Important test metrics that should be calculated and reported are the percentage of test requirements that have be covered by test cases, and the percentage of test requirements that have been successfully validated. The statement of objective completion criteria does not have to be a separate document. It can simply be an

addendum to the statement of test objectives. For example, if using SQA's Manager product, The description field that is included for each requirement could contain a statement of the requirement's validation rule(s). The test objectives should be prioritized based on the risk analysis findings.

Priority should be assigned using this scale:

- *High:* Most important tests: must be executed
- *Medium:* Second-level objectives: should be executed only after high-priority tests
- *Low:* Least important: should be tested last and only if there is enough time
- *High and Medium:* test objectives should be assigned more resources than Low priority objectives.
- *Output:* Prioritized Test Objectives
- *Manual:* Test objectives should be implemented manually in the form of quality checklists, with one or more checklist items satisfying a specific objective. (Single checklist items can also satisfy more than one objective, as is the case for the date field objectives).
- *Automated:* Test objectives should be translated into an appropriate form for the automated test tool being used. For example, when using SQA TeamTest Test Manager a test requirements hierarchy would be created. Automated test requirements would be stored in the tool's test repository and would be used as the basis for constructing automated test scripts.

5

Software Progrmming

Software Engineering is an approach to developing software that attempts to treat it as a formal process more like traditional engineering than the craft that many programmers believe it is. We talk of crafting an application, refining and polishing it, as if it were a wooden sculpture, not a series of logic instructions. Manufacturers cannot build complex life-critical systems like aircraft, nuclear reactor controls, medical systems and expect the software to be thrown together.

They require the whole process to be thoroughly managed, so that budgets can be estimated, staff recruited, and to minimize the risk of failure or expensive mistakes. In safety critical areas such as aviation, space, nuclear power plants, medicine, fire detection systems, and roller coaster rides the cost of failure can be enormous as lives are at risk. A divide by zero error that brings down an aircraft is just not acceptable.

Cad Engineering

Enormous design documents- hundreds or thousands of pages long are produced using C.A.S.E. (Computer Aided Software Engineering) tools then converted into Design Specification documents which are used to design code.

C.A.S.E suffers from the "not quite there yet" syndrome. There are no systems that can take a set of design constraints and requirements then generate code that satisfies all the requirements and constraints. Its far too complex a process. So the available C.A.S.E. systems manage parts of the lifecycle process but not all of it. One distinguishing feature of Software Engineering is the paper trail that it produces.

Designs have to be signed off by Managers and Technical Authorities all the way from top to bottom and the role of Quality Assurance is to check the paper trail. Many Software Engineers would admit that their job is around 70% paperwork and 30% code. It's a costly way to write software and this is why avionics in modern aircraft are so expensive.

Software Crisis

Indeed, the problem of trying to write an encyclopedia is very much like writing software. Both running code and a hypertext/encyclopedia are wonderful turn-ons for the brain, and you want more of it the more you see, like a drug. As a user, you want it to do everything, as a customer you don't really want to pay for it, and as a producer you realise how unrealistic the customers are. Requirements will conflict in functionality vs affordability, and in completeness vs timeliness.

Different Types of Crisis

Chronic Software Crisis

By today's definition, a "large" software system is a system that contains more than 50,000 lines of high-level language code. It's those large systems that bring the software crisis to light. If you're familiar with large software development projects, you know that the work is done in teams consisting of project managers, requirements analysts, software engineers, documentation experts, and programmers.

With so many professionals collaborating in an organized manner on a project, what's the problem? Why is it that the team produces fewer than 10 lines of code per day over the average lifetime of the project? And why are sixty errors found per every thousand lines of code? Why is one of every three large projects scrapped before ever being completed? And why is only 1 in 8 finished software projects considered "successful?"

- The cost of owning and maintaining software in the 1980's was twice as expensive as developing the software.
- During the 1990's, the cost of ownership and maintenance increased by 30% over the 1980's.
- In 1995, statistics showed that half of surveyed development projects were operational, but were not considered successful.
- The average software project overshoots its schedule by half.
- Three quarters of all large software products delivered to the customer are failures that are either not used at all, or do not meet the customer's requirements.

Software projects are notoriously behind schedule and over budget. Over the last twenty years many different paradigms have been created in attempt to make software development more predictable and controllable.

While there is no single solution to the crisis, much has been learned that can directly benefit today's software projects.

It appears that the Software Crisis can be boiled down to two basic sources:

1. Software development is seen as a craft, rather than an engineering discipline.
2. The approach to education taken by most higher education institutions encourages that "craft" mentality.

Software Development

Software development today is more of a craft than a science. Developers are certainly talented and skilled, but work like craftsmen, relying on their talents and skills and using techniques that cannot be measured or reproduced. On the other hand, software engineers place emphasis on reproducible, quantifiable techniques–the marks of science. The software industry is still many years away from becoming a mature engineering discipline.

Formal software engineering processes exist, but their use is not widespread. A crisis similar to the software crisis is not seen in the hardware industry, where well documented, formal processes are tried and true, and ad hoc hardware development is unheard of. To make matters worse, software technology is constrained by hardware technology. Since

hardware develops at a much faster pace than software, software developers are constantly trying to catch up and take advantage of hardware improvements.

Management often encourages ad hoc software development in an attempt to get products out on time for the new hardware architectures. Design, documentation, and evaluation are of secondary importance and are omitted or completed after the fact. However, as the statistics show, the ad hoc approach just doesn't work. Software developers have classically accepted a certain number of errors in their work as inevitable and part of the job. That mindset becomes increasingly unacceptable as software becomes embedded in more and more consumer electronics. Sixty errors per thousand lines of code is unacceptable when the code is embedded in a toaster, automobile, ATM machine or razor (let your imagination run free for a moment).

Computer Science and Product Orientation

Software developers pick up the ad hoc approach to software development early in their computer science education, where they are taught a "product orientation" approach to software development. In the many undergraduate computer science courses I took, the existence of software engineering processes was never even mentioned.

Computer science education does not provide students with the necessary skills to become effective software engineers. They are taught in a way that encourages them to be concerned only with the final outcome of their assignments–whether or not the programme runs, or whether or not it runs efficiently, or whether or not they used the

best possible algorithm. Those concerns in themselves are not bad. But on the other hand, they should not be the focus of a project. The focus should be on the complete process from beginning to end and beyond. Product orientation also leads to problems when the student enters the work force– not having seen how processes affect the final outcome, individual programmers tend to think their work from day to day is too "small" to warrant the application of formal methods.

Fully Supported Software

As we have seen, most software projects do not follow a formal process. The result is a product that is poorly designed and documented. Maintenance becomes problematic because without a design and documentation, it's difficult or impossible to predict what sort of effect a simple change might have on other parts of the system. Fortunately there is an awareness of the software crisis, and it has inspired a worldwide movement towards process improvement. Software industry leaders are beginning to see that following a formal software process consistently leads to better quality products, more efficient teams and individuals, reduced costs, and better morale.

Ratings range from Maturity Level 1, which is characterized by ad hoc development and lack of a formal software development process, up to Maturity Level 5, at which an organization not only has a formal process, but also continually refines and improves it. Each maturity level is further broken down into key process areas that indicate the areas an organization should focus on to improve its software process (*e.g.* requirement analysis, defect

prevention, or change control). Level 5 is very difficult to attain. In early 1995, only two projects, one at Motorola and another at Loral (the on-board space shuttle software project), had earned Maturity Level 5. Another study showed that only 2% of reviewed projects rated in the top two Maturity Levels, in spite of many of those projects placing an extreme emphasis on software process improvement.

Customers contracting large projects will naturally seek organizations with high CMM ratings, and that has prompted increasingly more organizations to investigate software process improvement. Mature software is also reusable software. Artisans are not concerned with producing standardized products, and that is a reason why there is so little interchangeability in software components.

Ideally, software would be standardized to such an extent that it could be marketed as a "part", with its own part number and revision, just as though it were a hardware part. The software component interface would be compatible with any other software system. Though it would seem that nothing less than a software development revolution could make that happen, the National Institute of Standards and Technology (NIST) founded the Advanced Technology Programme (ATP), one purpose of which was to encourage the development of standardized software components.

Extreme Programming (XP)

Is the latest incarnation of Waterfall model and is the most recent software fad. Most postulates of Extreme programming are pure fantasy. It has been well known for a long time that *big bang* or waterfall models don't work well on projects with complex or shifting requirements. The same is true for

XP. Too many shops implement XP as an excuse for not understanding the user requirements. XP try improve classic waterfall model by trying to start coding as early as possible but without creating a full-fledged prototype as the first stage. In this sense it can be considered to be variant of evolutionary prototyping (see below). Often catch phase "Emergent design" is used instead of evolutionary prototyping.

It also introduces a very questionable idea of pair programming as an attempt to improve extremely poor communication between developers typical for large projects. While communication in large projects is really critical and attempts to improve it usually pay well, "pair programming" is a questionable strategy.

There are two main problems here:

1 In a way it can be classified as a hidden attempt to create one good programmer out of two mediocre. But in reality it is creating one mediocre programmer from two or one good. No senior developer is going to put up with some jerk sitting on his lap asking questions about every line. It just prevents the level of concentration needed for high quality coding. Microsoft's idea of having a tester for each programmer is more realistic: one developer writes tests.

2 The actual code to be tested. This forces each of them to communicate and because tester has different priorities then developer such communication brings the developer a new and different perspective on his code, which really improves quality. This combination of different perspectives is a really neat idea as you can see from the stream of Microsoft Office products and operating systems.

Spiral model

The spiral model is a variant of "dialectical spiral" and as such provides useful insights into the life cycle of the system. Can be considered as a generalization of the prototyping model. That why it is usually implemented as a variant of prototyping model with the first iteration being a prototype.

The spiral model is similar to the incremental model, with more emphases placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed.

Each subsequent spirals builds on the baseline spiral. Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. Software is produced in the engineering phase, along with testing at the end of the phase.

Advantages

- High amount of risk analysis
- Good for large and mission-critical projects.
- Software is produced early in the software life cycle.

Disadvantages

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

Evolutionary prototyping model

This is kind of mix of Waterfall model and prototyping. Presuppose gradual refinement of the prototype until a usable product emerge. Might be suitable in projects where the main problem is user interface requirements, but internal architecture is relatively well established and static. In this case system first is coded in a scripting language and then gradually critical components are rewritten in the lower language.

OSS development model

It is the latest variant of evolutionary prototype model. The "waterfall model" was probably the first published model and as a specific model for military it was not as naive as some proponents of other models suggest. The model was developed to help cope with the increasing complexity of aerospace products. The waterfall model followed a documentation driven paradigm.

Prototyping model was probably the first realistic of early models because many aspects of the syst4m are unclear until a working prototype is developed. A better model, the "spiral model" was suggested by Boehm in 1985. The spiral model is a variant of "dialectical spiral" and as such provides useful insights into the life cycle of the system. But it also presuppose unlimited resources for the project. No organization can perform more then a couple iterations during the initial development of the system. the first iteration is usually called prototype. Prototype based development requires more talented managers and good planning while waterfall model works (or does not work) with

bad or stupid managers works just fine as the success in this model is more determined by the nature of the task in hand then any organizational circumstances.

Like always humans are flexible and programmer in waterfall model can use guerilla methods of enforcing a sound architecture as manager is actually a hostage of the model and cannot afford to look back and re-implement anything substantial. Because the life cycle steps are described in very general terms, the models are adaptable and their implementation details will vary among different organizations.

The spiral model is the most general. Most life cycle models can in fact be derived as special instances of the spiral model. Organizations may mix and match different life cycle models to develop a model more tailored to their products and capabilities.

There is nothing wrong about using waterfall model for some components of the complex project that are relatively well understood and straightforward. But mixing and matching definitely needs a certain level of software management talent.

V-Shaped Model

Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing is emphasized in this model more so than the waterfall model though. The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation.

Requirements begin the life cycle model just like the waterfall model. Before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in the requirements gathering.

The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

The low-level design phase is where the actual software components are designed, and unit tests are created in this phase as well. The implementation phase is, again, where all coding takes place.

Advantages

- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

Disadvantages

- Very rigid, like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- Model doesn't provide a clear path for problems found during testing phases.

Software Requirements Specification

There are many good definitions of System and Software Requirements Specifications that will provide us a good basis upon which we can both define a great specification and help us identify deficiencies in our past efforts. There is also a lot of great stuff on the web about writing good specifications. The problem is not lack of knowledge about how to create a correctly formatted specification or even what should go into the specification. The problem is that we don't follow the definitions out there.

We have to keep in mind that the goal is not to create great specifications but to create great products and great software. Can you create a great product without a great specification? Absolutely! You can also make your first million through the lottery – but why take your chances? Systems and software these days are so complex that to embark on the design before knowing what you are going to build is foolish and risky.

The IEEE is an excellent source for definitions of System and Software Specifications. As designers of real-time, embedded system software, we use IEEE STD 830-1998 as the basis for all of our Software Specifications unless specifically requested by our clients. Essential to having a great Software Specification is having a great System Specification. The equivalent IEEE standard for that is IEEE STD 1233-1998. However, for most purposes in smaller systems, the same templates can be used for both.

Benefits of SRS

Establish the basis for agreement between the customers and the suppliers on what the software product is to do. The

complete description of the functions to be performed by the software specified in the SRS will assist the potential users to determine if the software specified meets their needs or how the software must be modified to meet their needs.

Reduce the development effort. The preparation of the SRS forces the various concerned groups in the customer's organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.

Provide a basis for estimating costs and schedules. The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates. Provide a baseline for validation and verification. Organizations can develop their validation and Verification plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.

Facilitate transfer. The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.

Serve as a basis for enhancement. Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the

finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

Characteristics

An SRS should be:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable
  - *Correct*: This is like motherhood and apple pie. Of course you want the specification to be correct. No one writes a specification that they know is incorrect. We like to say - "Correct and Ever Correcting." The discipline is keeping the specification up to date when you find things that are not correct.
  - *Unambiguous*: An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Again, easier said than done. Spending time on this area prior to releasing the SRS can be a waste of time. But as you find ambiguities - fix them.
  - *Complete*: A simple judge of this is that is should be all that is needed by the software designers to create the software.

- *Consistent*: The SRS should be consistent within itself and consistent to its reference documents. If you call an input "Start and Stop" in one place, don't call it "Start/Stop" in another.
- *Ranked for Importance*: Very often a new system has requirements that are really marketing wish lists. Some may not be achievable. It is useful provide this information in the SRS.
- *Verifiable*: Don't put in requirements like - "It should provide the user a fast response." Another of my favorites is - "The system should never crash." Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."

Architectural design principles

Design principles are not necessarily right or wrong but should be an accurate reflection of the fundamentals that guide decision making in an enterprise. The following should therefore not be seen as design principles fixed in concrete but rather as examples of business principles. Best practice is to define the design principle in terms of its Benefits and rationale as well as the implication to the enterprise and the counter argument expressing the potential negative impact of the design principle.

Design principle

Description

All management information and business intelligence will be sourced from a single consolidated source of information.

Benefits

- A central source of management information will provide the enterprise with a wide breath of reporting and analysis without being constraint by the organisations functional structuring.
- Users will become used to a single interface to management information allowing managers to become familiar with the infrastructure and extracting maximum benefit from all information available in the organisation.
- The central information will eliminate contradicting information sources and ensure accurate reporting of current affairs and identification of issues and opportunities.
- Increase the flexibility and manageability of providing information rapidly and effectively to support business decisions.
- Use best of breed analytic functionality to support management decision making
- Increased security in managing access to The enterprise's management information

Implications

- Information must be sourced to the central information infrastructure from all the various operational applications as close to real time as possible
- No additional analytical modules are required for transactional applications.

- The interface to management information and training should be rolled out to all decision makers to effectively access information.

Counter Argument

- The central management information might not be adequate in situations where real time analytics of transactional information is needed
- Information in the central information source might not be structured for a specific requirement and the development time might be too long to provide the information in time for a once off request.
- The assumption cannot be made that a single tool will satisfy all the information requirements. The information infrastructure will therefore consist of a variety of integrated tools.

Internationalisation

Description

Information must be structured for global deployment in various cultures and support multi-currency, multi-language and multi platforms

Benefits

- Flexibility to enter into other global markets
- Consistency of being able to deploy a proven business model and then adapt to local conditions

Implications

- Applications should be much more flexible to accommodate differences defined by different countries

- Current applications should be evaluated in terms of internationalisation requirements.

Counter Argument

- Applications designated for use locally in South-Africa only does not have to comply to the internationalisation principle
- The enterprise might strategise to enter only into markets that have a certain set of international commonality which make the rigid application of this principle unnecessary
- It might be too expensive to change or replace legacy systems to adhere to the internationalisation requirement.

Single contact database

Description

A single contact database for all business contacts *e.g.* policyholders, intermediaries and service providers

Benefits

- A central source of information to manage relationships effectively
- Have a more comprehensive view of interrelationships between business contacts.
- More effective marketing campaign design and management

Implication

- High levels of data integration with transactional systems updating contact information.

- Transactional systems updating information must be assigned with levels of trust.
- Central contact information should not.complicate functional requirements to only view specific relationships.

Counter Argument

- Some functional units might not want to share contact information for the fear that it might be used wrongly or out of context by other parts of the organisation.

Single point of authentication

Description

Access for to information should be constraint through a single point of authentication infrastructure

Benefits

- Improved security of information

Implication

- Central management of user access to information.

Counter Argument

- The current infrastructure might not be mature enough to implement the principle.

Data quality measurement

Description

Data quality will be measured both in quantitative and qualitative terms eg. Audit procedures and Data quality questionnaires

Benefits

- Improved data quality
- Accurate usage of data
- Improved management information
- Increased operational efficiency

Implication

- Bi-annually measure data through with a data quality survey
- Audit data ownership procedures annually
- Build in data quality measures into applications
- Connect data quality results with performance incentives

Counter Argument

- Regular audits and subjective measurement techniques *e.g.* surveys might be time consuming.

Formalised data exchange/enrichment

Description

All data exchange/enrichment activities are managed and approved by the appointed data strategist and the exchange of information must be subjected to a standardised methodology for information exchange/enrichment.

Benefits

- Control costs associated with data exchange and enrichment
- Protect operational data
- Improve data quality and value
- Data sharing to allow for better detection of fraud

Implication

- Development of a formal policy and methodology for data exchange and enrichment
- Data strategist must be responsible for approving data exchange/enrichment efforts and minimise cost.
- Identification and management of organisations that can enrich and/or validate the enterprise data.

Counter Argument

- The formalised methodology should not become a constraint to enrich and improve data quality.
- The enterprise might not always be in a position to demand compliance from external parties to comply with data sharing standards.
- Current lack of industry standards might make it difficult to implement the principle.

Central repository of data naming standards

Description

Data names and field content must be standardised through a central reference repository and must be accessible to the business *e.g.* Street rather than str. is used to reference a street.

Benefits

- Consistency of information across all business processes
- Usability of information increases across the organisation.

Implication

- Alignment of all applications to support the standardised naming standards

Counter Argument

- Difficulty to implement naming standardisation in some applications

Align information requirements with data model

Description

All information requirements must be aligned with the corporate data model before requesting changes to the information architecture:

Benefits

- Integrity of transactional data model stays in tact.
- Prevent duplication of information.

Implication

- The corporate data model must be maintained to be up to date at all times.
- In any application development life cycle it is a condition to align the application with the corporate data model.

Counter Argument

Data owners might not understand the data model to update it with changes.

Information governance on all data elements

Description

All information elements must be subjected to information architecture governance

Benefits

- Sustain data quality.

- Improve operational efficiency.

Implication

- Design the business process application of the data element.
- Assign Applications sourcing the information.
- Align with corporate data model, rules, validations, naming standard.
- Define data management policies *e.g.* security, back-up, archiving/retrieval.
- Assign data ownership.

Counter Argument

- Lack training to adhere to information governance.
- Information governance is not adequately communicated.

Data privacy and legality

Description

Client privacy must be respected and legal requirements must be complied with, in any event of data exchange or commerce.

Benefit

- Maintain good relationships with clients and protect premium income
- Avoid legal costs due to mismanagement of information resulting in lawsuits.

Implication

- The enterprise must be up to date with laws relating to information

- Communicating The enterprise's data policy to clients
- Legal department needs to be up to date with laws governing information usage, commerce and distribution

Counter Argument

- Uncertainty of what constitutes data privacy might make it difficult to implement this principle.

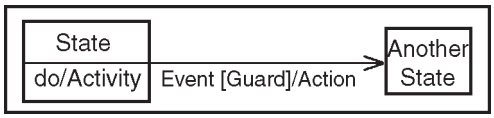
Bottom line: Design principles for enterprise architecture must provide a decision framework based on a clear rationale defined in terms of the benefits, implications and counter argument relating to the design principle

Unified Modeling Language (UML)

The Unified Modeling Language or UML is is a mostly graphical modelling language that is used to express designs. It is a standardized language in which to specify the artefacts and components of a software system. It is important to understand that the UML describes a notation and not a process. It does not put forth a single method or process of design, but rather is a standardized tool that can be used in a design process.

State Diagram

The state diagram shows the change of an object through time. Based upon events that occur, the state diagram shows how the object changes from start to finish.



States are represented as a rounded rectangle with the name of the state shown. Optionally you can include an activity that represents a longer running task during that state. Connecting states together are transitions. These represent the events that cause the object to change from one state to another. The guard clause of the label is again mutually exclusive and must resolve itself to be either true or false. Actions represent tasks that run causing the transitions.

Actions are different from activities in that actions cannot be interrupted, while an activity can be interrupted by an incoming event. Both ultimately represent an operation on the object being studied. For example, an operation that sets an attribute would be considered an action, while a long calculation might be an activity. The specific separation between the two depends on the object and the system being studied.

Architectural patterns

Patterns for system architecting are very much in their infancy. They have been introduced into TOGAF essentially to draw them to the attention of the systems architecture community as an emerging important resource, and as a placeholder for hopefully more rigorous descriptions and references to more plentiful resources in future versions of TOGAF. They have not (as yet) been integrated into TOGAF. However, in the following, we attempt to indicate the potential value to TOGAF, and to which parts of the TOGAF Architecture Development Method (ADM) they might be relevant.

Background

A "pattern" has been defined as: "an idea that has been useful in one practical context and will probably be useful in others" [*Analysis Patterns - Reusable Object Models*]. In TOGAF, patterns are considered to be a way of putting building blocks into context; for example, to describe a re-usable solution to a problem. Building blocks are what you use: patterns can tell you how you use them, when, why, and what trade-offs you have to make in doing so. Patterns offer the promise of helping the architect to identify combinations of Architecture and/or Solution Building Blocks (ABBs/SBBs) that have been proven to deliver effective solutions in the past, and may provide the basis for effective solutions in the future.

Content of a Pattern

Several different formats are used in the literature for describing patterns, and no single format has achieved widespread acceptance. However, there is broad agreement on the types of things that a pattern should contain. The headings which follow are taken from *Pattern-Oriented Software Architecture: A System of Patterns*. The elements described below will be found in most patterns, even if different headings are used to describe them.

Name

A meaningful and memorable way to refer to the pattern, typically a single word or short phrase.

Problem

A description of the problem indicating the intent in

applying the pattern - the intended goals and objectives to be reached within the context and forces described below (perhaps with some indication of their priorities).

Context

The preconditions under which the pattern is applicable - a description of the initial state before the pattern is applied.

Forces

A description of the relevant forces and constraints, and how they interact/conflict with each other and with the intended goals and objectives. The description should clarify the intricacies of the problem and make explicit the kinds of trade-offs that must be considered. (The need for such trade-offs is typically what makes the problem difficult, and generates the need for the pattern in the first place.) The notion of "forces" equates in many ways to the "qualities" that architects seek to optimize, and the concerns they seek to address, in designing architectures.

For example:

- Security, robustness, reliability, fault-tolerance
- Manageability
- Efficiency, performance, throughput, bandwidth requirements, space utilization
- Scalability (incremental growth on-demand)
- Extensibility, evolvability, maintainability
- Modularity, independence, re-usability, openness, composability (plug-and-play), portability
- Completeness and correctness
- Ease-of-construction

- Ease-of-use
- etc....

A description, using text and/or graphics, of how to achieve the intended goals and objectives. The description should identify both the solution's static structure and its dynamic behaviour - the people and computing actors, and their collaborations. The description may include guidelines for implementing the solution. Variants or specializations of the solution may also be described.

Resulting Context

The post-conditions after the pattern has been applied. Implementing the solution normally requires trade-offs among competing forces. This element describes which forces have been resolved and how, and which remain unresolved. It may also indicate other patterns that may be applicable in the new context. (A pattern may be one step in accomplishing some larger goal.) Any such other patterns will be described in detail under Related Patterns.

Examples

One or more sample applications of the pattern which illustrate each of the other elements: a specific problem, context, and set of forces; how the pattern is applied; and the resulting context.

Rationale

An explanation/justification of the pattern as a whole, or of individual components within it, indicating how the pattern actually works, and why - how it resolves the forces to achieve the desired goals and objectives, and why this is "good". The

Solution element of a pattern describes the external structure and behaviour of the solution: the Rationale provides insight into its internal workings.

Related Patterns

The relationships between this pattern and others. These may be predecessor patterns, whose resulting contexts correspond to the initial context of this one; or successor patterns, whose initial contexts correspond to the resulting context of this one; or alternative patterns, which describe a different solution to the same problem, but under different forces; or co-dependent patterns, which may/must be applied along with this pattern.

Known Uses

Known applications of the pattern within existing systems, verifying that the pattern does indeed describe a proven solution to a recurring problem. Known Uses can also serve as Examples.

Patterns may also begin with an Abstract providing an overview of the pattern and indicating the types of problems it addresses. The Abstract may also identify the target audience and what assumptions are made of the reader.

Low Level Design

The low level design document should contain a listing of the declarations of all the classes, non-member-functions, and class member functions that will be defined during the implementation stage, along with the associations between those classes and any other details of those classes (such as member variables) that are firmly determined by the low level

design stage. The low level design document should also describe the classes, function signatures, associations, and any other appropriate details, which will be involved in testing and evaluating the project according to the evaluation plan defined in the project's requirements document.

More importantly, each project's low level design document should provide a narrative describing (and comments in your declaration and definition files should point out) how the high level design is mapped into its detailed low-level design, which is just a step away from the implementation itself. This should be an English description of how you converted the technical diagrams (and text descriptions) found in your high level design into appropriate class and function declarations in your low level design. You should be especially careful to explain how the class roles and their methods were combined in your low level design, and any changes that you decided to make in combining and refining them.

Description

Control systems elements like Advanced Metering Infrastructure (AMI) networks fully field wireless sensors and controls outside a utility's physical security perimeter, placing them at a high risk of compromise. System attackers have every opportunity to damage, sniff, spoof, or tamper communications hardware platforms for malicious, hobbyist, or incidental reasons.

This paper demonstrates the relevance of common control systems communications hardware vulnerabilities that lead to direct control systems compromise. The paper describes several enabling vulnerabilities exploitable by an attacker,

the design principles that causing them to arise, the economic and electronic design constraints that restrict their defence, and ideas for vulnerability avoidance.

Topics include design induced vulnerabilities such as the extraction and modification of communications device firmware, man-in-the-middle attacks between chips of a communications devices, circumvention of protection measures, bus snooping, and other attacks. Specific examples are identified in this report, ranked by attack feasibility. Each attack was investigated against actual IEEE 802.15.4 radio architectures.

Embedded System Architecture

Standard wireless embedded implementation technologies such as IEEE 802.15.4 are generally designed to serve specific market needs. Therefore, the market offers components that translate such standards to mass producible designs. Embedded wireless technologies typically, but not always, have relatively low power consumption, component cost, computational power requirements, design cost, and implementation cost.

Commodity variants of components that implement wireless technology generally have higher individual reliability than custom designs, and a ready and willing engineer talent pool to integrate them. Almost all such components are designed to leverage or integrate with existing mass production components and subcomponents such as microcontrollers, RAM chips, ROM chips, and others.

All of the above is highly desirable. As with all such technologies that have the potential to achieve economy of

scale in design and implementation, vulnerability generally follows or surpasses all cost optimizations and design trade-offs unless specifically mitigated. Such optimizations and economies of scale can serve to broaden the impact of overlooked security flaws, turning their advantage into a weakness.

This paper does not attempt to cover all potential aspects for such wireless technology implementations, much less the entire range of implementation issues for a single technology. We present security vulnerabilities for typical components found in specific IEEE 802.15.4 implementations; and abstract them to help translate real-world tactical security vulnerabilities as recognizable design classes requiring consideration for mitigation. This paper does not educate the reader in the many nuances of RF design. For RF design and implementation issues, see individual standards such as and engineering references including, but not limited to. We present an abstraction of monolithic vulnerable aspects of a typical commodity IEEE 802.15.4 platform, the Telos-B development kit.

While the Telos-B is a basic user-programmable development kit, its architecture is close enough to most typical applications to be considered general. This abstraction is intended to give the reader a repeatable context as a starting point when looking at other platform architectures.

The RF physical, media access, link layer, and sometimes network layers will be offloaded onto an RF component such as the pictured CC2420. Breaking up the design lets designers implement the RF portion of the application with the best possible RF module for the lowest time to market

while targeting host applications to the optimal host processor. Most standalone communications modules will be linked to their host processor by a trivial board-level serial bus such as SPI or I2C. In some designs the host processor also contains the RF stack implementation, eliminating the board-level serial bus. Components such as microcontrollers or host processors rarely fully implement the analog portion of an RF module.

Antennae, inbound and outbound amplifiers, RF switches, and various filters are generally integrated separately as their requirements vary widely across potential applications. Due to their application orientation, host processors will have external timing means.

In general external oscillators reduce processor chip cost and allow the designer to scale the system to the cheapest clock source meeting application requirements. Though typically not used, many 802.15.4 RF modules have a means to slave a host's clock to the RF module to further reduce design cost. Power is supplied to the devices as required by the module, though often platform power requirements are aligned to reduce component count and subsequent cost.

Confidentiality

- Snooping Bus Traffic
- Extracting Firmware for Vulnerability Analysis
- Extracting Stored Information
- Snooping Side Channels

Integrity

- Tampering Bus Traffic
- Replacing Hardware Components

- Modifying Existing Components
- Bypassing Hardware Components
- Disrupting or Distorting Normal Hardware Operation
- Bypassing Software Components

Availability

- Jamming or Shrouding
- Alert/Condition Flooding
- Run Battery Down

PHY, Link Transceiver

The subcomponent that deals with PHY, MAC, and LINK layer issues. Potentially executes link layer cryptography algorithms.

Key Subcomponents: Registers, RAM, other storage, boot loader, internal programme storage, internal timing source, and architecture specific functionality

Key External Dependencies: RF Front End, NET & App Controller, data bus to NET & App Controller, external timing source, power supply, RF/EM environment, temperature environment

Potentially Vulnerable to: DoS, Disruption, Distortion, Spoofing, Snooping, live code injection, serial Bus tampering, reconfiguration, firmware analysis, firmware tampering, snooping side channels, environmental tampering, etc.

NET & APP Controller

The subcomponent that primarily focuses on executing any higher layer network functionality. This is generally an independent microprocessor or microcontroller that may also run the application.

Key Subcomponents: Registers, RAM, other storage, boot loader, internal programme storage, internal timing source, and architecture specific functionality

Key External Dependencies: PHY, Link Transceiver, external buses, data bus to the PHY, Link Transceiver, external timing source, power supply, RF/EM environment, temperature environment, external storage

Potentially Vulnerable to: DoS, Disruption, Distortion, Spoofing, Snooping, live code injection, serial Bus Tampering, reconfiguration, flash/RAM snooping, flash/RAM tampering, firmware analysis, firmware tampering, snooping side channels, environmental tampering, tampering of external flash, etc.

Low-Level Document

On PC-class hardware, there are two basic mechanisms for sending rendering commands to the graphics device: PIO/MMIO (see glossary for specific definitions) and DMA. The architecture described in this document is designed around DMA-style hardware, but can easily be extended to accommodate PIO/MMIO-style hardware.

- *Client* is a user-space X11 client which has been linked with various modules to support hardware-dependent direct rendering. Typical modules may include:
  - libGL.so, the standard OpenGL (or Mesa) library with our device and operating system independent acceleration and GLX modifications.
  - libDRI.so, our device-independent, operating-system dependent driver.
  - libHW3D.so, our *device-dependent* driver.

- *X server* is a user-space X server which has been modified with *device and operating-system independent* code to support DRI. It may be linked with other modules to support hardware-dependent direct rendering.

  Typical modules may include:
  - libDRI.so, our *device-independent, operating-system dependent* driver.
  - libH2D.so, our *device-dependent* driver. This library may provide hardware-specific 2D rendering, and 3D initialization and finalization routines that are not required by the client.
- *Kernel Driver* is a kernel-level device driver that performs the bulk of the DMA operations and provides interfaces for synchronization. [Note: Although the driver functionality is hardware-dependent, the actual implementation of the driver may be done in a generic fashion, allowing all of the hardware-specific details to be abstracted into libH3D.so for loading into the Kernel Driver at DRI initialization time. An implementation of this type is desirable since the Kernel Driver will not then have to be updated for each new graphics device. The details of this implementation are discussed in an accompanying document, but are mentioned here to avoid later confusion.]
- PROTO is the standard X protocol transport layer (*e.g.*, a named pipe for a local client).
- SAREA is a special shared-memory area that we will implement as part of the DRI. This area will be used

to communicate information from the X server to the client, and may also be used to share state information with the kernel. This area should not be confused with DMA buffers. This abstraction may be implemented as several different physical areas.

- DMA BUFFERS are memory areas used to buffer graphics device commands which will be sent to the hardware via DMA. These areas are not needed if memory-mapped IO (MMIO) is used exclusively to access the hardware.
- IOCTL is a special interface to the kernel device driver. Requests can be initiated by the user-space programme, and information can be transfered to and from the kernel. This interface incurs the overhead of a system call and memory copy for the information transfered. This abstract interface also includes the ability of the kernel to signal a listening user-space application (*e.g.*, the X server) via I/O on a device (which may, for example, signal the user-space application with the SIGIO signal).
- MMIO is direct memory-mapped access to the graphics device.

Initialization Analysis

The X server is the first application to run that is involved with direct rendering. After initializing its own resources, it starts the kernel device driver and waits for clients to connect. Then, when a direct rendering client connects, SAREA is created, the XFree86-GLX protocol is established, and other direct rendering resources are allocated. This section describes the operations necessary to bring the system to a steady state.

X Server Initialization

When the X server is started, several resources in both the X server and the kernel must be initialized if the GLX module is loaded. Obviously, before the X server can do anything with the 3D graphics device, it will load the GLX module if it is specified in the XFree86 configuration file. When the GLX module (which contains the GLX protocol decoding and event handling routines) is loaded, the device-independent DRI module will also be loaded. The DRI module will then call the graphics device-dependent module (containing both the 2D code and the 3D initialization code) to handle the resource allocation outlined below.

X Resource Allocation Initialization

Several global X resources need to be allocated to handle the client's 3D rendering requests. These resources include the frame buffer, texture memory, other ancillary buffers, display list space, and the SAREA.

Frame 3Buffer

There are several approaches to allocating buffers in the frame buffer: static, static with dynamic reallocation of the unused space, and fully dynamic. Static buffer allocation is the approach we are adopting in the sample implementation for several reasons that will be outlined below.

Static allocation. During initialization, the resources supported by the graphics device are statically allocated. For example, if the device supports front, back and depth buffers in the frame buffer, then the frame buffer is divided into four areas. The first three are equal in size to the visible display area and are used for the three buffers (front, back and

depth). The remaining frame buffer space remains unallocated and can be used for hardware cursor, font and pixmap caches, textures, pbuffers, etc.

Texture memory

Texture memory is shared among all 3D rendering clients. On some types of graphics devices, it can be shared with other buffers, provided that these other buffers can be "kicked out" of the memory. On other devices, there is dedicated texture memory, which might or might not be sharable with other resources. Since memory is a limited resource, it would be best if we could provide a mechanism to limit the memory reserved for textures. However, the format of texture memory on certain graphics devices is organized differently (banked, tiled, etc.) than the simple linear addressing used for most frame buffers. Therefore, the "size" of texture memory is device-dependent. This complicates the issue of using a single number for the size of texture memory.

Another complication is that once the X server reports that a texture will fit in the graphics device memory, it must continue to fit for the life of the client (*i.e.*, the total texture memory for a client can never get smaller). Therefore, at initialization time, the maximum texture size and total texture memory available will need to be determined by the device-dependent driver. This driver will also provide a mechanism to determine if a set of textures will fit into texture memory.

Other Ancillary Buffers

All buffers associated with a window (*e.g.*, back, depth, and GID) are preallocated by the static frame-buffer

allocation. Pixmap, pbuffers and other ancillary buffers are allocated out of the memory left after this static allocation.

During X server initialization, the size off-screen memory available for these buffers will be calculated by the device-dependent driver. Note that pbuffers can be "kicked out" (at least the old style could), and so they don't require virtualization like pixmaps and potentially the new style pbuffers.

Display Lists

For graphics devices that support display lists, the display list memory can be managed in the same way as texture memory. Otherwise, display lists will be held in the client virtual-address space.

SAREA

The SAREA is shared between the clients, the X server, and the kernel. It contains four segments that need to be shared: a per-device global hardware lock, per-context information, per-drawable information, and saved device state information.

- *Hardware lock segment*. Only one process can access the graphics device at a time. For atomic operations that require multiple accesses, a global hardware lock for each graphics device is required. Since the number of cards is known at server initialization time, the size of this segment is fixed.
- *Per-context segment*. Each GLXContext is associated with a particular drawable in the per-drawable segment and a particular graphics device state in the saved device state segment. Two pointers, one to the drawable that the GLXContext is currently bound and one to the saved device state is stored in

the per-context segment. Since the number of GLXContexts is not known at server start up time, the size of this segment will need to grow. It is a reasonable assumption to limit the number of direct rendering contexts so the size of this segment can be fixed to a maximum. The X server is the only process that writes to this segment and it must maintain a list of available context slots that needs to be allocated and initialized.

- *Per-drawable segment*. Each drawable has certain information that needs to be shared between the X server and the direct rendering client:
  - Buffer identification (*e.g.*, front/back buffer) (int32)
  - Window information changed ID
  - Flags (int32)

The window information changed ID signifies that the user has either moved, unmapped or resized the window, or the clipping information has changed and needs to be communicated to the client via the XFree86-GLX protocol.

Since OpenGL clients can create an arbitrary number of GLXDrawables, the size of this segment will need to grow. As with the per-context segment, the size of this segment can be limited to a fixed maximum. Again, the X server is the only process that writes to this segment, and it must maintain a list of available drawable slots that needs to be allocated and initialized.

- *Saved device state segment*. Each GLXContext needs to save the graphics hardware context when another GLXContext has ownership of the graphics device.

This information is fixed in size for each graphics device, but will be allocated as needed because it can be quite large. In addition, if the graphics device can read/write its state information via DMA, this segment will need to be locked down during the request.

Kernel Initialization

When the X server opens the kernel device driver, the kernel loads and initializes the driver. See the next section for more details of the kernel device driver.

Double Buffer Optimizations

There are typically three approaches to hardware double buffering:

1. *Video Page Flipping*: The video logic is updated to refresh from a different page. This can happen very quickly with no per pixel copying required. This forces the entire screen region to be swapped.
2. *Bitblt Double Buffering*: The back buffer is stored in offscreen memory and specific regions of the screen can be swapped by coping data from the offscreen to onscreen. This has a performance penality because of the overhead of copying the swapped data, but allows for fine grain independent control for multiple windows.
2. *Auxillary Per Pixel Control*: An additional layer contains information on a per pixel basis that is used to determine which buffer should be displayed. Swapping entire regions is much quicker than Bitblt Double Buffering and fine grain independed control

for multiple windows is achieved. However, not all hardware or modes support this method.

If the hardware support Auxillary Per Pixel Control for the given mode, then that is the preferred method for double buffer support. However, if the hardware doesn't support Auxillary Per Pixel Control, then the following combined opproach to Video Page Flipping and Bitblt Double Buffering is a potential optimization.

- Initialize in a *Bitblt Double Buffering* mode. This allows for X Server performance to be optimized while not double buffering is required.
- Transition to a *Video Page Flipping* mode for the first window requiring double buffer support. This allows for the fastest possible double buffer swapping at the expense of requiring the X Server to render to both buffers. Note, for the transition, the contents of the front buffer will need to be copied to the back buffer and all further rendering will need to be duplicated in both buffers for all non-double buffered regions while in this mode.
- Transition back to *Bitblt Double Buffering* mode when additional double buffering windows are created. This will sacrifice performance for the sake of visual accuracy. Now all windows can be independently swapped.

In the initial SI, only the Bitblt Double Buffering mode will be implemented.

Kernel Driver Initialization

When the kernel device driver is opened by the X server, the device driver might not be loaded. If not, the module is

loaded by kerneld and the initialization routine is called. In either case, the open routine is then called and finishes initializing the driver.

Kernel DMA Initialization

Since the 3D graphics device drivers use DMA to communicate with the graphics device, we need to initialize the kernel device driver that will handle these requests. The kernel, in response to this request from the X server, allocates the DMA buffers that will be made available to direct rendering clients.

Kernel Interrupt Handling Initialization

Interrupts are generated in a number of situations including when a DMA buffer has been processed by the graphics device. To acknowledge the interrupt, the driver must know which register to set and to what value to set it. This information could be hard coded into the driver, or possibly a generic interface might be able to be written. If this is possible, the X server must provide information to the kernel as to how to respond to interrupts from the graphics device.

Hardware Context Switching

Since the kernel device driver must be able to handle multiple 3D clients each with a different GLXContext, there must be a way to save and restore the hardware graphics context for each GLXContext when switching between them. Space for these contexts will need to be allocated when they are created byglXCreateContext(). If the client can use this hardware context (*e.g.*, for software fallbacks or window moves), this information might be stored in the SAREA.

Client DMA wait Queues

Each direct rendering context will require a DMA wait queue from which its DMA buffers can be dispatched. These wait queues are allocated by the X server when a new GLXContext is created (glXCreateContext()).

Client Initialization

This section examines what happens before the client enters steady state behaviour. The basic sequence for direct-rendering client initialization is that the GL/GLX library is loaded, queries to the X server are made (*e.g.*, to determine the visuals/FBConfigs available and if direct rendering can be used), drawables and GLXContexts are created, and finally a GLXContext is associated with a drawable. This sequence assumes that the X server has already initialized the kernel device driver and has pre-allocated any static buffers requested by the user at server startup (as described above).

Library Loading

When a client is loaded, the GL/GLX library will automatically be loaded by the operating system, but the graphics device-specific module cannot be loaded until after the X server has informed the DRI module which driver to load (see below). The DRI module might not be loaded until after a direct rendering GLXContext has been requested.

Client Configuration Queries

During client initialization code, several configuration queries are commonly made. GLX has queries for its version number and a list of supported extensions. These requests are made through the standard GLX protocol stream. Since

the set of supported extensions is device-dependent, similar queries in the device-dependent driver interface (in the X server) are provided that can be called by device-independent code in GLX.

One of the required GLX queries from the client is for the list of supported extended visuals (and FBConfigs in GLX 1.3). The visuals define the types of colour and ancillary buffers that are available and are device-dependent. The X server must provide the list of supported visuals (and FBConfigs) via the standard protocol transport layer (*e.g.*, Unix domain or TCP/IP sockets). Again, similar interfaces in the device-dependent driver are provided that can be called by the device-independent code in GLX. All of this information is known at server initialization time (above).

Drawable creation

The client chooses the visual (or FBConfig) it needs and creates a drawable using the selected visual. If the drawable is a window, then, since we use a static resource allocation approach, the buffers are already allocated, and no additional frame buffer allocations are necessary at this time. However, if a dynamic resource allocation approach is added in the future, the buffers requested will need to be allocated.

Not all buffers need to be pre-allocated. For example, accumulation buffers can be emulated in software and might not be pre-allocated. If they are not, then, when the extended visual or FBConfig is associated with the drawable, the client library will need to allocate the accumulation buffer. In GLX 1.3, this can happen withglXCreateWindow(). For earlier versions of GLX, this will happen when a context is made current (below).

Pixmaps and Buffers

GLXPixmaps are created from an ordinary X11 pixmap, which is then passed to glXCreatePixmap(). GLXPbuffers are created directly by a GLX command. Since we are using a static allocation scheme, we know what ancillary buffers need to be created for these drawables. In the initial SI, these will be handled by indirect rendering or software fallbacks.

GLXContext creation

The client must also create at least one GLXContext. The last flag to glXCreateContext() is a flag to request direct rendering. The first GLXContext created can trigger the library to initialize the direct rendering interface for this client. Several steps are required to setup the DRI. First, the DRI library is loaded and initialized in the client and X server. The DRI library establishes the private communication mechanism between the client and X server (the XFree86-GLX protocol). The X server sends the SAREA shared memory segment ID to the client via this protocol and the client attaches to it. Next, the X server sends the device-dependent client side 3D graphics device driver module name to client via the XFree86-GLX protocol, which is loaded and initialized in the client.

The X server calls the kernel module to create a new WaitQueue and hardware graphics context corresponding to the new GLXContext. Finally, the client opens and initializes the kernel driver (including a request for DMA buffers).

Making a GLXContext current

The last stage before entering the steady state behaviour occurs when a GLXContext is associated with a GLXDrawable

by making the context "current". This must occur before any 3D rendering can begin. The first time a GLXDrawable is bound to a direct rendering GLXContext it is registered with the X server and any buffers not already allocated are now allocated. If the GLXDrawable is a window that has not been mapped yet, then the buffers associated with the window are initialized to size zero. When a window is mapped, space in the pre-allocated static buffers are initialized, or in the case of dynamic allocation, buffers are allocated from the available offscreen area (if possible).

For GLX 1.2 (and older versions), some ancillary buffers (*e.g.*, stencil or accumulation), that are not supported by the graphics device, or unavailable due to either resource constraints or their being turned off through X server config options (see above), might need to be allocated.

At this point, the client can enter the steady-state by making OpenGL calls.

Steady-state Analysis

The initial steady-state analysis presented here assumes that the client(s) and X server have been started and have established all necessary communication channels (*e.g.*, the X, GLX and XFree86-GLX protocol streams and the SAREA segment). In the following analysis, we will impose simplifying assumptions to help direct the analysis towards the main line rendering case. We will then relax our initial assumptions and describe increasingly general cases.

Single 3D Client (1 GLXContext, 1 GLXWindow), X Server Inactive

Assume: No X server activity (including hardware cursor

movement). This is the optimized main line rendering case. The primary goal is to generate graphics device specific commands and stuff them in a DMA buffer as fast as possible. Since the X server is completely inactive, any overhead due to locking should be minimized.

Processing rendering requests

In the simplest case, rendering commands can be sent to the graphics device by putting them in a DMA buffer. Once a DMA buffer is full and needs to be dispatched to the graphics device, the buffer can be handed immediately to the kernel via an ioctl.

The kernel then schedules the DMA command buffer to be sent to the graphics device. If the graphics device is not busy (or the DMA input queue is not full), it can be immediately sent to the graphics device. Otherwise, it is put on the WaitQueue for the current context.

In hardware that can only process a single DMA buffer at a time, when the DMA buffer has finished processing, an IRQ is generated by the graphics device and handled by the kernel driver.

In hardware that has a DMA input FIFO, IRQs can be generated after each buffer, after the input FIFO is empty or (in certain hardware) when a low-water mark has been reached. For both types of hardware, the kernel device driver resets the IRQ and schedules the next DMA buffer(s).

A further optimization for graphics devices that have input FIFOs for DMA requests is that if the FIFO is not full, the DMA request could be initiated directly from client space.

Synchronization

GLX has commands to synchronize direct rendering with indirect rendering or with ordinary X11 operations. These include glFlush(), glFinish(), glXWaitGL() and glXWaitX() synchronization primitives. The kernel driver provides several ioctls to handle each of the synchronization cases. In the simplest case (glFlush()), any partially filled DMA buffer will be sent to the kernel.

Since these will eventually be processed by the hardware, the function call can return. WithglFinish(), in addition to sending any partially filled DMA buffer to the kernel, the kernel will block the client process until all outstanding DMA requests have been completely processed by the graphics device. glXWaitGL() can be implemented using glFlush(), glXWaitX() can be implemented with XSync().

Buffer Swaps

Buffers swaps can be initiated by glXSwapBuffers(). When a client issues this request, any partially filled DMA buffers are sent to the kernel and all outstanding DMA buffers are processed before the buffer swap can take place.

All subsequent rendering commands are blocked until the buffer has been swapped, but the client is not blocked and can continue to fill DMA buffers and send them to the kernel.

If multiple threads are rendering to a GLXDrawable, it is the client's responsibility to synchronize the threads. In addition, the idea of the *current* buffer (*e.g.*, front or back) must be shared by all GLXContexts bound to a given drawable. The X double buffer extension must also agree.

Kernel-driver Buffer Swap Ioctl

When the buffer swap ioctl is called, a special DMA buffer with the swap command is placed into the current GLXContext's WaitQueue. Because of sequentiality of the DMA buffers in the WaitQueue, all DMA buffers behind this are blocked until all DMA buffers in front of this one have been processed. The header information associated with this buffer lets the scheduler know how to handle the request.

There are three ways to handle the buffer swap:

1. *No vert sync*: Immediately schedule the buffer swap and allow subsequent DMA buffers in the WaitQueue to be scheduled. With this policy there will be tearing. In the initial SI, we will implement this policy.
2. *Wait for vert sync*: Wait for the vertical retrace IRQ to schedule the buffer swap command and allow subsequent DMA buffers in the WaitQueue to be scheduled. With this policy, the tearing should be reduced, but there might still be some tearing if a DMA input FIFO is present and relatively full.
3. *No tearing*: Wait for vertical retrace IRQ and all DMA buffers in the input FIFO to be processed before scheduling the buffer swap command. Since the buffer swap is a very fast bitblt operation, no tearing should be present with this policy.

Software Fallbacks

Not all OpenGL graphics primitives are accelerated in all hardware. For those not supported directly by the graphics device, software fallbacks will be required. Mesa and SGI's OpenGL SI provide a mechanism to implement these

fallbacks; however, the hardware graphics context state needs to be translated into the format required by these libraries. The hardware graphics context state can be read from the saved device state segment of SAREA. An implicit glFinish() is issued before the software fallback can be initiated to ensure that the graphics state is up to date before beginning the software fallback. The hardware lock is required to alter any device state.

Image Transfer Operations

Many image transfer operations are required in the client-side direct rendering library. Initially these will be software routines that read directly from the memory mapped graphics device buffers (*e.g.*, frame buffer and texture buffer). These are device-dependent operations since the format of the transfer might be different, though certain abstractions should be possible (*e.g.*, linear buffers). An optimization is to allow the client to perform DMA directly to/from the client's address space. Some hardware has support for page table translation and paging. Other hardware will require the ability to lock down pages and have them placed contiguously in physical memory.

The X server will need to manage how the frame and other buffers are allocated at the highest level. The layout of these buffers is determined at X server initialization time.

Texture Management

Each GLXContext appears to own the texture memory. In the present case, there is no contention. In subsequent cases, hardware context switching will take care of texture swapping as well (see below).

For a single context, the image transfer operations described above provides the necessary interfaces to transfer textures and subtextures to/from texture memory.

Display List Management

Display lists initially will be handled from within the client's virtual address space. For graphics devices that supports display lists, they can be stored and managed the same as texture memory.

Selection and Feedback

If there is hardware support for selection and feedback, the rendering commands are sent to the graphics pipeline, which returns the requested data to the client. The amount of data can be quite large and are usually delivered to a collection of locked-down pages via DMA. The kernel should provide a mechanism for locking down pages in the client address space to hold the DMA buffer.

Queries

Queries are handled similarly to selection and feedback, but the data returned are usually much smaller. When a query is made, the hardware graphics context state has to be read. If the GLXContext does not currently own the graphics device, the state can be read from the saved device state segment in SAREA. Otherwise, the graphics pipeline is temporarily stalled, so that the state can be read from the graphics device.

Events

GLX has a "pbuffer clobbered" event. This can only be generated as a result of reconfiguring a drawable or creating

a new one. Since pbuffers will initially be handled by the software, no clobbered events will be generated. However, when they are accelerated, the X server will have to wrap the appropriate routine to determine when the event needs to be generated.

Single 3D Client (1 GLXContext, 1 GLXWindow), X Server can Draw

Assume: X server can draw (*e.g.*, 2D rendering) into other windows, but does not move the 3D window. This is a common case and should be optimized if possible. The only significant different between this case and the previous case, is that we must now lock the hardware before accessing the graphics device directly directly from the client, X server or kernel space.

The goal is to minimize state transitions and potentially avoid a full hardware graphics context switch by allowing the X server to save and restore 3D state around its access for GUI acceleration.

Hardware Lock

Access to graphics device must be locked, either implicitly or explicitly. Each component of the system requires the hardware lock at some point. For the X server, the hardware lock is required when drawing or modifying any state. It is requested around blocks of 2D rendering, minimizing the potential graphics hardware context switches.

In the 3D client, the hardware lock is required during the software fallbacks (all other graphics device accesses are handled through DMA buffers). The kernel also must request the lock when it needs to send DMA requests to the graphics

device. The hardware lock is contained in the *Hardware lock segment* of the SAREA which can be accessed by all system components. A two-tiered locking scheme is used to minimize the process and kernel context switches necessary to grant the lock. The most common case, where a lock is requested by the last process to hold the lock, does not require any context switches. See the accompanying locks.txt file for more information on two-tiered locking (available late February 1999).

Graphics Hardware Context Switching

In addition to locking the graphics device, a graphics hardware context switch between the client and the X server is required. One possible solution is to perform a full context switch by the kernel (see the "multiple contexts" section below for a full explanation of how a full graphics hardware context switch is handled). However, the X server is a special case since it knows exactly when a context switch is required and what state needs to be saved and restored.

For the X server, the graphics hardware context switch is required only (a) when directly accessing the graphics device and (b) when the access changes the state of the graphics device. When this occurs, the X server can save the graphics device state (either via a DMA request or by reading the registers directly) before it performs its rendering commands and restore the graphics device state after it finishes.

Three examples will help clarify the situations where this type of optimization can be useful. First, using a cfb/mi routine to draw a line only accesses the frame buffer and does not alter any graphics device state. Second, on many vendor's cards changing the position of the hardware cursor

does not affect the graphics device state. Third, certain graphics devices have two completely separate pipelines for 2D and 3D commands. If no 2D and 3D state is shared, then they can proceed independently (but usually not simultaneously, so the hardware lock is still required).

6

System Development Model and Software Engineering

It specifies the relationships between project phases, including transition criteria, feedback mechanisms, milestones, baselines, reviews, and deliverables. Typically, a life cycle model addresses the phases of a software project: requirements phase, design phase, implementation, integration, testing, operations and maintenance. Much of the motivation behind utilizing a life cycle model is to provide structure to avoid the problems of the "undisciplined hacker" or corporate IT bureaucrat (which is probably ten times dangerous then undisciplined hacker). As always, it's a matter of picking the right tool for the job, rather than picking up your hammer and treating everything as a nail.

Component Assembly Model

Object technologies provide the technical framework for a component-based process model for software engineering.

The object oriented paradigm emphasizes the creation of classes that encapsulate both data and the algorithm that are used to manipulate the data. If properly designed and implemented, object oriented classes are reusable across different applicationsand computer based system architectures.

Component Assembly Model leads to software reusability. The integration/assembly of the already existing software components accelerate the development process. Nowadays many component libraries are available on the Internet. If the right components are chosen, the integration aspect is made much simpler.

All these different software development models have their own advantages and disadvantages. Nevertheless, in the contemporary commercial software evelopment world, the fusion of all these methodologies is incorporated. Timing is very crucial in software development. If a delay happens in the development phase, the market could be taken over by the competitor. Also if a 'bug' filled product is launched in a short period of time (quicker than the competitors), it may affect the reputation of the company. So, there should be a tradeoff between the development time and the quality of the product. Customers don't expect a bug free product but they expect a user-friendly product.

System/Information Engineering and Modeling

As software is always of a large system (or business), work begins by establishing the requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when

the software must interface with other elements such as hardware, people and other resources.

System is the basic and very critical requirement for the existence of software in any entity. So if the system is not in place, the system should be engineered and put in place. In some cases, to extract the maximum output, the system should be re-engineered and spruced up. Once the ideal system is engineered or tuned, the development team studies the software requirement for the system.

Software Requirement Analysis

This process is also known as feasibility study. In this phase, the development team visits the customer and studies their system. They investigate the need for possible software automation in the given system. By the end of the feasibility study, the team furnishes a document that holds the different specific recommendations for the candidate system. It also includes the personnel assignments, costs, project schedule, target dates etc....

The requirement gathering process is intensified and focussed specially on software. To understand the nature of the programme(s) to be built, the system engineer or "Analyst" must understand the information domain for the software, as well as required function, behaviour, performance and interfacing. The essential purpose of this phase is to find the need and to define the problem that needs to be solved.

System Analysis and Design

In this phase, the software development process, the software's overall structure and its nuances are defined. In terms of the client/server technology, the number of tiers

needed for the package architecture, the database design, the data structure design etc... are all defined in this phase.

A software development model is thus created. Analysis and Design are very crucial in the whole development cycle. Any glitch in the design phase could be very expensive to solve in the later stage of the software development. Much care is taken during this phase. The logical system of the product is developed in this phase.

Code Generation

The code generation step performs this task. If the design is performed in a detailed manner, code generation can be accomplished without much complication. Progra-mming tools like compilers, interpreters, debuggers etc... are used to generate the code. Different high level programming languages like C, C++, Pascal, Java are used for coding. With respect to the type of application, the right programming language is chosen.

Testing

Once the code is generated, the software programme testing begins. Different testing methodologies are available to unravel the bugs that were committed during the previous phases. Different testing tools and methodologies are already available. Some companies build their own testing tools that are tailor made for their own development operations.

Maintenance

The software will definitely undergo change once it is delivered to the customer. There can be many reasons for this change to occur. Change could happen because of some

unexpected input values into the system. In addition, the changes in the system could directly affect the software operations. The software should be developed to accommodate changes that could happen during the post implementation period.

Prototyping Model

This is a cyclic version of the linear model. In this model, once the requirement analysis is done and the design for a prototype is made, the development process gets started. Once the prototype is created, it is given to the customer for evaluation. The customer tests the package and gives his/her feed back to the developer who refines the product according to the customer's exact expectation. After a finite number of iterations, the final software package is given to the customer.

In this methodology, the software is evolved as a result of periodic shuttling of information between the customer and developer. This is the most popular development model in the contemporary IT industry. Most of the successful software products have been developed using this model - as it is very difficult (even for a whiz kid!) to comprehend all the requirements of a customer in one shot. There are many variations of this model skewed with respect to the project management styles of the companies. New versions of a software product evolve as a result of prototyping.

The goal of prototyping based development is to counter the first two limitations of the waterfall model discussed earlier. The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements.

This prototype is developed based on the currently known requirements. Development of the prototype obviously undergoes design, coding and testing.

But each of these phases is not done very formally or thoroughly. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements.

The basic reason for little common use of prototyping is the cost involved in this built-it-twice approach. However, some argue that prototyping need not be very costly and can actually reduce the overall development cost. The prototype are usually not complete systems and many of the details are not built in the prototype.

The goal is to provide a system with overall functionality. In addition, the cost of testing and writing detailed documents are reduced. These factors helps to reduce the cost of developing the prototype. On the other hand, the experience of developing the prototype will very useful for developers when developing the final system. This experience helps to reduce the cost of development of the final system and results in a more reliable and better designed system.

Advantages of Prototyping

Creating software using the prototype model also has its benefits. One of the key advantages a prototype modeled software has is the time frame of development. Instead of concentrating on documentation, more effort is placed in

creating the actual software. This way, the actual software could be released in advance. The work on prototype models could also be spread to others since there are practically no stages of work in thismodel. Everyone has to work on the same thing and at the same time, reducing man hours in creating a software. The work will even be faster and efficient if developers will collaborate more regarding the status of a specific function and develop the necessary adjustments in time for the integration.

Another advantage of having a prototype modeled software is that the software is created using lots of user feedbacks. In every prototype created, users could give their honest opinion about the software. If something is unfavorable, it can be changed. Slowly the programme is created with the customer in mind.

- Users are actively involved in the development
- It provides a better system to users, as users have natural tendency to change their mind in specifying requirements and this method of developing systems supports this user tendency.
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier as the system is mode side by side.
- Quicker user feedback is available leading to better solutions.

Disadvantages

Implementing the prototype model for creating software has disadvantages. Since its being built out of concept, most

of the models presented in the early stage are not complete. Usually they lack flaws that developers still need to work on them again and again. Since the prototype changes from time to time, it's a nightmare to create a document for this software. There are many things that are removed, changed and added in a single update of the prototype and documenting each of them has been proven difficult.

There is also a great temptation for most developers to create a prototype and stick to it even though it has flaws. Since prototypes are not yet complete software programmes, there is always a possibility of a designer flaw. When flawed software is implemented, it could mean losses of important resources.

Lastly, integration could be very difficult for a prototype model. This often happens when other programmes are already stable. The prototype software is released and integrated to the company's suite of software. But if there's something wrong the prototype, changes are required not only with the software. It's also possible that the stable software should be changed in order for them to be integrated properly.

Prototype Models Types

There are four types of Prototype Models based on their development planning: the Patch-Up Prototype, Nonoperational Prototype, First-of-a-Series Prototype and Selected Features Prototype.

Patch Up Prototype

This type of Prototype Model encourages cooperation of different developers. Each developer will work on a specific

part of the programme. After everyone has done their part, the programme will be integrated with each other resulting in a whole new programme. Since everyone is working on a different field, Patch Up Prototype is a fast development model. If each developer is highly skilled, there is no need to overlap in a specific function of work. This type of software development model only needs a strong project manager who can monitor the development of the programme. The manager will control the work flow and ensure there is no overlapping of functions among different developers.

Non-Operational Prototype

A non-operational prototype model is used when only a certain part of the programme should be updated. Although it's not a fully operational programme, the specific part of the programme will work or could be tested as planned. The main software or prototype is not affected at all as the dummy programme is applied with the application. Each developer who is assigned with different stages will have to work with the dummy prototype.

This prototype is usually implemented when certain problems in a specific part of the programme arises. Since the software could be in a prototype mode for a very long time, changing and maintenance of specific parts is very important. Slowly it has become a smart way of creating software by introducing small functions of the software.

First of a Series Prototype

Known as a beta version, this Prototype Model could be very efficient if properly launched. In all beta versions, the

software is launched and even introduced to the public for testing. It's fully functional software but the aim of being in beta version is to as for feedbacks, suggestions or even practicing the firewall and security of the software.

It could be very successful if the First of a Series Prototype is properly done. But if the programme is half heartedly done, only aiming for additional concept, it will be susceptible to different hacks, ultimately backfiring and destroying the prototype.

Selected Features Prototype

This is another form of releasing software in beta version. However, instead of giving the public the full version of the software in beta, only selected features or limited access to some important tools in the programme is introduced. Selected Features Prototype is applied to software that are part of a bigger suite of programmes. Those released are independent of the suite but the full version should integrate with other software. This is usually done to test the independent feature of the software.

Rapid Application Development

The RAD model is a "high speed" adaptation of the linear sequential model in which rapid development is achieved by using a component-based construction approach. Used primarily for information systems applications, the RAD approach encompasses the following phases:

Business Modeling

The information flow among business functions is modeled in a way that answers the following questions:

- What information drives the business process?
- What information is generated?
- Who generates it?
- Where does the information go?
- Who processes it?

Data Modeling

The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristic (called attributes) of each object is identified and the relationships between these objects are defined.

Process Modeling

The data objects defined in the data-modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing the descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application Generation

The RAD model assumes the use of the RAD tools like VB, VC++, Delphi etc... rather than creating software using conventional third generation programming languages. The RAD model works to reuse existing programme components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and Turnover

Since the RAD process emphasizes reuse, many of the programme components have already been tested. This minimizes the testing and development time.

System and Specification

Important issues are not defined up front and Mechanical, Electronic and Software designers do not really know what their requirements are:

- Define the functions of the system
- Define the Hardware/ Software Functional Partitioning
- Define the Performance Specification
- Define the Hardware/ Software Performance Partitioning
- Define Safety Requirements
- Define the User Interface (A good user's manual is often an overlooked part of the System specification. Many of our customers haven't even considered that this is the right time to write the user's manual.)
- Provide Installation Drawings/Instructions.
- Provide Interface Control Drawings (ICD's, External I/O)

One job of the System specification is to define the full functionality of the system. In many systems we work on, some functionality is performed in hardware and some in software. It is the job of the System specification to define the full functionality and like the performance requirements, to set in motion the trade-offs and preliminary design studies to allocate these functions to the different disciplines (mechanical, electrical, software).

Another function of the System specification is to specify performance. For example, if the System is required to move a mechanism to a particular position accurate to a repeatability of ± 1 millimeter, that is a System's requirement.

Some portion of that repeatability specification will belong to the mechanical hardware, some to the servo amplifier and electronics and some to the software. It is the job of the System specification to provide that requirement and to set in motion the partitioning between mechanical hardware, electronics, and software.

Very often the System specification will leave this partitioning until later when you learn more about the system and certain factors are traded off (For example, if we do this in software we would need to run the processor clock at 40 mHz.

However, if we did this function in hardware, we could run the processor clock at 12 mHz). However, for all practical purposes, most of the systems we are involved with in small to medium size companies, combine the software and the systems documents.

This is done primarily because most of the complexity is in the software. When the hardware is used to meet a functional requirement, it often is something that the software wants to be well documented.

Very often, the software is called upon to meet the system requirement with the hardware you have. Very often, there is not a systems department to drive the project and the software engineers become the systems engineers. For small projects, this is workable even if not ideal. In this case, the specification should make clear which requirements are software, which are hardware, and which are mechanical.

Design and Requirement

SRS should not include any design requirements. However, this is a difficult discipline. For example, because of the

partitioning and the particular RTOS you are using, and the particular hardware you are using, you may require that no task use more than 1 ms of processing prior to releasing control back to the RTOS.

Although that may be a true requirement and it involves software and should be tested – it is truly a design requirement and should be included in the Software Design Document or in the Source code. Consider the target audience for each specification to identify what goes into what documents.

Marketing/Product Management

Creates a product specification and gives it to Systems. It should define everything Systems needs to specify the product

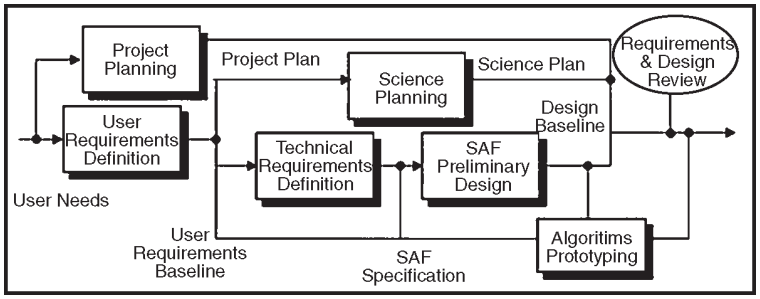
Systems/Software

Creates a Software Specification and gives it to Software. It should define everything Software needs to develop the software. Thus, the SRS should define everything explicitly or (preferably) by reference that software needs to develop the software.

References should include the version number of the target document. Also, consider using master document tools which allow you to include other documents and easily access the full requirements.

Requirement Engineering Process

Based on assessed user needs, the SAF User Requirements are established and implemented into a Technical Specification and Design baseline, in line with scientific assessments and plans.



Software requirements engineering is the process of determining what is to be produced in a software system. In developing a complex software system, the requirements engineering process has the widely recognized goal of determining the needs for, and the intended external behaviour, of a system design.

This process is regarded as one of the most important parts of building a software system: " The hardest single part of building a software system is deciding what to build. No other part of the conceptual work is as difficult a establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.

Tracing the emergence of significant ideas in software development over the years, one can observe that in the '60s the attention was on coding, in the '70s on design and in the '80s on specification. However, in the process of requirements engineering it is often difficult to state the real 'what' level of a system because one person's 'how' may be another person's 'what' and conversely. In this perspective, the requirements engineer faces a complex problem, in meeting the needs of the customer and at the same time meeting the needs of the designer.

The four specific steps in software requirements engineering are:

1. Requirements elicitation
2. Requirements analysis
3. Requirements specification
4. Requirements validation

Although they seem to be separate tasks, these four processes cannot be strictly separated and performed sequentially. Some of the requirements are implicit in the working practices, while others may only arise when design solutions are proposed.

Inquiry based requirements

The Inquiry-Based Requirements Analysis Model views the analysis process as essentially inquiry-based "a series of questions and answers designed to pinpoint where information needs come from and when". The Inquiry Cycle Model, a "formal structure for describing discussion about requirements", addresses the case of mass-market-driven product development, for which there may be no clear customer authority.

The term used in this model is "stakeholder", anyone who shares information about the system, its implementation constraints or problem domain. The model consists of an integration of three phases, where the stakeholders write down their proposed requirements, challenge them by attaching typed annotations and then refine the requirements when change requests are approved.

These requirements are derived from many sources and in many formats, hence a tremendous amount of complex raw data comprise the source material for a given system."AMORE is interested in modeling those vast amounts

of raw source material as requirements, and provides access to knowledge about the problem domain, as well as tools for the capture, modeling, analysis and manipulation of raw requirements data.

User-development interaction

The most important aspect of user-development interaction is the mutual learning and cooperation among them. Some methodologies assume that the transfer of knowledge between users and designers can be achieved in the environment of a meeting room. At the same time, other methodologies (*e.g.* PD) foster the full collaboration of stakeholders through a process where users are directly faced with the designers' work situation and conversely, and by the end of the elicitation process everyone learned about real needs of users and technical possibilities.

In this context, success in meeting the real needs of the software system is contingent upon the ability of users to clearly specify what their requirements are. For this reason, requirements definition needs close interaction between developers and end-users of the software.

It is critical that requirements engineering tools must support collaborative development of the software requirements negotiation. Requirements definition should be an iterative process where, through reflection and experience, users become familiar with the technology and developers become familiar with the work. For example, scenarios, prototypes or mock-ups which provide the opportunity for the users to "experience" the new technology and for the developers to "experience" the work practice.

Team Rooms

The groupware system called Tearooms provides an electronic equivalent of a team room for groups that are either co-located or at a distance. More about Tearooms as a Group Kit application may be found in Roseman and Greenberg. It is implemented using an extended version of the groupware toolkit GroupKit. Facilities offered by the GroupKit's Application Programming Interface are preserved in Team Rooms. This enabled the developers to move the existing GroupKit applications to TeamRooms and rapidly create new ones. It combines the rich applications and interfaces found in the existing real-time groupware applications, providing a persistent work space suitable for both synchronous and asynchronous collaboration. It encapsulates both structured and unstructured work through its applications and also takes into account individual and group work. Apples are special-purpose applications, designated for more specific needs of a group. Team Rooms supports any type of application which can be constructed in Group Kit, such as meeting tools, drawing tools, text editors, card games and so on. When a user starts up the system, he or she is prompted for a user name and a password. If he is among the work group permitted to use the system, he or she will be connected to the Team Rooms central server.

Elicitation

Using an elicitation method can help in producing a consistent and complete set of security requirements. However, brainstorming and elicitation methods used for ordinary functional (end-user) requirements usually are not oriented towards security requirements and do not result in

a consistent and complete set of security requirements. The resulting system is likely to have fewer security exposures when security requirements are elicited in a systematic way.

Elicitation Evaluation Criteria

The following are example evaluation criteria that may be useful in selecting an elicitation method, but certainly there are other criteria that you could use. The main point is to use criteria and to have a common understanding of what they mean.

- *Adaptability*: The method can be used to generate requirements in multiple environments. For example, the elicitation method works equally as well with a software product that is near completion as with a project in the planning stages.
- *Computer-aided software engineering (CASE) tool*: The method includes a CASE tool. (The Software Engineering Institute defines a CASE tool as "a computer-based product aimed at supporting one or more software engineering activities within a software development process")
- *Stakeholder acceptance*: The stakeholders are likely to agree to the elicitation method in analyzing their requirements. For example, the method isn't too invasive in a business environment.
- *Easy implementation*: The elicitation method isn't overly complex and can be properly executed easily.
- *Graphical output*: The method produces readily understandable visual artifacts.
- *Quick implementation*: The requirements engineers and stakeholders can fully execute the elicitation method in a reasonable length of time.

- *Shallow learning curve*: The requirements engineers and stakeholders can fully comprehend the elicitation method within a reasonable length of time.

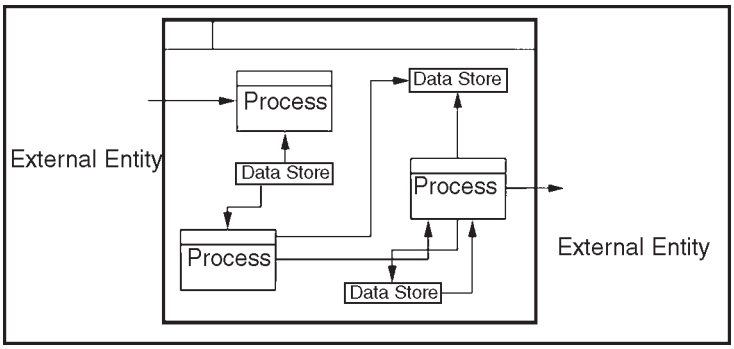
Data Flow Diagrams

Introduction

A data flow diagram (DFD) is a significant modeling technique for analyzing and constructing information processes. DFD literally means an illustration that explains the course or movement of information in a process. DFD illustrates this flow of information in a process based on the inputs and outputs. A DFD can be referred to as a Process Model. Additionally, a DFD can be utilized to visualize data processing or a structured design. A DFD illustrates technical or business processes with the help of the external data stored, the data flowing from a process to another, and the results. A designer usually draws a context-level DFD showing the relationship between the entities inside and outside of a system as one single step. This basic DFD can be then disintegrated to a lower level diagram demonstrating smaller steps exhibiting details of the system that is being modeled.

Uses

The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out to precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.



The result is a series of diagrams that represent the business activities in a way that is clear and easy to communicate. A business model comprises one or more data flow diagrams (also known as business process diagrams). Initially a context diagram is drawn, which is a simple representation of the entire system under investigation. This is followed by a level 1 diagram; which provides an overview of the major functional areas of the business. Don't worry about the symbols at this stage, these are explained shortly. Using the context diagram together with additional information from the area of interest, the level 1 diagram can then be drawn.

The level 1 diagram identifies the major business processes at a high level and any of these processes can then be analysed further - giving rise to a corresponding level 2 business process diagram. This process of more detailed analysis can then continue ýÿC through level 3, 4 and so on. However, most investigations will stop at level 2 and it is very unusual to go beyond a level 3 diagram.

Identifying the existing business processes, using a technique like data flow diagrams, is an essential precursor to business process re-engineering, migration to new technology, or refinement of an existing business process.

However, the level of detail required will depend on the type of change being considered.

The process model is typically used in structured analysis and design methods. Also called a data flow diagram (DFD), it shows the flow of information through a system. Each process transforms inputs into outputs.

process

The process shows a part of the system that transforms inputs into outputs; that is, it shows how one or more inputs are changed into outputs. Some systems analysts prefer to use an oval or a rectangle with rounded edges, as shown in Figure below; still others prefer to use a rectangle, as shown in Figure. The differences between these three shapes are purely cosmetic, though it is obviously important to use the same shape consistently to represent all the functions in the system. Throughout the rest of this book, we will use the circle or bubble.

In some cases, the process will contain the name of a person or a group of people (*e.g.*, a department or a division of an organization), or a computer, or a mechanical device. That is, the process sometimes describes who or what is carrying out the process, rather than describing what the process is.

The Flow

A flow is represented graphically by an arrow into or out of a process. The flow is used to describe the movement of chunks, or packets of information from one part of the system to another part. For most of the systems that you model as a systems analyst, the flows will indeed represent data, that

is, bits, characters, messages, floating point numbers, and the various other kinds of information that computers can deal with. But DFDs can also be used to model systems other than automated, computerized systems; we may choose, for example, to use a DFD to model an assembly line in which there are no computerized components.

In such a case, the packets or chunks carried by the flows will typically be physical materials; an example is shown in Figure below. For many complex, real-world systems, the DFD will show the flow of materials and data.

The flows in Figures are *named*. The name represents the meaning of the packet that moves along the flow. A corollary of this is that the flow carries only one type of packet, as indicated by the flow name. The systems analyst should not name a dataflow Apples and organes and widgets and various other thing.

However, we will see in Part III, that there are exceptions to this convention: it is sometimes useful to consolidate several elementary dataflow into a consolidated flow. Thus, one might see a single dataflow labeled vegetables instead of several different dataflow labeled potatoes, brussel sprouts, and peas. whether data (or material) are moving into or out of a process (or doing both). The flow shown in Figure, for example, clearly shows that a telephone number is being sent into the process labeled Validate Phone Number.

Dataflows can diverge and converge in a DFD; conceptually, this is somewhat like a major river splitting into smaller tributaries, or tributaries joining together. However, this has a special meaning in a typical DFD in which packets of data are moving through the system: in the case of a diverging

flow, it means that duplicate copies of a packet of data are being sent to different parts of the system, or that a complex packet of data is being split into several more elementary data packets, each of which is being sent to different parts of the system, or that the dataflow pipeline carries items with different values (*e.g.*, vegetables whose values may be "potato," "brussel sprout," or "lima bean") that are being separated. Conversely, in the case of a converging flow, it means that several elementary packets of data are joining together to form more complex, aggregate packets of data.

The Terminator

The next component of the DFD is a *terminator*; it is graphically represented as a rectangle, as shown in Figure. Terminators represent external entities with which the system communicates. Typically, a terminator is a person or a group of people, for example, an outside organization or government agency, or a group or department that is *within* the same company or organization, but *outside*the control of the system being modeled.

In some cases, a terminator may be another system, for example, some other computer system with which your system will communicate.

Fig. Graphical Representation of a Terminator.

There are three important things that we must remember about terminators:

1. They are *outside* the system we are modeling; the flows connecting the terminators to various processes

(or stores) in our system represent the interface between our system and the outside world.

2. As a consequence, it is evident that neither the systems analyst nor the systems designer are in a position to change the contents of a terminator or the way the terminator works. In the language of several classic textbooks on structured analysis, the terminator is outside the domain of change. What this means is that the systems analyst is modeling a system with the intention of allowing the systems designer a considerable amount of flexibility and freedom to choose the best (or most efficient, or most reliable, etc.) implementation possible.
3. Any relationship that exists *between* terminators will not be shown in the DFD model. There may indeed be several such relationships, but, by definition, those relationships are not part of the system we are studying. Conversely, if there *are*relationships between the terminators, and if it is essential for the systems analyst to model those requirements in order to properly document the requirements of the system, then, by definition, the terminators are actually part of the system and should be modeled as processes.

Entity Relationship Diagrams

Introduction

An entity can be thought of as a class of data. Each entity has a name, a definition, a type. In addition, each entity has a set of attributes that describe the various characteristics

of the entity. Each attribute also has a name, a definition, a type and constraints. The attribute types are text, numeric, binary and date types. Field and attributes are different name for the same thing. Entity and table are different name for the same thing. In the context of relationship diagrams, the words entity and attributes are used. In the context of physical database work, the words table and field are used.

An Entity-Relationship (E-R) Diagram (or E-R Model) visually depicts an organization's entities, the entities' relationships to each other, and the business rules (*i.e.*, cardinality and dependency) associated with the relationships. The E-R Diagram is the picture used to represent and test the knowledge obtained from Data Modelling.

The output of Data Modelling includes:

- E-R Diagram,
- Descriptions of entities and their relationships,
- Dttributes and their descriptions,
- Edit rules,
- Business rules,
- Volumetrics.

The last step in creating entity relationship diagrams is the specification of the relationships among the entities. Just as every object in the real world has some kind of relationship to one or more objects so too the entities in a database are related to other entities.

The nature of relationships between entities is usually implied in the very definition of the entity. Despite the obviousness of these relationships, it is important to review all entities and specify how they relate to each other.

There are at least three types of relationships possible:

1. One to one, where one entity corresponds to exactly another entity. For example, a table about patient's death has a one to one relationship with the table "Person."
2. One to many, where one instance of one entity can be repeatedly used by another. For example the look up table Gender may be repeatedly used in the table "Patient."
3. Many to many where one instance of both entities can be repeatedly used by another. For example, tables "Patient" and "Clinician" have many to many relationships as a clinician may have many patient and a patient may have many clinicians.

Sometimes, the relationship between two entities is not clear. The most common cause is that a third entity is missing. This often occurs when two entity have many to many relationship. For example, the entity Patient and the entity Clinician have, as mentioned earlier, many to many relationship. It is difficult to show these relationships inside a database in a way that can easily be manipulated.

An alternative is to show a new table that links these two tables to each other and has one to many relationship to each of the tables.

For example, we can make a new table called Visit. Within a visit a patients is diagnosed. Both the patients and the clinicians identity are kept in the visit table. The Visit table has one to many relationship with either patient or clinician table. Sometimes, as we specify the relationships among entities, a new entity must be defined.

Linkages between entities are part of the business rules that databases should capture. In our example, the business rule for the linkage between a Clinician and a Patient is that a clinician may have zero, one, or, more patients.

The business rule for the linkage between the Patient and the Clinician is that a patient may have one, or, more clinicians. Note that these are the business rules that someone may have specified. In a different information system someone could decide that a patient can only have one clinician at a time, or that the number of clinicians dealing with a patient must always be 3, or some other similar rule. The important point is that entities can be linked to each other, and that the nature of the linkage is part of the business rules of the system.

In Access, a database, The line shows the relationship between the two tables and the shared field shows the nature of the relationship.

The arrow shows if the relationship is one to many, with the many side shown by the direction of the arrow. As with the specification of the entities discussed at the beginning of this lecture, the documentation of the relationships is part of the logical information model.

The format for documenting the linkages among entities includes the name of both entities, the verb phrase that describes the semantics of the linkage and the cardinality of the linkage (*i.e.* whether one to one, one to many or many to many).

The statement of the cardinality can be made plain English. All relationships must be documented before proceeding to the physical design of the database.

Components of Entity-Relationship Diagram

Entities

An entity is a person, place, thing, event, or concept of interest to the business or organization about which data is likely to be kept. For example, in a school environment possible entities might be Student, Instructor, and Class.

Entity type refers to a generic class of things such as Company. Entity is the short form of entity-type. Entity occurrence refers to specific instances or examples of a type. For example, one occurrence of the entity Car is Chevrolet Cavalier. An entity usually has attributes (*i.e.*, data elements) that further describe it. Each attribute is a characteristic of the entity. An entity must possess a set of one or more attributes that uniquely identify it (called a primary key).

Identifying entities is the first step in Data Modelling. Start by gathering existing information about the organization. Use documentation that describes the information and functions of the subject area being analysed, and interview subject matter specialists (*i.e.*, end-users). Derive the preliminary entity-relationship diagram from the information gathered by identifying objects (*i.e.*, entities) for which information is kept. Entities are easy to find. Look for the people, places, things, organizations, concepts, and events that an organization needs to capture, store, or retrieve information about.

Types of Entities

Different types of entities are required to provide a complete and accurate representation of an organization's

data and to enable the analyst to use the Entity-Relationship Diagram as a starting point for physical database design.

Types of entities include:

- Fundamental where the entity is a base entity that depends on no other for its existence. A fundamental entity has a primary key that is independent of any other entity and is typically composed of a single attribute. Fundamental entities are real-world, tangible objects, such as, Employee, Customer, or Product.
- Attributive where the entity depends on another for its existence, for example, Employee Hobby depends on Employee. An attributive entity depends on another entity for parts of its primary key. It can result from breaking out a repeating group, the first rule of normalization, or from an optional attribute.
- Associative where the entity describes a connection between two entities with an otherwise many-to-many relationship, for example, assignment of Employee to Project (an Employee can be assigned to more than one Project and a Project can be assigned to more than one Employee). If information exists about the relationship, this information is kept in an associative entity. For example, the number of hours the Employee worked on a particular Project is an attribute of the relationship between Employee and Project, not of either Employee or Project. An associative entity is uniquely identified by concatenating the primary keys of the two entities it connects.

The common data elements are put in the supertype entity and the specific data elements are placed with the subtype to which they apply. For example, Employee (supertype) may contain three subtypes, Permanent Employee, Part-time Employee, and Temporary Employee.

All data elements of the supertype must apply to all subtypes. Each subtype contains the same key as the supertype.

Relationships between an entity supertype and its subtypes are always described as "is a." For example, Employee is a Permanent Employee, Employee is a Part-time Employee.

Identifying Entity Supertypes/Subtypes

Entity supertypes/subtypes involve classes of entities that are truly different, but at the same time, significantly similar.

When identifying supertypes/subtypes, look for:

- Entity types that have the same attributes,
- Entity types that participate in the same relationships,
- Occurrences of an entity that do not participate in all the relationships in which the entity type participates,
- Occurrences of an entity that do not have all the attributes that the entity type has.

Categories of Entities

There are different general categories of entities:

- Physical entities are tangible and easily understood. They generally fall into one of the following categories:

- People, for example, doctor, patient, employee, customer,
- Property, for example, equipment, land and buildings, furniture and fixtures, supplies,
- Products, such as goods and services.
- Conceptual entities are not tangible and are less easily understood. They are often defined in terms of other entity-types.

  They generally fall into one of the following categories:
  - organizations, for example, corporation, church, government,
  - agreements, for example, lease, warranty, mortgage,
  - abstractions, such as strategy and blueprint.
  - Event/State entities are typically incidents that happen. They are very abstract and are often modelled in terms of other entity-types as anassociative entity. Examples of events are purchase, negotiation, service call, and deposit. Examples of states are ownership, enrollment, and employment.

Imposter Entities

When an Entity is not an Entity

There are a number of things that may appear to be entities about which facts are kept, but which should not be defined as such.

These include:

- Processes,

- Calculations,
- Reports,
- Facts about entities.

Processes

Processes may actually perform actions on entities but are not, themselves, entities. Examples are:

- Payroll deduction,
- Budgeting (an action on an organization unit).

Calculations

Calculations are derived from the attributes of an entity. *Examples are*:

- Inventory level,
- Average age,
- Net worth.

Reports

Reports present facts about one or more entities. *Examples are*:

- Project schedule,
- Income statement.

Facts About Entities

Facts about entities describe characteristics of an entity and should be modelled as attributes. *Examples are*:

- Telephone number,
- Date of hire.

Attributes Define Entities

Collectively, attributes define an entity. An attribute is meaningless by itself. For example, date of birth comes from

the context of the entity to which it is assigned, for example, date of birth of an employee. Attributes are not shown on the Entity-Relationship Model but are recorded in the underlying data dictionary which contains the definitions of attributes for all entities and relationships identified in the model. An attribute should not have facts recorded about it. In practice, however, there are exceptions.

For example, you might wish to show address as an attribute of Customer. Address is not significant enough to be modelled as an entity in its own right and would typically be shown as an attribute of Customer. However, at the detailed level, it may itself have attributes such as an indicator for mailing address or home address.Attributes do not have to be recognized and defined during the early stages of entity definition. Entity definition is an iterative process, and it is unlikely that a completely satisfactory Entity-Relationship Model will be obtained on the first iteration.

Identifying Attributes

To identify entity attributes, examine:

- All external entities from the Context Diagram,
- The data flows passed by the external entities,
- Existing automated data,
- Each entity (*i.e.*, generate a list of entity attributes that describe the entity).

Attributes Versus Data Elements

Attributes have a looser description than data elements. For instance, whereas an attribute may have only a descriptive name, a data element needs:

- A size and range,
- A format and length,
- An accurate and detailed description,
- valid values,
- Defined edit rules.

Some attributes may be converted into many data elements.

For instance, the attribute "address" may become four data elements representing:

1. Street Address,
2. City/Town,
3. State/Province,
4. Postal or Zip Code.

Additional data elements may also be defined as a result of customer requirements. For example, the customer may require a list of all companies by county. For the purposes of Data Modelling, attributes and data elements are often considered identical because attributes in the data model typically become data elements in the database.

Relationships

A relationship is an association that exists between two entities. For example, Instructor teaches Class or Student attends Class. Most relationships can also be stated inversely. For example, Class is taught by Instructor. The relationships on an Entity-Relationship Diagram are represented by lines drawn between the entities involved in the association. The name of the relationship is placed either above, below, or beside the line.