Management Information Systems
Rutgers Business School / Undergraduate New Brunswick
Professor Eckstein, Spring 2006

Class Notes

Class 1 — Overview, Course Rules, General Definitions and Trends

Overview
- Topic: using computer and network technology to help run businesses and other organizations
- Won’t focus especially on “managers”
- Will combine “Top-down” descriptive learning (the TRP book) with “bottom-up” learning by example (Microsoft Access and GB book)

Rules and Procedures – see the syllabus and schedule

Data, Information and Knowledge
- Datum is singular, data is plural
- Information is data structured and organized to be useful in making a decision or performing some task
- Knowledge implies “understanding” of information
  - Knowledge representation in computers is called “artificial intelligence” (AI). It got a lot of hype in the 1980’s, and then went somewhat out of fashion, but it is still growing gradually. We will not discuss it much, and stick to information instead.

Information systems
- The ways that organizations
  - Store
  - Move
  - Organize
  - Manipulate/process their information
- Components that implement information systems – in other words, Information Technology
  - Hardware – physical tools: computer and network hardware, but also low-tech things like pens and paper
  - Software – (changeable) instructions for the hardware
  - People
  - Procedures – instructions for the people
  - Data/databases
- Information systems existed before computers and networks – they just used very simple hardware that usually didn’t need software (at least as we know it today).
- Impact of electronic hardware
  - Greatly reduces cost and increases speed of storing, moving (etc.) information
  - Information doesn’t have to be stuck with particular things, locations, or people
- Can increase efficiency of things you already do
- Can permit new things
  - Combine scale efficiencies of a large firm with responsiveness of a small one – for example, produce at a mass scale, but customize each item
  - Can remove middlemen or levels of inventory that shielded you from handling information
  - Makes physical location etc. less important – for example, one can now browse for obscure products from rural North Dakota
  - Can make it easier for parties to know one another exist and transact with one another (for example, eBay)
- Can have a downside
  - Small glitches can have much wider impact (for example, “Bugs Ground Planes in Japan”, TRP p. 23)
  - Fewer people in the organization understand exactly how information is processed
  - Sometimes malfunctions may go unnoticed (American Airlines yield management story)

Waves of technology
- The first wave of electronic technology replaced manual/paper systems with isolated or tightly coupled computers responsible for a particular business function (such as accounting or ticket reservations), often running on “mainframes”.
- These systems were gradually interconnected between organizations’ functional units
- The second wave of technology has been toward networked/distributed computing – many systems tied together by large networks, and networks tied together into “inter-networks”, that is, “the internet”.
- Networked systems have provided a lot of new ways to operate businesses and perform transactions (what Chapter 1 of TRP calls the “new economy”)
- Networked electronic information systems encourage a trend to reduce the degree to which each transaction/operation is tied to a physical place or thing. Examples (not all equally successful!):
  - Shopping online
  - MetroCards instead of tokens
  - Digital images instead of film
  - Electronic payments instead of cash
  - Online college courses

Class 2 – Hardware Basics

Electronic computing equipment is constructed from
- Wires
- Transistors and the like
- Storage devices (such as tiny magnets) that can be in one of two possible states

Although technically possible, we do not want to think about complex systems as being made out transistors and tiny magnets. If somebody said “make an accounts payable system, and here is a pile of transistors and tiny magnets”, you would probably not get very far!
The keys to organizing information systems (and other computer-based systems) are

- Layering – provide foundations that do simple tasks and then build on them without worrying about how they work internally
- Modularity – divide each layer into pieces that have well-defined task and communicate with one another in some standardized way

The most basic layering distinction is hardware and software

- Hardware consists of physical devices (like PC’s) that are capable of doing many different things – often generic devices suitable for all kinds of tasks
- Software consists of instructions that tell hardware what to do (for example: word processing, games, database applications…)

Kinds of hardware

- Processors (CPU’s = central processing units; like “Pentium IV”); typical processor subcomponents:
  - Control unit/instruction decoder
  - Arithmetic logical unit (ALU)
  - Registers (small amount of very fast memory inside the processor)
  - Memory controller/cache memory
  - A “microprocessor” means all these subcomponents on a single “chip” that is manufactured as a single part. This only became possible in the 1970’s. Before that, a CPU consisted on many chips, or even (before that) many individual transistors or vacuum tubes!
- Primary storage
  - RAM
  - ROM (read only)
  - Variations on ROM (EPROM – can be changed, but not in normal operation)
- “Peripherals” that move information in and out of primary storage and the CPU,
  - Things that can remember data: secondary storage
    - Should be “non-volatile” – remembers data even if electrical power is off
    - Generally it is slower to use than primary storage
    - Most ubiquitous example – the “hard” disk
    - Removable – “floppy” disks, optical CD/DVD disks, memory sticks
      - Read/write
      - Write once (like a CD-R and DVD-R)
      - Read only
    - Other input/output (“I/O”) – screens, mice, keyboards etc.
- Network hardware
- The wires that move data between hardware components are often called “buses” (much faster than Rutgers buses!)
- Cache memory is fast memory that is usually part of the processor chip. The processor tries to keep the most frequently used instructions in the cache. It also tries to use the cache to keep the most frequently used data that will not fit in the registers. The more cache, the less the processor has to “talk” to the primary storage memory, and generally the faster it runs.

If you look at things at each hardware module, you’ll find layers and modules within it. For
example, a CPU will have modules inside like the ALU, control unit, registers, memory controller, etc. Within each of these, you will in turn find modules and structure.

Standard way of arranging hardware (like PC’s and laptops)

- One processor and bank of memory, and everything attached to them
  - A key innovation in the design of modern computers was to use the same main memory to hold instructions and data. This innovation is generally credited to Hungarian-born mathematician John Von Neumann (who spent the last 12 or so years of his life in New Jersey at the Institute for Advanced Study in Princeton), and his EDVAC research team. It was critical to modern computing, because it allowed computers to manipulate their own programs and made software (at least as we now conceive it) possible.

- Variations on this basic theme that are common today:
  - Desktops are regular PC’s
  - Laptops are similar but portable
  - Servers are similar to desktops, but with higher quality peripheral components – intended to run websites, central databases, etc.
  - Mainframes are like PC’s, but designed to do very fast I/O to a lot of places at the same time (they used to compute faster as well). Mainframes can perform much better than PC’s in applications involving moving lots of data simultaneously between many different peripherals (for example, an airline reservation system)
  - Supercomputer can mean several things. At first it meant a single very fast processor (see picture at top left of TRP p. 410) designed for scientific calculations. This approach gradually lost out to parallel processing supercomputers (see below), and is now fairly rare.

More recent things –

- Thin client systems – a cheap screen/processor/network interface/keyboard/mouse combination without secondary storage or very much memory. Typically, the software inside the thin client is fairly rudimentary and is not changed very often; the “real” application resides on a server with secondary storage and more RAM. Thin clients are basically more powerful, graphics-oriented revivals of the old concept of “terminals”. These can be cost-effective in some corporate settings.

- 2 to 16 processors sharing memory – “Symmetric Multiprocessors” or “SMP’s” (servers and fast workstations). Most larger servers and even high-end desktops are now of this form. “Dual-core” microprocessors – essentially two processors on a single chip – are becoming common in high-end PC’s.

- Parallel processing involves multiple memory/CPU units communicating via a (possibly specialized) network. Such systems can contain a few standard (or nearly standard) microprocessor modules, up to tens of thousands.
  - Large websites and now large database systems are often implemented this way: each processor handles some of the many users retrieving data from or sending data into the system
  - Large scientific/mathematical supercomputers are now constructed this way
  - Depending on the application, each processing module might or might not have its own secondary storage
- **Blade or rack** servers: one way of setting up parallel/distributed process capabilities. The “blades” are similar to PC’s, but fit on a single card that can be slotted into a rack to share its power supply with other “blades”.

- **Enterprise storage systems** or “Disk farms” that put together 100’s-1000’s of disks and connect them to a network as a shared storage device

- **Mobile devices** such as web-enabled cell phones, wireless PDA’s, or Blackberries. Presently, these are not particularly powerful, but can already be an integral part of an organization’s information system, and very useful because of their mobility. Their processing power is already quite significant, but battery life, small screen size, and small keyboard sizes are problems.

- Basically, network technology has “shaken up” the various ways that hardware modules are connected, although the basic PC style is one of the most common patterns

- Nowadays, only a one-person company has only one computer. So all companies are doing a certain amount of “parallel” or “distributed” computing.

- **Specialized and embedded** systems: microprocessors, typically with their programs (“firmware”) burned into ROM, are “embedded” in all kinds of other products, including music players, cars, refrigerators,…

---

**Data representation – introduction:**

Computers store number in base 2, or binary. In the decimal number system we ordinarily use, the rightmost digit is the “1’s place”; as you move left in the number, each digit position represents 10 times more than the previous one, so next we have 10’s place, then a 100’s place, then a 1000’s place and so forth. Thus, 4892 denotes, or $(2 \times 1) + (9 \times 10) + (8 \times 100) + (4 \times 1000)$, or equivalently $(2 \times 10^0) + (9 \times 10^1) + (8 \times 10^2) + (4 \times 10^3)$. In the binary system, we also start with a 1’s place, but, as we move left, each digit represents 2 times more than the previous one. For example,

$$100101_2 = (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

$$= (1 \times 1) + (0 \times 2) + (1 \times 4) + (0 \times 8) + (0 \times 16) + (1 \times 32)$$

$$= 37_{10}$$

When bits are combined to represent a number, sometimes one bit – often called a “sign bit” – is set aside to indicate + or –. (Most computers today use a system called “two’s complement” to represent negative numbers; I will not go into detail, but it essentially means the first bit is the sign bit).

There are also formats that are the binary equivalent of “scientific notation”. Instead of $3.478 \times 10^5$, you have things like $1.00101011 \times 2^{13}$. These are called “floating point”. They are usually printed and entered in decimal notation like $3.478 \times 10^5$, but represented internally in binary floating point notation (note: this can occasionally cause non-intuitive rounding errors, like adding 1000 numbers all equal to 0.001, and not getting exactly 1).

Some common amounts of memory for computers to manipulate at one time:

- A single bit – 1 means “yes” and 0 means “no”
• 8 bits, also called a “byte” – can hold $2^8 = 256$ possible values. These can represent a single character of text, or a whole number from 0 to 255. If one bit is used to indicate + or –, can hold a whole number from –128 to +127.

• 16 bits, or two bytes. Can hold a single character from a large Asian character set, a whole number between 0 and about 65,000, or (with a sign bit) a whole number between about –32,000 and +32,000.

• 32 bits, or four bytes. Can hold an integer in the range 0 to about 4 billion, or roughly –2 billion to +2 billion. Can also hold a “single precision” floating-point number with the equivalent of about 6 decimal digits of accuracy.

• 64 bits. Can hold a floating-point number with the equivalent of about 15 digits of accuracy, or some really massive whole numbers (in the range of + or – 9 quintillion).

Performance and memory measures for processors:
• Clock speed – number of hardware cycles per second. A “megahertz” is a million cycles per second and a “gigaherz” is a billion cycles per second. But a “cycle” is hard to define and what can be accomplished in a cycle varies, so don’t try to compare clock rates of different kinds of processors. For example, a “Pentium M” does a lot more per cycle than a “Pentium 4”.
  o Note: it is very uncommon nowadays, but there are such things as CPU’s with no clock, called *asynchronous* CPU’s. There are some rumors they could be revived.
  o An alternative measure is MIPS (millions of instructions per second); this measure is less sensitive to details of the processor design, but different processors can still do differing amounts of work in a single “instruction”.
  o Another alternative measure is FLOPS (floating point operations per second). This is less processor-sensitive than clock speed and MIPS, but measures only certain kinds of operations. The values usually quoted are “peak” values that are hard to achieve, and how close you can get to peak depends on the kind of processor.
  o Another alternative is performance on a set of “benchmark” programs. This is probably the best measure, but is rarely publicized.

• Word length – the number of bits that a processor manipulates in one cycle.
  o Early microprocessors in the late 70’s had an 8-bit (or even a 4-bit) word length. However, some of the specialized registers were longer than 8 bits (otherwise you could only “see” 256 bytes of memory!)
  o The next generation of microprocessors, such as used in the IBM PC, had a 16-bit word length
  o Most microprocessors in use today have a 32-bit word length
  o However, 64-bit processors are gradually taking over
  o Other sizes (most typically 12, 18, 36, and 60 bits) used to be common before microprocessors but started disappearing in the 1970’s in favor of today’s 8-16-32-64 scheme, invented by IBM in the 1960’s.

• Bus speed – the number of cycles per second for the bus that moves data between the processor and primary storage. In recent years, this is generally slower than the processor clock speed

• Bus width – the number of bits the CPU-to-primary-storage bus moves in one memory cycle. This is typically the same as the processor word length, but does not have to be
• Note: the TRP text conflates the concept of bus width and bus speed!
Bottom line: processors are extremely complicated devices with many capabilities and it’s hard to boil down their performance into a single number.

Memory measures:
• **Kilobyte** or KB. Typically used in the binary form, \(2^{10} = 1,024\) bytes. This is about \(10^3 = 1,000\), hence the prefix “kilo”, meaning “1,000”. Just to confuse things, in some other contexts, the “kilo” prefix is sometime used in its decimal form, meaning exactly 1,000.
• **Megabyte** or MB. In binary form, \(2^{10} = 1,024\) kilobytes = \(2^{10} \times 2^{10} = 2^{20} = 1,048,576\) bytes. In the decimal form it means precisely 1 million.
• **Gigabyte** or GB: In binary form, \(2^{10} = 1,024\) megabytes = \(2^{10} \times 2^{20} = 2^{30} = 1,073,741,824\) bytes. In the decimal form it means precisely 1 billion.
• **Terabyte** or TB: \(2^{10} = 1,024\) gigabytes
• **Petabyte**: \(2^{10} = 1,024\) terabytes
• **Exabyte**: \(2^{10} = 1,024\) petabytes

Today, primary storage is typically in the hundreds of megabytes to small numbers of gigabytes per processor. Secondary storage is usually tens to hundreds of gigabytes per hard disk, with one to four hard disks per processor. Terabytes are currently the realm of enterprise storage systems, but single hard disks storing a terabyte should appear soon. Petabytes and exabytes are **big**.

Performance trends:
• Moore’s law: the number of transistors that can be placed on a single chip – roughly equivalent to the computing power of a single-chip processor – double approximately every 18 to 24 months. This “law” has held approximately true for about 25 years. Gordon Moore was a co-founder of Intel Corporation. Doubling in a fixed time period is a form of exponential growth. Physics dictates this process cannot continue indefinitely, but so far it has not slowed down.
• Primary storage sizes are also essentially proportional to the number of transistors per chip, and roughly follow Moore’s law.
• Hard disks, measured in bits of storage per dollar purchase price, have grown at an exponential rate even faster than Moore’s law
  o This leads to problems where disk storage outstrips processing power and the capacity of other media one might want to use for backup purposes (like magnetic tape)

**Class 3 – Software Basics**

In the context of information systems, **hardware** is the physical equipment that makes up a computer system. **Software** consists of the instructions that tell the equipment how to perform (presumably) useful tasks.

Software is possible because of the **Von Neumann architecture**, called the **stored program concept** in the textbook. This innovation, dating back to fundamental research from World War II through the early 1950’s, allows computers to manipulate their software using the same processors, memory, secondary storage, and peripherals that they use for things we would more normally think of as “data”, such as accounting transaction information.
The original term for software was program, meaning a plan of action for the computer. People who designed programs were called programmers. Now they are also called developers, software engineers, and several other things.

At the most fundamental level, all computer software ends up as a pile of 0’s and 1’s in the computer’s memory. The hardware in the processor’s control unit implements a way of understanding numbers stored in the computer’s memory as instructions telling it to do something.

<Review binary representation>

For example, the processor might retrieve the number “000000100110100” from memory and interpret it as follows:

```
0000001  0011  0100
“Add”  “R3”  “R4”,
```

meaning that the processor should add the contents of register 3 (containing a 32- or 64-bit number) to the contents of register 4 (recall that registers are very fast memory locations inside the processor).

The scheme the processor uses to interpret numbers in this way is called its machine language. The dialect of machine language depends on the kind of processor. For example, the PowerPC processors in Macintosh computers use a totally different machine language from Intel Pentium processors – a sensible program for one would be nonsense to the other. However, many internally different processors implement essentially identical machine languages: for example, all Intel and AMD 32-bit processors implement virtually the same machine language, so most machine-language programs can be moved freely from one to the other.

Writing programs in machine language is horrible: only very smart people can do it at all, and even for them it becomes very unpleasant if the program is longer than a few dozen instructions. Fortunately, the stored program concept allows people to express programs in more convenient, abstract ways.

The next level of abstraction after machine language is called assembler or assembly language. Assembly language allows you to specify exactly what a machine language program should look like, but in a form more intelligible to a person. A fragment of a program in assembly language might look like

```
Add R3, R4   // Add register 3 to register 4
Comp R4, R7   // Compare register 4 to register 7
Ble cleanup  // If register 4 was less <= register 7, go to “cleanup”
Load total, R3 // Otherwise, load a memory location called “total” into
Etc.         // register 3
```
Note that the stuff after the “//” markers are explanatory comments and not part of the program. People can actually write amazingly long and powerful programs this way, but it is very inefficient and error-prone. Furthermore, the program is still tied to a particular machine language and thus to a particular family of processors. An assembler program takes the text representation of the program, as above, and translates it into machine language. Note that without the stored program concept/Von Neumann architecture, you could not have an assembler – the results of the assembler program could not be used as instructions because the would be “data”. Note also that some poor soul had to write the very first assembler program in machine language!

After assembly language, higher-level languages evolved. These languages allow programs to be expressed in a form closer to human forms of communication like English and algebra. For example,

\[
\text{PROFIT} = \text{REVENUE} - \text{COST} \\
\text{IF} \ \text{PROFIT} < 0 \ \text{THEN PRINT} \ \text{“We lost money!”}
\]

Examples of higher-level languages:
- BASIC (numerous dialects)
- C (allows a lot of low-level control)
- C++ (similar to C, but “object-oriented”)
- Java (looks a bit like C++, but simplified in many ways)
- JavaScript (web-oriented, simplified offshoot of Java)
- FORTRAN (most often used for science and engineering applications)
- COBOL (used for old-fashioned mainframe-style business application)
- … many, many, more…

The classic way of implementing a high level language is called a compiler.
- Compilers translate a program in a higher level language into a machine-language program (in Windows, a “.exe” file).
- Internally, they may actually translate the program into various levels of internal intermediate representation, then into assembly language, and finally into machine language
- Pieces of the program can be compiled separately into little chunks of machine language called object modules (“object” here has a different meaning from “object-oriented”), which are then combined into a single machine language program by a program called a linker. Commonly used object modules can be grouped into “libraries” so they don’t have to be recompiled all the time.
- The result of the compiler/linker combination is a “raw” machine language program. So, to run on computers with different machine languages (or even different operating systems; see below) a program may have to be compiled multiple times in different ways.

Interpreters are an alternative approach to using compilers. With classic interpreters, the program is stored in essentially the same textual form the programmer used to write it. When you “run” the program, the interpreter essentially translates each line as it is encountered.
This can be inefficient: if a line is executed many times (in a loop, for example), it could get translated many times. There are ways of reducing this penalty, but fundamentally interpreted programs run slower than compiled ones.

Interpreters tend to make writing and debugging the program easier.

The program can often be easily moved between different kinds of computers so long as each has an interpreter for the language being used.

In many applications, the difference in speed isn’t noticeable (especially as processors keep getting faster), so the portability and easier debugging of interpreters may make them a better choice.

There are intermediate approaches between compilers and interpreters; one example is the Java language. A compiler can turn the program into some intermediate form (still not tied to a particular machine language); at run time, a simpler, faster interpreter “translates” the intermediate form.

Note that programs in high-level languages become essentially independent of the particular type of processor or operating system used. For example, once the above program is compiled, the variable PROFIT might reside in register 3, and the “Print” statement might interact with the operating system in a particular way, but we don’t have worry about such details. However, once one gets into advanced programming techniques, there are ways to make programs dependent on particular features of the operating system or processor. If one avoids such techniques or uses them carefully, a compiled program can be moved between systems simply by recompiling it. Interpreted programs can often be moved with no changes.

Software structure: Only computers with very simple jobs (like running a thermostat) have just one layer of program. Most software systems, like hardware, are constructed out of layers and modules so that the complexity of the system is manageable. The most popular current arrangement is:

- **BIOS** is lowest level (Basic I/O System)
  - The BIOS typically comes with the computer in “read-only” memory, but can be updated by “flashing” that memory. Updating the BIOS is typically rare.
  - The BIOS provides low level “subroutines” (program modules) to send data to and from peripherals etc.

- **Operating system (OS)** on top of BIOS.
  - The OS implements basic user-understandable capabilities
    - Arbitrates between tasks
    - Tracks files
    - Structures contents of screen
  - You can run different operating systems on top of the same BIOS, for example either Windows or Linux on the same kind of computer.
  - Because the BIOS manages the details of communicating with the computer’s peripherals etc., the same operating system can run in an outwardly identical manner on computers with different motherboards etc.
  - Within an operating system, you will find modules and layers, for example
    - The *kernel* manages the basic scheduling of tasks
    - *Device drivers* send and receive data from particular peripherals
- The file system understands how secondary storage is organized
- The GUI (Graphical User Interface) manages the screen, mouse etc.
- Compilers and linkers
  - Some of these modules and layers may have their internal own modules and layers, and so forth.
  - Modern operating systems are written (mostly, anyway) in high-level languages, so they can be moved between computers with different machine languages (once everything is compiled properly).
- **Application software** on top of the operating system. Application software makes the computer perform a specific, presumably useful task. You may run totally different software on top of the same operating system – for example, a video game or accounting software.
  - Application software often has layers. A very common situation:
    - Relational database engine (MS Access, Oracle, Sybase, DB2, etc.)
    - Specific business application built on top of engine (we will study how to do that!)
  - As always, these layers may contain their own modules and layers

When programs are constructed out of layers, there are specific ways that the layers interact with one another. These are usually called “API’s” – *Application Program Interfaces*.

The term **system software** refers to lower-level software like the BIOS or the operating system, whereas **application software** is more oriented to specific tasks. As software gets more complicated and varied, it sometimes is hard to the exactly “draw the line” between “systems” and “applications” software; it can depend on your viewpoint.

The situation with languages is also blurring. In addition to classic **procedural** languages like assembly language, C, and JavaScript, which express lists of explicit instructions (“do this, then do that”), other languages have evolved for expressing other sorts of information. For example:

- **HTML** (HyperText Markup Language) expresses the appearance of textual information with “hypertext” links between elements, graphics, and “fill-in” boxes or “form” elements.
- **SQL** (Structured Query Language) describes information to be retrieved from a database. It doesn’t necessarily say exactly how to assemble the information, just what kind of information is needed.
- **Modeling languages** for expressing the nature of a mathematical problem, but not necessarily a procedure for solving it.
- Non-procedure semantic languages derived from XML. These languages express some kind of free-form, richly structured information, but not necessarily procedures. For example, XML dialects exist for sheet music, mathematical equations, geographic information. The main file formats for the upcoming version of Microsoft Office are reputed to be XML-based.

Such nonprocedural languages are often a critical element of modern software.
Often a key portion of an application is not expressed in either a programming or non-programming language, but in configuration tables of some kind, which tell a general-purpose piece of software how to behave. For example, as we shall soon see, you can set up Microsoft Access to manipulate a wide variety of kinds of databases. The specification of the properties of a database exist within an Access file in the form of configuration tables or metadata, which the user can see and change through a graphical user interface. For example, these metadata might say that the database has a single data table, whose first column in this table is a 10-character piece of text called “CustomerID”, the second column is a 20-character piece of text called “CustomerFirstName”, and so forth. Many kinds of software have this kind ability: programmability or broad configurability which is not exactly classical “programming”. Again, it may be a key part of your software system.

**Classes 4-5: See Chapters 2-3 of the GB Book**

**Class 6: See the Handout on Memory Storage Calculations**

**Partial Notes for Class 7 (TRP Sections 3.1-3.4)**

We have seen a little bit now about the tables in which business data are stored, and to how to calculate the amount of storage they might consume. Managing data in large modern organizations can be very challenging. Particular challenges:

- The volume of data increases as new data is added. New technologies mean that gathering new data is easier and faster. So, not only is the total volume of data increasing, but the rate at which is increasing is also increasing!
- Data tend to be scattered throughout the organization. It is often desirable to centralize data storage, but by no means always – it may be better to leave departments or working groups “in charge” of the data they use the most. It is costly and risky to replace older “legacy” information subsystems that are working smoothly. Sometimes it may be better to created “federated” systems that combine information from constituent systems.
- We may also want to use data from outside the organization (either public-domain or purchased). It may also be advantageous to share some information with suppliers or vendors (for example, sharing information about inventories can reduce inventory fluctuations and costs throughout a “supply chain”).
- Data security and quality are important, but are more easily jeopardized the larger an information system becomes.

The text distinguishes between two main modes of using data:

- **Transactional processing** (sometimes called TPS): keeping track of day-to-day events, such as logging orders and shipments, and posting entries to accounting ledgers. In terms of a data table, transaction processing means an ongoing process of adding rows (for example, to reflect a new order), modifying table cells here and there (for example, if a customer changes their telephone number), and perhaps deleting rows.
- **Analytical processing**: means using multiple table rows to obtain “higher-level” information. Entering a row into a table to reflect a newly-received order would be
transaction processing; an example of analytical processing would be computing the number of orders and total dollar value of orders for this month, and comparing them to last month.

- Analytical processing can be as simple as sorting, grouping, and summary calculations in an Access query or report. For example, providing all group managers with a summary of their groups’ costs for the month, broken down by cost category. This kind of application can be called “classic” MIS (Management Information Systems).

- Analytical processing can get a lot more sophisticated. For example, data mining refers to using sophisticated statistical or related techniques to discover patterns that might not be obvious in classical reports.

- Decision support can involve using the data to help managers make complex decisions, i.e. how to route 400 shipments from 20 warehouses to 100 customers.

- Database systems like Access don’t ordinarily do data mining or decision support by themselves. For such uses, usually need to be connected to other pieces of software.

Sometimes it can be mistake to do transaction processing and analytical processing on the same database, especially if the analytical processing is very time consuming or complex.

- The analytical processing may make the transaction system run slowly
- Conversely, the transactions may interfere with the analytical processing and make it run to slowly
- If an analytical processing step takes too long, the data it is using may change in the middle of its calculation. “Locking” the data to avoid this can block transaction processing.

It may be better to make a copy or “snapshot” of the database used for the transaction system.

- This is often called a “data warehouse”
- You can do a lot of analysis on the data warehouse without disrupting the transaction system and a lot of transactions without disrupting data analysis
- The data warehouse will not reflect the very latest transactions, but for large-scale aggregate analysis, that may not be a big problem.

**Remainder of Class 7: roughly follows GB pages 218-222**
(However, I also introduced the notion of a repeating group.)

**Class 8: Followed Database Design Handout**

**Class 9: Video Store Database Design Example; Subtypes**

**Class 10: Lab Class**
Class 11: Lab Class, Personnel Database Design

Class 12: Information System Application Areas and Classification

Figure 2.2 on TRP p. 36 is a nice overview of the interaction of IT and organization structure.

- Base of “pyramid” is the IT infrastructure (hardware, system/basic software)
- IT services constitutes the human base of the pyramid
- Transaction processing systems manage day-to-day record keeping, and may or may not cut across functional divisions.
- Above the transaction base are various functional divisions
  - Accounting
  - Finance
  - Human Resources
  - Operations/Production
  - Marketing
  - Etc.
- These may have their own systems/applications fed by data from the transaction base
- Decision making levels above the transaction base are often roughly categorized as
  - Operational
  - Tactical
  - Strategic
- The operational/tactical/strategic levels might or might not require specialized information systems or applications
- Enterprise Resource Planning (ERP) systems are conceived as a means of integrating transaction processing and most higher level IT applications.
  - Try to fulfill most of your IS software needs by purchasing a massive product from a single vendor, then configuring it to your needs (and ignoring unneeded parts).
  - Advantages:
    - Speeds up information flow between functional areas, since everybody is using the same up-to-date information. Without ERP, you need to build “bridges” between systems used by different functional areas:
      - “As needed” bridges may be slow and cumbersome
      - Periodic bridges may result in out-of-date information; for example, accounting’s information system may be a few days behind sales’ information system.
    - Benefit from enormous design investment by ERP vendors
    - Less in-house programming
  - Disadvantages
    - Massive complexity – typical ERP databases have many thousands of tables
    - Configuration/customization may be tricky and require specialized skills
    - Loss of flexibility: it’s hard for ERP to anticipate everything every customer might want. You may have to do things “their way”.
    - High purchase cost.
Material on TRP pp. 37-39 was covered in earlier lectures.

Information systems may be classified by organizational scope (TRP p. 41)
- Functional – supports a single traditional department such as marketing and HR.
- Example: the personnel database we designed last time supports HR
- Enterprise-wide – serve several traditional departments or the entire enterprise (example: ERP tries to serve entire enterprise).
- Interorganizational – connect to the information systems of suppliers, customers, or partners. Examples:
- Supply chain management systems are a currently trendy example. Idea is to increase efficiency by sharing some information with suppliers/customers, and somehow sharing the monetary gains from sharing the information.
- When you order a used book through Amazon.com, their information systems automatically pass the order to an appropriate used book dealer.

Information systems may be classified by their degree of processing sophistication (not mentioned in chapter 2 of TRP, but important)
- **Transaction Processing Systems** (TPS): recording day-to-day business events; displaying results of very specific queries. Examples:
  - Record an order for customer \( X \)
  - Record that shipment \( Y \) was loaded onto truck \( Z \)
  - Display recent orders placed by customer \( X \)

In Access terms, one may think of such systems as functioning on the level of tables and forms.

- **Management Information Systems** (classic MIS): distill/aggregate information gathered by TPS into a form (usually reports) that give some sort of “bigger picture”. MIS systems typically perform relatively simple processing operations like sorting, grouping, adding up, and averaging, much as one can do in a Microsoft Access query. Examples:
  - Weekly, monthly, or quarterly sales reports broken down by region and product
  - Quarterly inventory summary
  - Annual report on employee sick time, broken down by location and department

Examples of tasks performed by classic MIS systems (TRP table 2.2, p. 44):
- Periodic reports – example: weekly sales figures
- Exception reports – example: when production defects exceed some limit
- Ad hoc reports – generated on demand (essentially an arbitrary query)
- “Drill down” – an interactive “report” allowing the user to focus detail on particular areas

In Access terms, we can think of such systems as functioning on the level of queries and reports.

- **Decision Support Systems** (DSS): try to go beyond simple reports, combining information from a database with relatively sophisticated analytical techniques, with a view to helping make some potentially complex or difficult decision. Examples:
Take today’s list of deliveries, and suggest how they should be split up among delivery trucks, and how each truck should be routed.

Statistically forecast next quarter’s demand for 20 different lines of shoes, and suggest an order quantity for each kind of shoe.

DSS systems typically include a database, but also require special programming. On its own, a DBMS (Database Management System) like Access or Oracle will not have sufficient functionality to implement a DSS. To implement a DSS, you need to extract information from the DBMS into another kind of program, or add special code to the DBMS.

For example, Access allows users to add specialized code in the *Visual Basic for Applications* (VBA) language.

Some ERP systems contain DSS modules to support standard kinds of decisions like production planning. These modules may have sophisticated functionality, but might or might not be suitable to a particular firm’s situation.

Sometimes a DSS can be entrusted to make certain routine decisions without human review, in which case I guess it would become a “DS” (Decision System), and not a Decision Support System.

Information systems may be classified by the management/decision level they support (TRP pp. 42-45):

- **Clerical/operational**: customer service, claims processing, package delivery, etc.
  - Supported by TPS, and occasional DSS (example: routes for a delivery driver)
- **Operational decision making**: first line operational managers such as dispatchers
  - Supported by TPS, plus MIS and/or DSS
- **Tactical management**
  - Supported by MIS and possibly DSS
- **Strategic management**
  - Supported by MIS and possibly DSS

All of the above may also be supported by basic office automation tools: e-mail, word processing, spreadsheets, etc.

No categorization approach is perfect or useful for all purposes. Real systems may cut across boundaries – for example, a system might have TPS and DSS aspects, or be used by both operational and tactical managers.

---

Class 13: Networks

Note: this class covers roughly the material in TRP chapters TG4, TG5, and 4. However, my emphasis will be different, and I’ll give some details not in the text.

We now discuss transmitting data between physically separated computers

- Something that connects two computers is a *link*.
- Many computers connected by many links comprise a *network*.
- Each computer on the network is called a *node*.
- Generally speaking, data should be able to get from any network node to any other node.
- There are many different shapes (or topologies) that can be used to build a network.
Today the predominant network topology is a collection of interconnected “stars.” At one time, interconnected “rings” were also popular, and they are still in use.

- Some nodes on the network serve primarily as connection points or to make sure data gets sent to the right place
  - Switches
  - Hubs
  - Routers

Kinds of links

- Link speed is usually measured in bits per second (b/s), with the usual (decimal) prefixes K (kilo/1,000), M (mega/1,000,000), G (giga/1,000,000,000), etc.
- Wires (usually copper) can be used in many ways.
  - Twisted-pair wires, such as
    - Regular telephone wiring
    - Ethernet wires, which currently come in three flavors, 10 Mb/s, 100 Mb/s, and 1Gb/s.
  - Coaxial cables, like those used for cable TV. These have higher theoretical capacity but are harder to work with.

Wires can carry a lot of data for short distances, but slow down for longer distances
- Optical fiber (carries light pulses)
  - Invented about 30 years ago
  - More economical than wire for high data rates and long distances. Links can have capacities in the many Tb/s
  - More difficult to work with than either twisted-pair wires or coaxial cables. In particular, it’s hard to “splice” two of them together (however, that also makes them more secure).

- Broadcast electromagnetic waves (radio/infrared/microwave) – “wireless”
  - Microwave links (directional – don’t broadcast equally in all directions)
  - Satellite links
  - Within-building (“wi-fi”) broadcast: capacities typically about 11 Mb/s right now
  - Wider area wireless: slower than wi-fi right now (cell-phone modems); this technology is just emerging, but I believe that demand will make it successful.

Current services are advertised at $80/month for 400-700 Kb/s access, with approximately the same coverage area as standard cell phones.

A history of computer communications:
- The first large-scale electronic networks built were telephone networks. But they were not used by computers initially (Because computers didn’t exist! In fact, “computer” was actually a job title for a person who did mathematical calculations for engineers and scientists)
- When computers started, each organization had its own computer in its own room. Data got in and out of the computer room by being physically carried as punched cards, printed reports, magnetic tape etc. (eventually floppy disks, too) – later called “sneakernet”.
- People began placing I/O devices outside the computer room, connected by wires: printers, card readers, terminals (=printer + keyboard or screen + keyboard), etc.
- Technology was invented to encode (modulate) data into sounds the telephone network could carry. The data would be “demodulated” back into bits at the other end (thus the term “modem” – modulator/demodulator); see TRP p. 461.
This allowed people to have terminals at home and work over telephone lines.

Many other business applications involving sending or receiving data from remote locations.

Early modems were slow (100 b/s = 0.1 Kb/s in the 1960’s). This gradually increased to about 56 Kb/s today.

The technology is still widely used, but in decline.

In the late 1960’s, interest was growing in large general-purpose data networks independent of the telephone network.

Before, these existed only for specialized application (mostly military).

ARPANET – the (defense department) Advanced Research Projects Agency NETwork was built in the early 70’s.

This network gradually evolved into “the internet”.

The internet had a fairly small user base until the mid 80’s. Then it began to gather momentum.

In the early 90’s, the “world wide web” became very popular and drove a massive expansion of the internet (along with the “.com” boom).

In the 90’s there was a general telecommunications boom of which the internet boom was a big part. Numerous firms tried to secure their place in the boom by building lots of network links, especially in North America.

A lot of network capacity was built. About the same time technology appeared that greatly increased the amount of data an optical fiber could carry (by simultaneously sending multiple information streams using light beams of several different colors). Things cooled off a lot, but internet use continues to climb.

How networks work: LAYERING is very important.

Bottom: physical layer – the physical workings of the links (wire, fiber, wireless, etc.)

Network layer (typically “IP”, standing for “internet protocol”): lets the network figure out what computer the data is meant for.

Currently, each computer has a 32 bit “IP address” (usually split into four bytes printed in decimal like 128.6.59.202).

The addresses have structure – for example “128.6” in the first two bytes of the address means somewhere at Rutgers (although 165.230 also means Rutgers), the 59 designates a particular “subnet” (roughly the same as a building), and the 202 identifies which computer on the subnet.

Note that most computers also have a “hostname” and “domain name” that is easier for humans to remember, like “business.rutgers.edu” or www.amazon.com.

While these are related to IP addresses, they aren’t exactly the same. Special computers called “name servers” provide the translation. Small organizations may not have a name server, relying on a name server from their internet service provider. Large organizations like Rutgers may have dozens of name servers.

32 bits are no longer enough space for an IP address, and we will gradually move from IPv4 (32 bits) to IPv6 (128 bit addresses). Various workarounds suffice for now:

- Dynamically allocating IP addresses only when computers are actively connected to the network (“DHCP” is a common way of doing this), or
- Grouping small sets of computers to share a single IP (network address translation or “NAT”).
• Transport layer (typically “TCP”). Specifies how data is split up and logically moved in the network
  o TCP specifies up to 65,000 logical “ports” for each computer on the network. Each port can be used for a different application.
  o For each port, there can be more than one “session” or logical connection between computers (for example, you could have two independent web browser windows connected to the same website from your own PC)
  o For each session, there may be a sequence of messages in each direction
  o TCP is a “packet switched” protocol – messages are cut up into “packets” that might take different paths through the network and are reassembled at the destination. Telephone networks are “circuit switched” – the whole conversation uses the same route through the network.
• Application layer: specifies different protocols for moving data in different ways. These constitute an “alphabet soup”:
  o First: TELNET (old) – run a terminal session (a text-based interaction between a person and a computer)
  o FTP (old) – move files back and forth (still in some use when security isn’t an issue)
  o SSH – encrypted terminal sessions and file transfers. This is how you connect to the “Eden” system to do text-based interactions. This works much the same way as TELNET and FTP, but is far more secure. Other protocols can be “tunneled” through SSH to make them secure.
  o HTTP/HTTPs – hypertext transmission. This appeared in the early 1990’s and rapidly evolved into a way of projecting a wide range of graphic user interfaces across the internet. The “s” means secure/encrypted. HTTP is a much easier and more secure way to do arbitrary things on a remote user’s screen than making them run custom software.
  o SMB, NFS – file sharing. Making disks on a distant computer look like they’re on yours
  o SMTP – sending e-mail to and between mail servers (computers that can route e-mail). This is a “push” protocol: the computer initiating the connection sends the messages.
  o POP3, IMAP – retrieving mail from e-mail servers. These are “pull” protocols: the computer initiating the connection receives the messages (if there are any)
  o And many, many, more…

• Typically, each protocol uses a single TCP port (or perhaps a few). For example, HTTP usually uses port 80, and SSH usually uses port 22.

Some more notes on layers and protocols
• As you move downwards in the protocol layer “stack”, more and more “bookkeeping” data — also called “headers” — get appended around the data you are sending. This means the actual number of bits transmitted can be substantially more than you might think. Header information may get appended to each packet, if the message is divided into packets.
• TCP and IP usually go together and are known as “TCP/IP”
• You can run more than one network layer on top of a physical layer on the same link (for example, IP and AppleTalk)
You can run several transport layers on top of a network layer (for example, TCP and UDP on top of IP)
And, of course, you can run many application layers on top of a transport layer (SSH and HTTP on top of TCP)

Kinds of networks
- LAN – “Local Area Network” on the level of a single building
- WAN – “Wide Area Network” a vague term for something larger than a LAN
- Enterprise network – a larger-than-LAN network dedicated to a particular company or organization
- Internet – multiple networks networked together
  - The idea of an internet preceded the current notion of the internet – “the” internet happened when most things got connected!
  - The “IP” network layer was specifically designed to make it easy to create internets. That is why “the” internet could grow so quickly in the 1980’s and 1990’s, and conversely why TCP/IP is now the dominant network layer.
- VPN – “Virtual Private Network” – runs over the internet but encrypted in such a way that it looks like a private WAN that outsiders can’t snoop on (we hope!)

Current network technology
- Most firms now have LANs implemented with copper wire, usually Ethernet, and now also building-level wireless
- Many larger firms have WANs and/or enterprise networks containing wire and/or fiber and maybe some satellite/microwave (depending on the firm’s size). The longer links in these networks are typically leased from ISP’s (see the next item)
- Internet service providers (ISP’s) maintain interconnected, overlapping networks made primarily of fiber (examples: AOL, ATT, Sprint, etc.) ISP’s also lease capacity for use in enterprise networks. Large and medium firms connect directly to ISP’s.
  - Also, there are some non-profit alternatives to ISP’s, like “Internet2” which serves large universities like Rutgers
- Large firms can afford to lease dedicated high speed connections to ISP’s, like “T3” lines
- The dreaded “last mile”: smaller firms and individual households connect to the ISP’s in various non-ideal ways:
  - By phone and modem (sometimes directly to employer instead of ISP)
  - Cable modem – signals carried over the same coaxial cable that distributes TV signals. Capacity usually 0.5-6 MB/s, but capacity may be shared with other users in the neighborhood
  - DSL – signals carried over regular phone lines, but not at audible frequencies. About 0.5-1 Mb/s, but occasionally faster. Only works if you are within 2 miles of telephone switching center, but does not have capacity sharing problems.
- Most network connections carry a fixed charge per month, without tracking the exact number of bits sent – one reason we have so much “spam”!

Uses for networks
- Sending messages and data between people by “push”: e-mail possibly with attachments, instant messaging, voice over IP (VoIP) telephone
- Sharing/disseminating information by “pull” (basic web, FTP). Computers that are “pulled” from are usually called “servers”
• Other modes of sharing data. Some (or all) computers hold data that other computers can share.
  ○ Computers that share data on their disks are often called “servers” or “file servers”.
  ○ An example: “network drives” disks that are not on your computer, but act like they are (if a little slowly)
• Sharing other hardware like printers, scanners (these actually contain processors)
• Combination push/pull messaging and sharing: chat rooms, newsgroups
• Specific teamwork applications
  ○ Calendar/scheduling applications
  ○ Joint authorship systems (Lotus Notes?)
• Gathering data and interacting with customers
  ○ Websites that gather data or take orders
  ○ Sensors and scanners
• Offsite backup (this used to be done with tapes, but they are so slow now compared to hard disks)
• And many, many, more…

Data transfer calculations
• Calculate size of data to be moved
• Divide by the speed of the transmission line
• Remember:
  ○ File sizes are commonly in bytes, with binary-style K, M, G etc.
  ○ Transmission line speeds are usually in bits per second, with decimal-style K, M, G etc.
  ○ This mismatch is annoying, but unfortunately, it is the common convention.
  ○ It’s easiest to convert the file size to decimal bits, and then divide.
  ○ The transmission protocol will probably add header information that will cause the real download to take longer
  ○ Network congestion or malfunctions could cause even more delays

**Sample file transfer calculation:** Suppose we want to do “video-on-demand” downloads of 4 GB movies in DVD format (binary-style GB). How long would that take over a 1 Mb/s DSL line?

Size of movie = (4 GB)(1024 MB/GB)(1024 MB/KB)(1024 B/KB)(8 bits/B) = 3.44 \times 10^{10} \text{ bits}

Seconds to transfer with DSL = (3.44 \times 10^{10} \text{ bits})/(1 \times 10^6 \text{ bits/sec}) = 3.44 \times 10^4 \text{ sec}
= (3.44 \times 10^4 \text{ sec})/(60 \text{ sec/min} \times 60 \text{ min/hr}) = 9.54 \text{ hours – probably not acceptable!}

Note that actual transfer times would be somewhat larger due to overhead (headers) added by the application, transport, network, and physical network layers
Class 14 – Brief Discussion of E-Commerce

This mini-lecture relates to Chapter 5 of the text, which attempts to categorize all manner of e-commerce. I will take a different, quicker approach, and try to categorize what I view as the key aspects of successful e-commerce.

In the 90’s, there was a “dot-com boom”. During this boom, many people was assumed that the internet would take over the entire economy and people trying to sell products like dog food over the web would become billionaires. We have since returned to reality. But the internet has had and continues to have a huge impact. A few companies, like Amazon, E-Bay, and Google have been spectacularly successful, and many smaller companies have done well.

I believe the key aspects of successful e-commerce are:

- Making it easier to locate other parties with which to have an economic transaction
- Making it easier to identify desired products and services
- Decoupling the information aspects of a transaction from the physical transfer of goods
- Reducing the costs of communication between economic parties
- Promoting pull-based communication between economic parties

Note that these categories are interrelated and overlapping.

Making it easier to locate parties (firms or individuals) with whom to conduct transactions:

- If I am looking for obscure cross-country ski accessories, I can now locate them quickly even if I live in Alabama. A simple web search will probably identify dozens of firms selling the sort of thing I am looking for. Before the internet, I would have had to go the library and search through national business directories, or try my luck with 800 directory assistance.
- E-Bay’s core business is helping buyers and sellers of possibly obscure used items find one another easily. If I have a very specialized item, services like E-Bay allow prospective buyers from all over the world to find me, instead of hoping that one of them will drive by my yard sale.

Making it easier to identify desired products and services

- Suppose we don’t know what particular product/service we want. It is now relatively easy to
  - Search the entire e-commerce world using a search engine like Google
  - Search within particular suppliers’ websites

Decoupling information and physical aspects of a transaction

- A bookstore or music store stocking all the titles carried by Amazon would be impossibly large.
- We do not have to depend on having goods automatically distributed to our local area before we make a purchase decision. It is now possible to purchase goods not popular in one’s geographical area, and to have much wider choice of products and suppliers than if we depended wholly on local stores.
- For physical goods, online commerce is dependent on an efficient parcel delivery system. Parcel delivery underwent a revolution in the 1980’s with the formation of Federal Express, and information technology was critical in its growth. It has grown symbiotically with web commerce. Parcel shipping still costs more per unit than bulk
shipping to retailers, but allows inventory to be kept in a more centralized, more efficient manner. Thus, the overall “supply chain” may be more efficient in many cases.

- Physical stores and showrooms are not going away, though
  - Physical examination of goods very important for some products
  - Immediate delivery can have enormous value

Reducing communication costs

- A supplier can now update pricing and availability information and transmit it into our firm’s information system as frequently, without human intervention
- Our customers can query flight times, inventory availability, pricing etc. without our having to dedicate so many people to physically talk to them
- Customers can lengthily custom-configure products (like computers) without consuming a lot the supplier’s labor

Pull-based communication

- When communication is initiated by the potential consumer of a good or service
- Internet technologies greatly reduce the cost and human effort required for pull-based communication
- Successful e-commerce tends to contain a large “pull” element. Even effective internet advertising (as on Google) is based on determining a level of customer interest, based on their query
- The internet is also tempting for push-based communication because of very low costs. But the results are frequently irritating: spam and pop-up ads.

I believe e-commerce have the potential to make it possible for market economies to function efficiently with smaller, more entrepreneurial, less internally politicized economic units. But it is not clear whether that will happen.