Research Computing

Contents

Chapter 1

Introduction

This resource contains teaching materials for an overview course in **research computing**. This course offers a broad overview of general computing concepts designed to support research activities in a university setting.

1.1 Audience

The target audience is primarily college students, particularly graduate students, who conduct academic or scientific research. The information will also be useful for working professionals.

1.2 Course Goals

The main goal of this course is to help students improve their technical readiness for engaging in the increasingly data-centric work that they face in their degree programs, in their related research, and in their future professional and research practice. Students will become acquainted with key concepts in research computing and data management, including overviews of systems analysis techniques, data security and integrity, and database tools, among others.

1.3 Learning Objectives

At the end of this course students should be able to:

- Analyze requirements for management of data in different situations and projects.

- Choose appropriate technical tools and techniques to support that data management.

- Identify hazards and pitfalls in data-related projects.

- Identify factors that affect performance in the collection and preparation of data for analysis.

- Describe the core technologies most frequently employed in large-scale research data management.

1.4 File Contents

The course materials primarily consist of the presentation slides in Markdown format and the transcripts which go along with those slides. The transcripts are posted as wiki pages in ASCIIDoc format and are also offered as PDF and EPUB eBooks. These materials may be found online at: https://github.com/brianhigh/research-computing.

1.5   The Research Computing Team

Thanks to the following people who have contributed to this resource (in no particular order):

Brian High, Jim Hogan, John Yocum, Elliot Norwood, and Lianne Sheppard

1.6   Copyright, License and Disclaimer

Chapter 2

Information Systems

Your research computing experience will involve using *and* developing information systems. We will take a quick look at the various components, types, and development models of these systems.

2.1   Information System Components

The primary components of an information system[1] are hardware, software, data, and *people* — the most important component of all! Why? Because systems are designed and built *by* people *for* people. If people don't use them, or they do not serve the people's needs, then they are worthless! Today we will take a closer look at how information systems are designed to help us.

Figure 2.1: Exploded view of a personal computer - Image: Gustavb, CC BY-SA 3.0 Unported

---

[1] *Information system - Components*, Wikipedia, CC BY-SA 3.0

2.2   Hardware

The physical machinery of a computer system is called its hardware. Of course, this means the computer itself, its chassis and the parts inside it, including its core integrated circuit known as the central processing unit (CPU), as well as its memory, called "RAM" or Random Access Memory, and any internal storage devices like hard disk drives (HDD) and solid state devices (SSD).

Accessories or peripherals are the devices you plug into the computer, mostly for input and output.

Networking equipment includes all of the devices that allow your computer to communicate with other systems. Examples are the network cables and the boxes they connect to, such as routers, switches, hubs, wireless access points, and modems.

2.3   Software

Software is the name for the instructions we give to computing devices to tell them what to do. Software is "soft" because the instructions are not physical entities like hardware devices. The instructions may be stored on physical media like a hard disk or USB thumbrive, just as a cooking recipe may be written on a piece of paper or printed in a book. However, the recipe itself is just a *conceptual model* of how to perform a task. Likewise, a software program is essentially just a list of instructions (or a *logical model* that issues instructions) for the execution of a set of desired computing operations.

2.3.1   Application Software

As you use a computer, the software instructions that are executed on your behalf by the CPU, such as programs and apps, are called application software. Applications are the programs that serve a specific purpose for a computer user or are to be used for completing certain tasks, such as exploring the Internet, editing a text document, or working with data.

2.3.2   System Software

2.3.2.1   The Operating System

Applications run within a *overall* software environment called the operating system (OS).

Notable examples are the familiar Microsoft Windows, OS X, iOS, Android and Linux operating systems.

2.3.2.2   Kernel, Drivers, and Firmware

An operating system also has a kernel, which is the central software program that manages the data exchange between the CPU and the other components within a computer. The kernel communicates with those components using device drivers, which are small programs that provide a software interface to the hardware. Devices that contain integrated circuits of their own may store software in firmware that allows updates through a procedure called flashing. The computing system will also contain utility software such as configuration and management tools, plus shared software libraries used by both applications *and* system software.

2.4   Data

Data refers to all of the information in the system. It may be stored as raw (unprocessed) values, or may be in the form of summary tables, plots, written documents, photographs, music, videos, or just about any other form of information which can be digitized. Data can be *at rest*, in which case it will probably be saved in some sort of file or may just be occupying some bits of system memory. Data may also be *in motion*, flowing between the *components* within a single computer system or between *nodes* of a network. As the boundaries of an information system will usually extend beyond computer systems, data may also reside on scraps of paper or may only exist in a person's mind in the form of a thought or idea, waiting to be communicated to the rest of the information system.
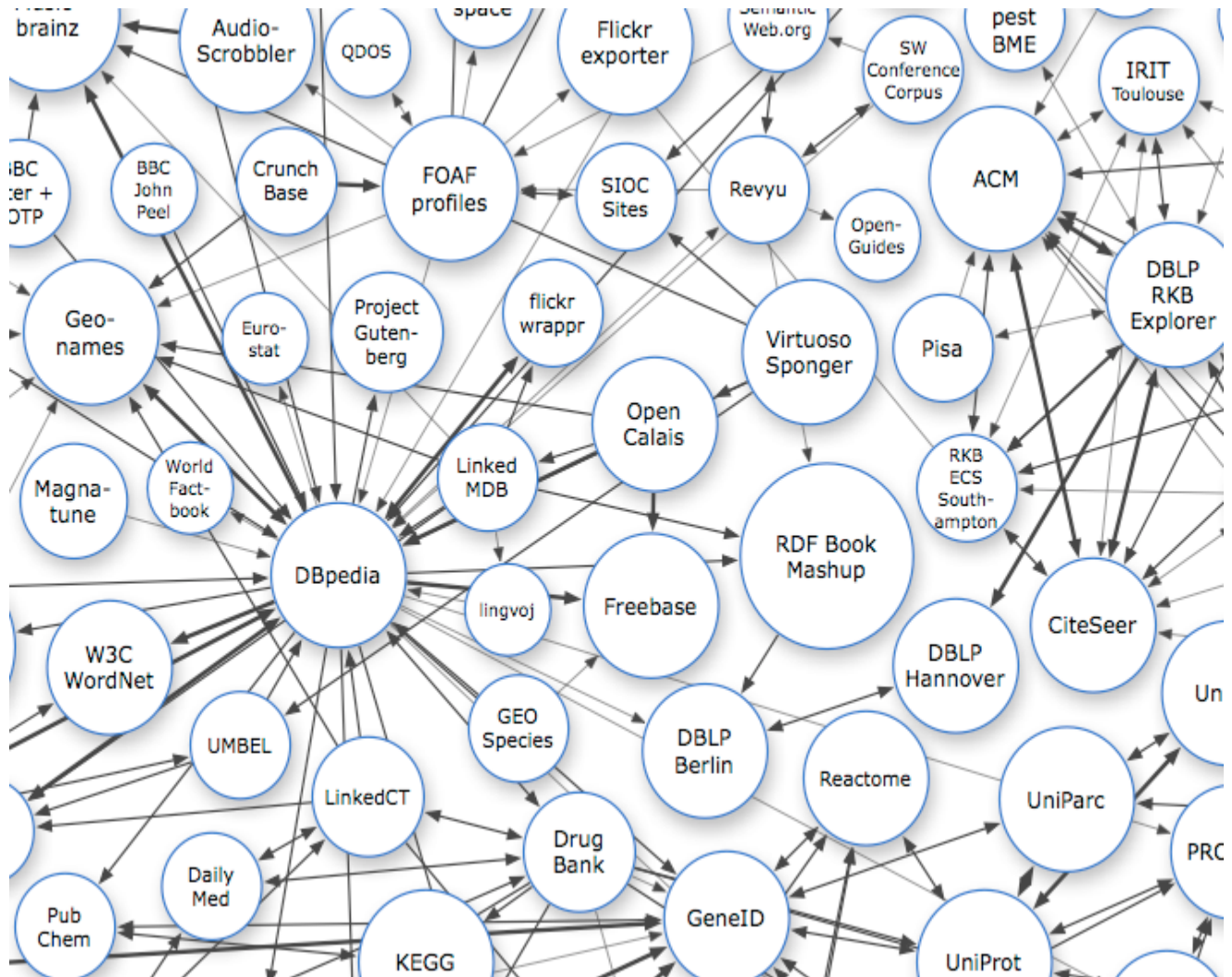

Figure 2.2: Linked Data - Image: linkeddata.org, CC BY-SA 3.0

2.5   People

People are an integral part of the system. We design it, build it, use it, maintain it, and adapt it to new uses. Our systems should serve *us*, not the other way around. Every aspect of the system should be designed to serve people's

needs *optimally*. But our needs vary, and so we need various types of systems.

Figure 2.3: Pair Programming - Image: Ted & Ian, CC BY 2.0 Generic

2.6 Information System Types

We may use several types of information systems each day. Let's take a quick look at a few of the most common types.[2]

Most of us are very familiar with *search information systems* like web search engines, such as Google Search, but many sites use domain-specific search engines like PubMed.

Spatial information systems in the form of Geographic information system (GIS) have become increasingly important in recent years. ArcGIS has dominated this field, with the free and open QGIS gaining in popularity.

Global information systems (GLIS) are those either developed or used in a global context. Public health examples include global health databases such as the UNHCR Statistics & Operational Data Portals and the WHO's Global Health Observatory (GHO).

Enterprise systems are comprehensive organization-wide applications used for Enterprise Resource Planning (ERP).

Expert systems support such specialty domains as diagnosis, forecasting, and delivery scheduling. They use artificial intelligence to apply knowledge and reasoning in order to solve complex problems.[3]

Office automation systems refer to systems which support the everyday business operations of an organization. Business Process Automation (BPA) uses these systems to improve efficiency by streamlining routine activities.

---

[2] *Information system - Types of information system*, Wikipedia, CC BY-SA 3.0

[3] For more information about expert systems for public health, see "Decision Support and Expert Systems in Public Health" by William A. Yasnoff and Perry L. Miller, *Public Health Informatics and Information Systems, 2nd Edition*, pp. 449-467 (Springer, 2014). The chapter uses *IMM/Serve* as an example. See also: IMM/Serve: a rule-based program for childhood immunization. P. L. Miller, S. J. Frawley, F. G. Sayward, W. A. Yasnoff, L. Duncan, D. W. Fleming Proc AMIA Annu Fall Symp. 1996 : 184–188. http://www.ncbi.nlm.nih.gov/pmc/articles/-PMC2233221/.

Personal *information systems* help people manage their individual communications, calendaring, note-taking, diet, and fitness.

2.7   Software Development Process

Developers undertake the software development process using several different approaches. Let's take a look at a few of the most popular models.
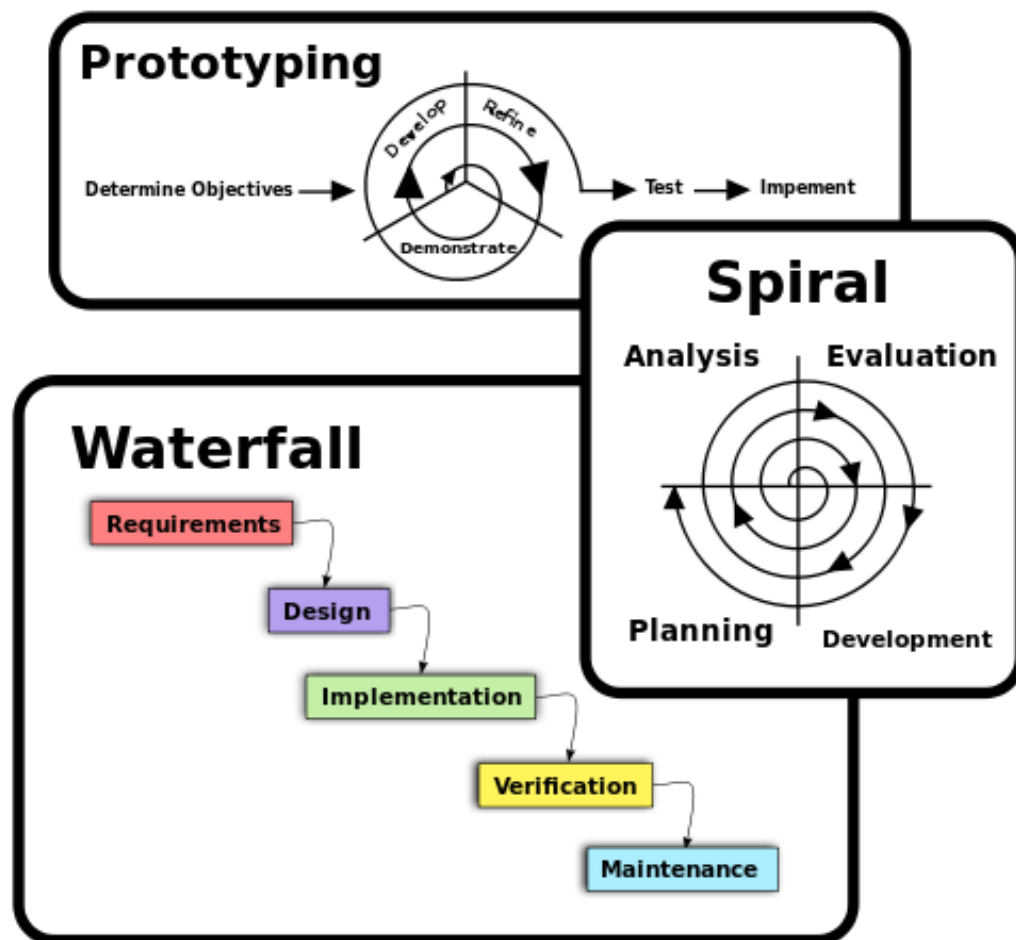
Figure 2.4: Three software development patterns mashed together - Image: Beao, Paul Smith, Public Domain

Here is a short list of common development models.[4] We have provided links from each of these to relevant Wikipedia pages. You are encouraged to read more about them. We'll just go through the list quickly to give you a rough idea of the differences between them.

The Systems development life cycle (SDLC) is the classic model. It involves lots of up-front planning and is risk averse.

---

[4] *Software development process*, Wikipedia, CC BY-SA 3.0

Waterfall development is another "old school" favorite, It's like the SDLC but does not offer any sort of feedback loop.

Prototyping is a useful technique for many models. Good when a small-scale experiment can prove an idea without risking heavy investment.

Iterative and incremental development might evoke the image of "baby steps" or the notion of "try, try, again". There is a central loop, between initial planning and final deployment, which repeats as needed. Like prototyping, it is a technique which can be used in other models.

Likewise, Spiral development is meant to address evolving requirements through cycles of repeated analysis and design, getting closer and closer to the desired product. The idea is that the *entire process* is repeated over and over until you are finally satisfied.

Rapid application development (RAD) focuses on development more than up-front planning.

Agile development is a more evolved form of RAD, with more of a focus on user engagement, and gaining wide popularity.

Code and fix sounds like what it is — *cowboy coding* — what most lone programmers do, and what might seem most familiar to you as a scientific researcher. This can be quick for easy projects, but can be very inefficient and expensive for larger projects, due to insufficient planning.

They are all useful methods, though, some more generally than others. The approach you take should depend upon your situation.

We'll look more closely at three of these right now.

2.7.1   Systems Development Life Cycle

Since information systems are so complex, it is very helpful to follow a standard development model to make sure you take care of all of the little details without missing any.
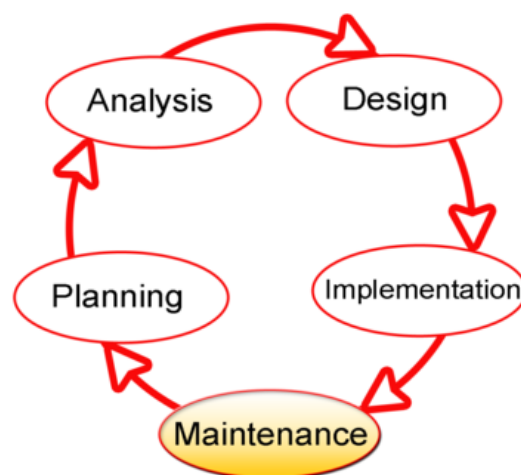

Figure 2.5: SDLC - Image: Dzonatas, CC BY-SA 3.0

For years, the standard development model was known as the SDLC, or Systems Development Life Cycle.[5] It works well for large, complex, expensive projects, but can be scaled down as needed. Many of its phases are used in the other models as well. Let's take a quick look at them.

Systems development life cycle (SDLC) phases:

- Planning (feasibility study)

  – There is a focus on careful *planning* before any design or coding takes place. The feasibility study explores your options and gains approval from stakeholders.

- Analysis

  – *Analysis* includes a detailed study of the current system and clearly identifies requirements before designing a new system.

- Design

  – Once you have thoroughly defined the requirements, you can begin to model the new system.

- Implementation

  – *Implementation* is where the hardware assembly, software coding, testing, and deployment takes place.

- Maintenance

  – *Maintenance* may sound boring, but it is essential to ensure that the project is an overall success.

The main idea is that systems development is a cycle — a continual process. You need to allow for maintenance, updates, and new features. The use and upkeep of the system provides feedback which goes into planning the next version.

We will spend more time on the SDLC and its early phases in a separate module.

2.7.2 Waterfall Model

A related model is the Waterfall model.[6] It has basically same same steps as the SDLC, but visualizes them as cascading stair-steps instead of a circle.

---

[5] *Systems development life cycle*, Wikipedia, CC BY-SA 3.0
[6] *Waterfall model*, Wikipedia, CC BY-SA 3.0

Figure 2.6: Waterfall model - Image: Peter Kemp / Paul Smith, CC BY 3.0

It's basically similar to the SDLC, but without the feedback loop. There are cascading stair-steps, where one phase leads to another and the output of one phase becomes sthe input of another. Its came from manufactoring where after-the-fact changes are expensive or impossible.

2.7.3 Agile Model

The Agile model is a newer development model that is especially popular among smaller teams within budding organizations. Hallmarks of this model include methods such as pair programming, test-driven development, and frequent product releases.[7]

---

[7] *Agile software development*, Wikipedia, CC BY-SA 3.0

Figure 2.7: Agile Software Development methodology - Image: VersionOne, Inc., CC BY-SA 3.0

Agile development is characterized by smaller teams that can meet regularly, ideally face-to-face, working in pairs, with one person coding and the other helping "over the shoulder". After you identify use cases, then you write tests and then build the system to pass the tests. By developing an automated test and build system, releases can be pushed out quickly and more often.

2.7.4   Transparency

Information systems vary in the openness of the their implementations, in terms of both interoperability standards and specific design details.[8]

You can have *open* systems (and standards, source), where the technical specifications are publicly available.[9] Different organizations may implement them in their own way, yet still maintain interoperability with other implementations.

Or systems may be *closed*, or proprietary, where an organization keeps the details to itself, making it more difficult for competitors to inter-operate. While this may provide a competitive advantage for the producer it contributes to what is called vendor lock-in, where a consumer becomes dependent on the vendor, unable to switch to another due to the high costs and disruption.

Interoperability aspects will include file formats, communications protocols, security and encryption. These become especially important when you are collaborating, sharing data and files with others, who might be using different platforms.

By using transparent systems, you not only increase your ease of communication and collaboration, you also contribute to openness in a broader, social context.

Information transparency supports openness in:

- Government

- Research

- Education

- Courseware

- Content

- Culture

We have provided links (in the list above) to several popular movements which are working to increase openness and transparency in various aspects of society. You are encouraged to spend some time learning about these trends.

So, if you want the benefits of openness in your work and more freedom to make changes, consider building your information infrastructure with open technologies.

2.7.5   Transparency Example: This Course

As an example, we have assembled a transparent information system to create and support this course.

We have developed the course transparently, using an open content review process where students, staff and faculty look at the materials and evaluate them to determine whether or not they best meet the course goals.

We have an open content license, the Creative Commons Attribution Share-Alike CC BY-SA 4.0 International license.

We have open development where our source is freely and publicly available on GitHub).

We are using open file formats (Markdown, HTML, CSS, PNG, AsciiDoc, PDF), open source tools tools (RStudio, Git, Redmine, Canvas, Linux, Bash) and open communications protocol standards (HTTP/HTTPS).

As you take part in this course, and provide feedback which will go toward improving it, we thank *you* for contributing!

---

[8] *Openness*, Wikipedia, CC BY-SA 3.0
[9] *Software standard*, Wikipedia, CC BY-SA 3.0

2.8   Wrap-Up

We hope that this brief overview of Information Systems has given you a more clear picture of what they are and how they are built.

For more information, please read the related sections in the Computing Basics Wiki, particularly, the pages on hardware and software.

In the next module, we will take a closer look at requirements gathering and systems analysis, two of the most important topics of this course.

Chapter 3

Systems Analysis

Investing time and money into a computer or information system without a clear course of action can be expensive and wasteful. By taking some time to fully consider the issue at hand and pursue a disciplined approach to finding a practical solution, you greatly increase your odds of success and decrease costs. We call this process *Systems Analysis*.

3.1 Systems Development Life Cycle

Systems analysis is an important part of an overall approach to *systems development*.

The life of an information system follows a cycle. The classic development model is called the Systems Development Life Cycle, or SDLC.[1]
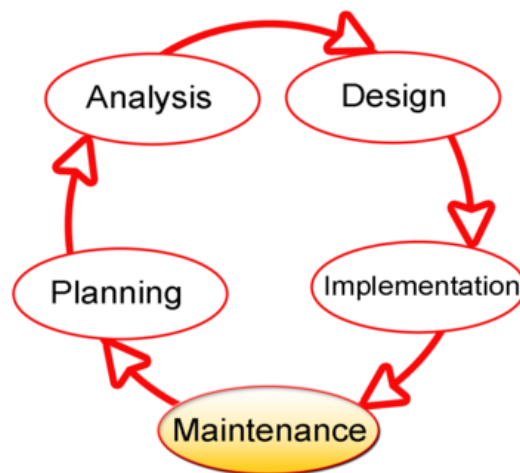

Figure 3.1: SDLC - Image: Dzonatas, CC BY-SA 3.0

---

[1] *Systems development life cycle - Phases*, Wikipedia, CC BY-SA 3.0

3.1.1   Planning Phase

The Planning phase defines the primary issue (*problem* or *goal*) and performs a feasibility study. Here, you clarify the project scope, compare your best options, and come up with a plan.

3.1.2   Analysis Phase

The Analysis phase focusses on the issue, defined previously, and studies its role in the current (or proposed) system. The system is explored, piece by piece, in light of the project goals, to determine system requirements.

3.1.3   Design Phase

In the Design phase, a detailed model of the proposed system is created. Various components or modules address each of the requirements identified earlier.

3.1.4   Implementation Phase

During the Implementation phase, a working system is built from the design and put into use.

3.1.5   Maintenance Phase

The Maintenance phase includes ongoing updates and evaluation. As changes are needed, the cycle repeats with more planning, analysis, and so on. We will take a closer look at the **systems analysis** phase next.

3.2   What will you need?

Essentially, in *Systems analysis* we answer the question, "*What will you need* to reach your goal?" In other words, **"What are your requirements?"**

A **primary goal** of this course is to help you develop your skills in *requirements analysis*.

Systems analysis helps you **clarify your project needs** and **plan ahead** in order to *obtain and allocate* **critical resources**.

The main idea here is . . .

> If you don't *know* what you *need*, how can you *ask* for it?

3.3   Systems Analysis

After completing an initial feasibility study to "determine if creating a new or improved system is a viable solution", proposing the project, and gaining approval from stakeholders, you may then conduct a systems analysis.

Systems analysis will involve . . .

> breaking down the system in different pieces to analyze the situation, **analyzing project goals**, breaking down what needs to be created and attempting to **engage users** so that **definite requirements** can be defined.[2]

---

[2] *Systems development life cycle - System investigation*, Wikipedia, CC BY-SA 3.0

3.4   Systems Analysis Definition

So, what, exactly, *is* "Systems Analysis"?

- "A *system* is a set of interacting or interdependent components"[3]

- *Analysis* means "to take apart"[4]

Putting these two ideas together, we have:

> Systems analysis is a problem solving technique that decomposes a system into its component pieces for the purpose of the studying how well those component parts work and interact to accomplish their purpose.[5]
>
> — Lonnie D. Bentley *Systems Analysis and Design for the Global Enterprise*

We perform this analysis by working through a series of five phases.

3.5   Systems Analysis Phases

Systems Analysis[6] has its own series of phases. We will look at each of these one by one.

---

[3] *System*, Wikipedia, CC BY-SA 3.0

[4] *Systems analysis*, Wikipedia, CC BY-SA 3.0

[5] *Systems Analysis and Design for the Global Enterprise 7th ed.*, by Lonnie D. Bentley, as quoted by Wikipedia

[6] *Systems analysis - Information technology*, Wikipedia, CC BY-SA 3.0
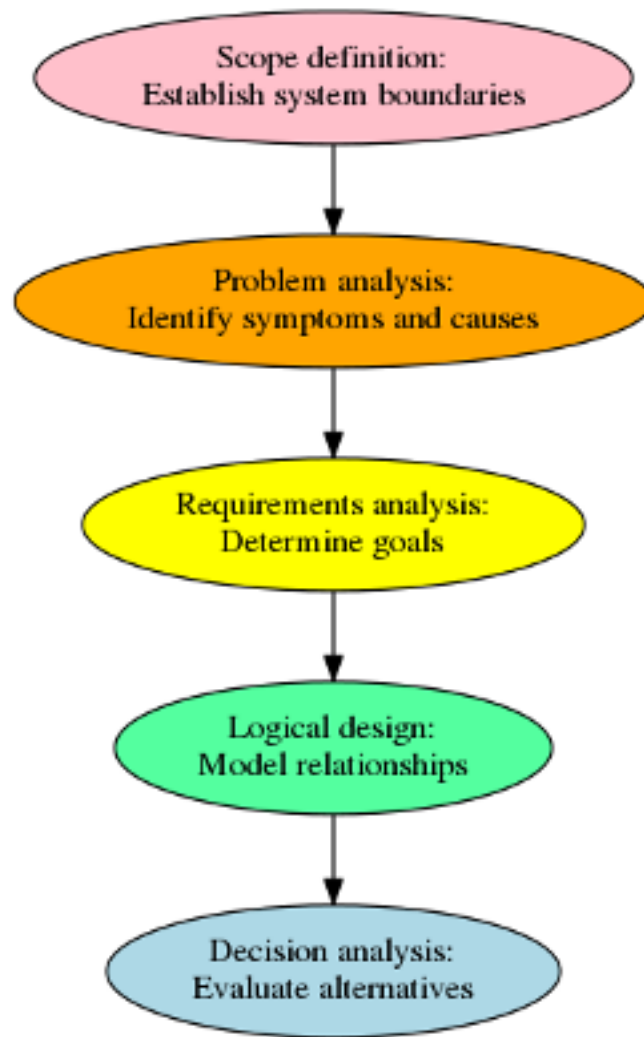
Figure 3.2: Systems Analysis Phases

During Scope definition, we establish our system boundaries. In our Problem analysis phase, we identify the symptoms and causes. Our Requirements analysis determines our system goals. The Logical design phase models relationships within the system. And our Decision analysis evaluates the various alternatives.

3.6 Scope Definition

So, do we mean by *scope*?[7]

> **Scope** involves *getting information* required to start a project, and the *features* the product would need to have in order to meet its *stakeholders requirements*.

> — Wikipedia *Scope (project_management)*

---
[7] *Scope (project management)*, Wikipedia, CC BY-SA 3.0

Put another way, *project* scope defines the *work to be done* whereas *product* scope is concerned with the *desired features and functions*.

By being careful to define scope *early on*, we can be more watchful for *scope creep*.[8]

> **Scope creep** is [. . .] the *incremental expansion* of the scope of a project [. . .], while nevertheless *failing to adjust* the schedule and budget.
>
> — Wikipedia *Scope (project_management)*

3.7 Problem Analysis

Problem analysis is critical. When we say, *problem*, we can also think *goal*, *research question*, or *issue*. If you don't get this right, you can waste a lot of time and money solving the *wrong problem* or address a *non-issue*.

We can summarize the key points of problem analysis[9] as:

- Define and clarify the problem (or issue)

  - *What exactly are we trying to solve?*

- Determine the problem's importance

  - *How much does it matter?*

- Assess the feasibility of solving the problem

  - *Do we have the resources we need?*

- Consider any negative impacts (unintended consequences)

  - *What could go wrong?*

- Prioritize problems to solve (bottlenecks? low-hanging fruit?)

  - *What are the most critical issues?*

- Answer: *what*, *why*, *who*, *when*, *where*, and *how much*?

  - *Drill down into the problem with all kinds of questions to expose dependencies.*

- Find **causes** (especially root cause) and **symptoms** (effects)

  - *What is really going on here?*

---

[8] *Scope (project management)*, Wikipedia, CC BY-SA 3.0

[9] Jenette Nagy, *Defining and Analyzing the Problem*, Analyzing Community Problems and Designing and Adapting Community Interventions, Kansas University, CC BY-NC-SA 3.0 US

3.8   Root Cause Analysis

There are several methods of root cause analysis, but one simple one is "Ask Why Five Times" . . .

Root Cause Analysis Tree Diagram

Figure 3.3: Root Cause Analysis Tree Diagram - Image: KellyLawless, CC BY-SA 3.0 Unported

. . . where you keep asking "Why? But, why? But, Why?" over and over again until there is a clear root cause that you can actually do something about.

A real example is mentioned in Ask Why 5 Times, Business Analysis Guidebook/Root Cause Analysis (wikibooks.org), where the US National Park Service was able to slow the rate of deterioration of the Jefferson memorial simply by changing the lighting schedule.[10]

This method may also be used in Requirements Analysis.

3.9   Requirements Analysis

• Elicit, Analyze, and Record (EAR):

---

[10] Ask Why 5 Times, Business Analysis Guidebook/Root Cause Analysis, Wikibooks

- System and project requirements

EAR Analysis is all about *listening*.

Gather requirements by interviewing stakeholders. Observe how people currently do their work or use the system. Make sure the requirements are detailed, clear, unambigious, and comprehensive. Document the requirements as a list, in diagrams, or narratives.[11]

As an example, in Spring 2014, we held a meeting with our graduate students and asked them about what sort of computing needs they had. We had a discussion, *listened* to what they had to say, summarized the main concerns, and documented them.

- Further elucidate Measurable goals

By continuing to ask questions like "Why? ... Why? ... Why? ...", get more specific until the goals become quantifiable. Measured goals are easier to meet since you can measure your progress in achieving them.

Mission objectives determine the goals. Compare the stated goals against mission objectives to produce a small set of critical, measured goals.

- Output: Requirements specification

This document will be used in the *logical design* phase to model the relationships within the system. So, it should be clear and thorough. The more specific and unambiguous this specification is, the easier it will be to design the system to meet the requirements.

One way to be more clear is to organize the requirements by type.

3.9.1   Requirements Categories

You may categorize requirements according to several important types:

Types of requirements:

Operational requirements are the **utility, effectiveness, and deployment** needs, with repect to practical use of the system within the overall operation. These are the **customer Requirements**.

Functional requirements are specific things the system must **do**.

Non-functional requirements are specific things the system must **provide**.

Architectural requirements define how the system must be **structured**, in other words, how the components must **interrelate**.

Behavioral requirements describe how **users** and other systems will **interact** with the system and how the **system** will respond.

Performance requirements define **how well** the systems needs to do things, **measured** in terms of quantity, quality, coverage, timeliness or readiness.

While there are other ways to group requirements, these are the most significant categories.

---

[11] *Requirements analysis*, Wikipedia, CC BY-SA 3.0

3.10   Requirements Modeling: Example

Another way to make requirements clear is through *requirements modeling*. Here is an example of modeling behavioral requirements with a Use Case Diagram.

Survey Data System

The behavioral requirements are stated in **role** *goal* format.

- **Researcher** *uploads survey*.

- **Subject** *takes survey*.

- **Subject** *uploads results*.

- **Researcher** *downloads results*.

Gramatically speaking, the role is the *subject* and the goal is stated as a *verb* and its *object*. Each of these requirements form the basis of a *use case*.

3.10.1   Use Case Diagram

The Use Case Diagram is a standard means of showing these requirements pictorially.
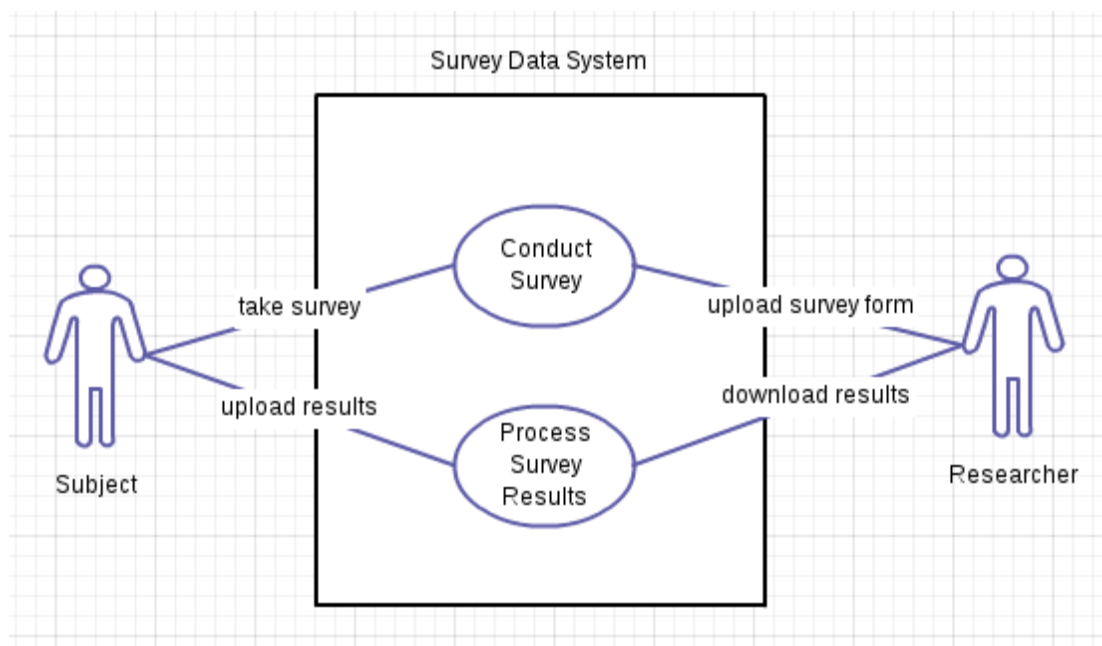
Figure 3.4: Research Survey Data System

Here, human *actors* are depicted as stick figures. These are the **roles**. The *goal* is the line connecting the role to a system activity or function, shown as an oval. In this example, the *system* is drawn as a box with the actors outside the box. The reason is that the system's *product scope* was defined to be the electronic data system. The human actors interact with that system. The information flowing to and from an actor, as well as data flows within a system, will be modeled in the *logical design*.

3.11   Logical Design

Once system behaviours have been modeled from the perspective of user interaction, we can begin to model the internal workings of the system with the *logical design*.[12] The *logical* part means that this is an *abstract representation* of the system. Therefore, the system is modeled in terms of abstractions such as *data flows, entities, and relationships*. We model how the system will satisfy functional requirements such as *inputs and outputs*. To make the abstract more concrete, we make use of *Graphical modeling* techniques to produce graphics such as the Data Flow Diagram (DFD) and Entity Relationship Diagram (ERD).

3.11.1   Example Entity Relationship Diagram (ERD)

For example, let's consider a system which records the playlists of musical performances at a local pub. You will want to store which artist performs which song. The relationship between an artist and a song can be shown in a simple Entity Relationship Diagram. We start with the behavioral *use case* written as **Artist** *performs Song*. Then, using a standard set of symbols, we can produce this diagram.
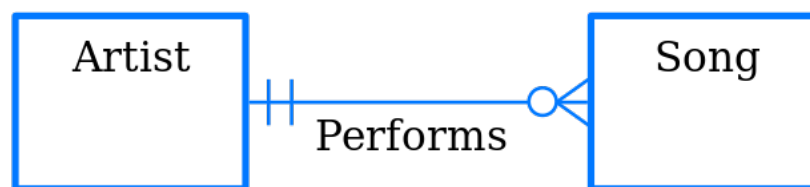
Figure 3.5: ERD: Artist performs song

The boxes represent *entities* and the line connecting them is the relationship. Here, the symbols indicate that there is a one-to-many relationship. This diagram is saying that exactly one artist performs one or more songs, (or doesn't perform any).

Since this "system" describes only solo performers, you would want to model it differently to include groups of artists that perform more than one song. You would want a many-to-many relationship as well as other entities to represent the musical groups. Since there will by many performances, you will also want an entity to represent the performances.

The value of this type of diagram is that complex relationships can be mapped to a great level of detail. In fact, they can be used to automatically generate database schemas.

3.12   Logical Design: Diagrams

Several examples of these diagrams can be found in the Systems Analysis and Design turorial.

This tutorial[13] (PDF, HTML, MP4) provides several examples from a fictitious public health research study.

You will find several other real-world examples from actual public health research projects in that course repository.

---

[12] *Systems design - Logical design*, Wikipedia, CC BY-SA 3.0

[13] See also: *Data Management*, UW Canvas.
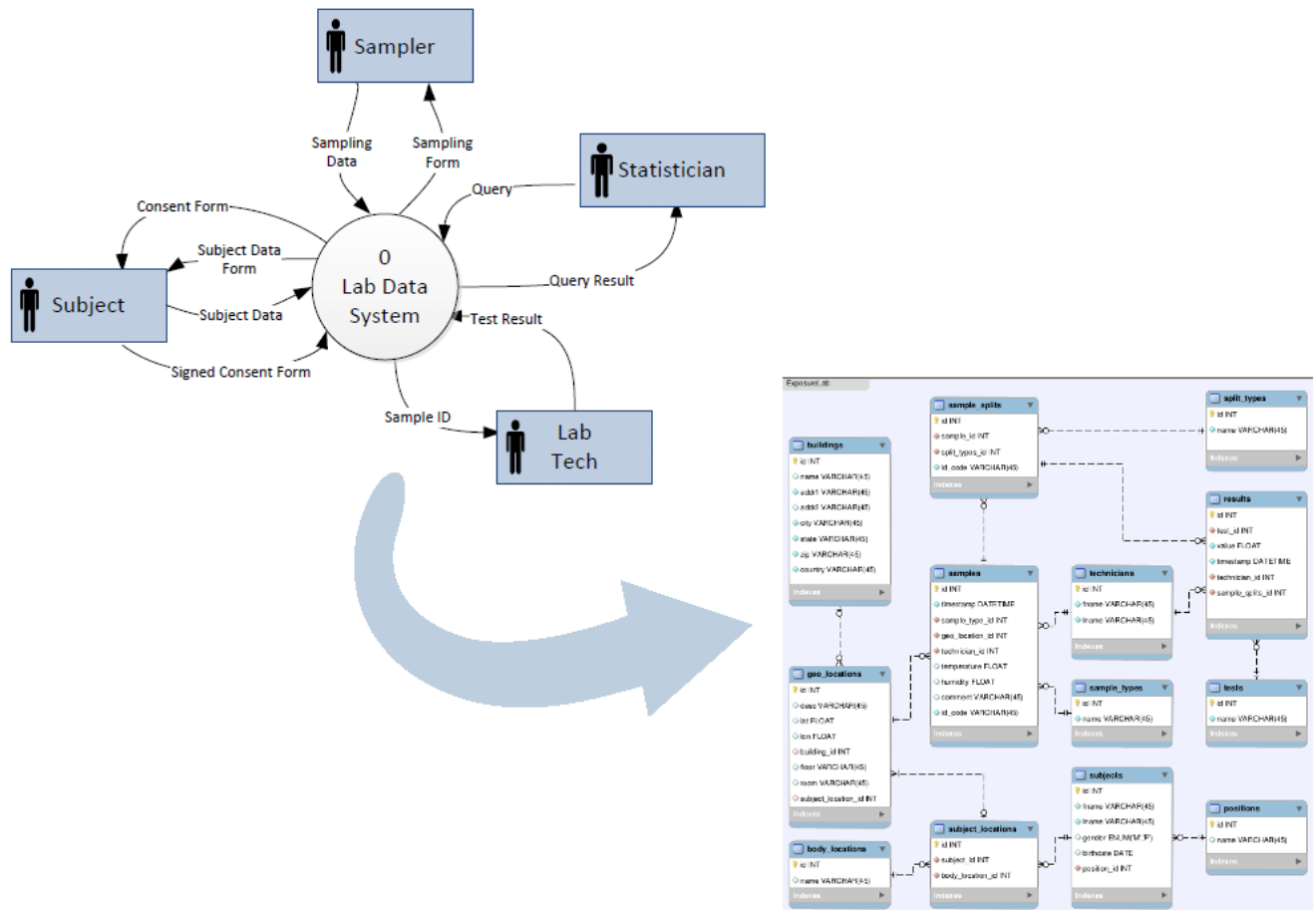
Figure 3.6: DFD and ERD

3.13 Decision Analysis

These models and diagrams will help you make a very important decision. Before you go further in the system design process, you need to decide whether to buy a system or build one. Or you may consider building a system of components, some of which you might buy and others might be custom built. You will want to come up with a few of the best alternatives and present them to your stakeholders. In your case that might be your lab manager, principal investigator, or funding agency.

Your presentation will also include a Decision analysis to weigh the pros and cons of the various options against the requirements in order to help the stakeholders with their decision. You should conclude your analysis with a recommendation of your top choice and explain why this choice is the most compelling. Then you will want to get a decision, and the approval to continue, before you invest any more time on further analysis.[14]

The system development life cycle (SDLC) continues on to other phases, which we do not have time to cover here. However, we hope that this glimpse at the systems analysis phase has demonstrated the value of this approach in gathering and clarifying requirements that can be used to design and build an information system.

---

[14] *Decision analysis*, Wikipedia, CC BY-SA 3.0

Chapter 4

Computer Networking

4.1 Introduction

In this day and age, it is rare to use a computer without some sort of network connection, so learning more about the ins and outs of networking is essential for scientific researchers and other computing professionals.

The intent of this presentation is to give you a somewhat better understanding of the basic underpinnings of computer networking as it relates to Research Computing and Data Management.

We'll cover a number of topics including a bit of history of networking, it's purpose, the evolution to contemporary computer networks, and many important technical aspects including networking topologies, protocols and standards, and networking system components.

4.2 Networking History
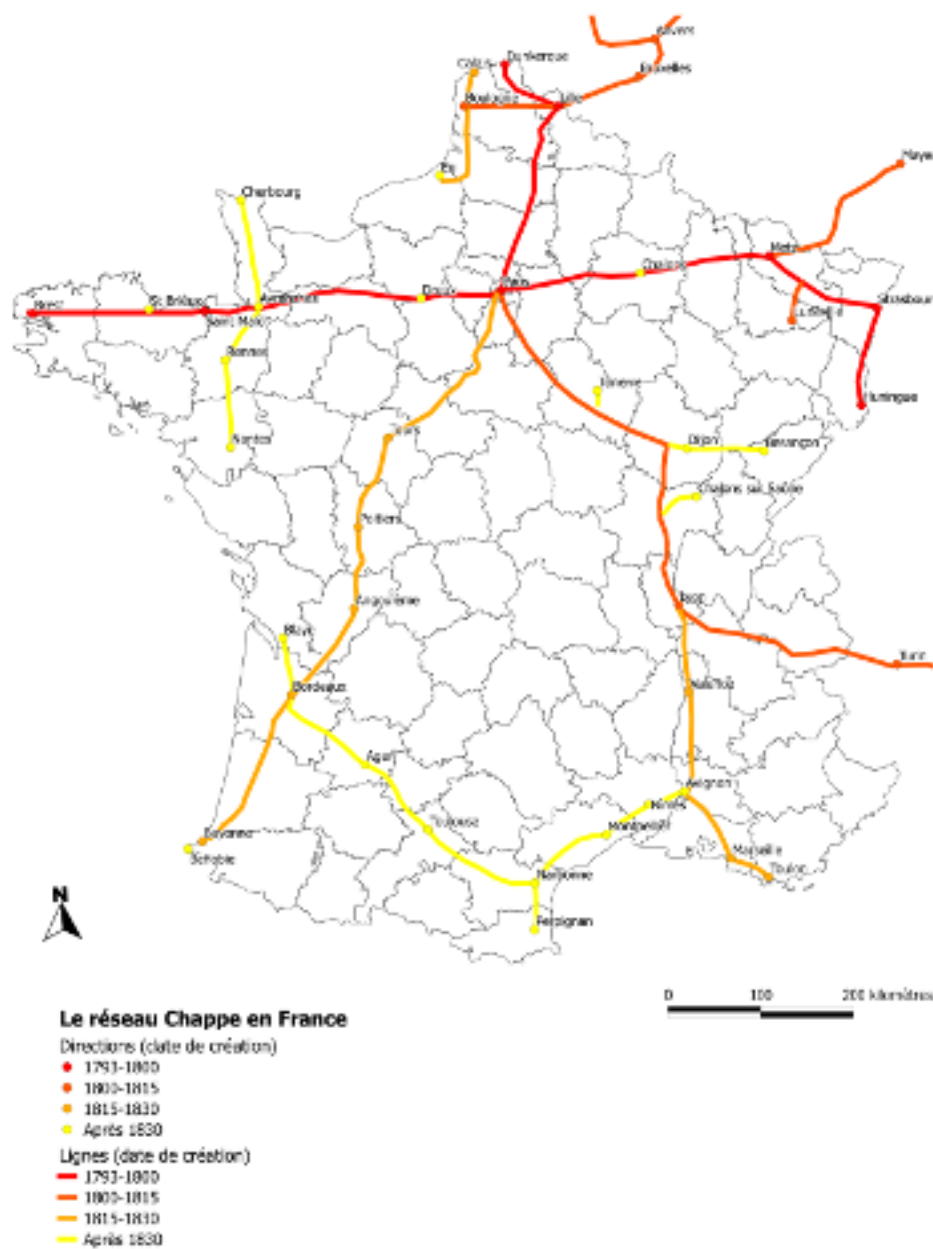
Figure 4.1: Chappe Network - Image: Jeunamateur, CC BY-SA 3.0

Here's an example of an early data network that spanned the length and breadth of France. They year was 1792 and the Chappe system used what were called optical telegraphs.
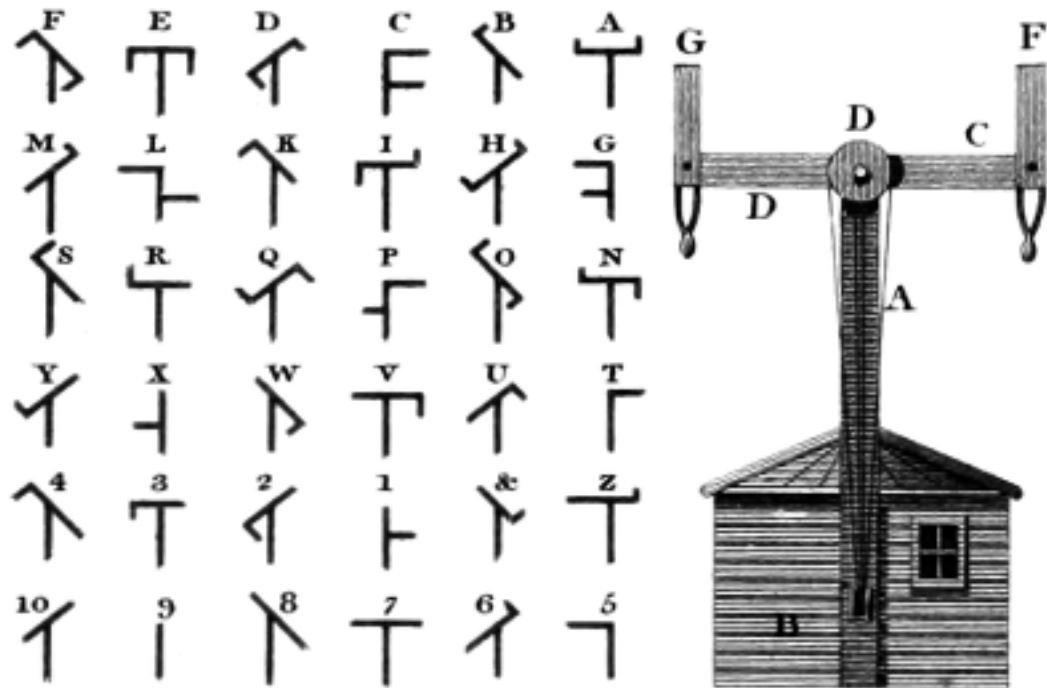
Figure 4.2: Chappe System - Image: Morgan Riley, Public Domain

---

The notion of a communications network goes back some time. Think of smoke signals or drums and later on semaphore and telegraph systems like Chappe's that connected European capitals to their frontier defense posts during Napoleonic times.

Fast forward a bit and you find yourself looking at the electrical telegraph and telephone all of which were technological improvements but which would be considered as "analog" technologies. Analog communications all suffered from problems like ambiguity — "Does that flag mean attack or retreat?" — and other limitations depending upon the technology. For example, a semaphore system might only be usable during periods of clear weather, and sound or smoke traveling over long distances might be misinterpreted due to audible or visual degradation.

4.2.1 Moving Along

From the industrial revolution to the early 1900s, things improved considerably. However, communications systems still suffered from the limitations of analog technology. While a tin can telephone is a far cry from the extensive telephone networks that followed, this 1943 switchboard photo depicts a system using the same fundamental technology.

Figure 4.3: Bell Switchboard - Image: The U.S. National Archives

4.2.2   Enter Binary

What we think of as a computer network really could not arise until certain preconditions were met and certain technologies discovered. For one it required the invention of usable computers that employed binary-based "digital" technology to perform calculations. Secondarily, it required a realization that binary/digital technologies could be employed in communication to set the stage for the emergence of computer networks and networking. Specifically, it wasn't until the 1960s that computer networks as we know them really started to take off with something called "packet switching".

4.2.3   Some Binary Basics

The essential elements in the binary world is that only two data states exist — 0 and 1 — yet these can be used to describe incredibly complex information. It's importance cannot be over emphasized. However, its simple use of zeros and of ones can be very difficult for normal mortals to cope with. Consider the following number. Do you recognize it?

```
128.95.112.1
```

Some of you may recognize this number as the address of a server at the University of Washington that provides so-called domain name services. .

How about this number?

```
1000000001011111011000000000001
```

In fact the first number simply a decimal notation representing the second binary number.

While speaking in binary won't make you the hit of the party it is important to remember that all of our contemporary communications rely completely on this underlying binary math.

Question

Would not Morse code be considered a binary communication protocol?

4.3   Computer Networks: What's the Point?

So, what do computer networks provide? First, they are a means for a computer system to communicate with another computer system or many computers systems. I intentionally say "computer system" rather than "computer" as it is often the case that it is a system running on a computer that is communicating to peer system, say a specialized system that performs bank transfer

Secondarily, networks allow humans to interact with computer systems.

And lastly, networks allow humans to communicate with other humans….. at least for so long as the computers do not rise up and prohibit this.

4.4   Contemporary Computer Networks:Key Elements

Several characteristics define modern computer networking as we know it today. First is its basis in the binary world that we just reviewed.

Second is a basis in something we have mentioned but not examined and that is "packet switching".

The third characteristic is what could generally be called "protocols".

One other crucial element in modern computer networking and applications that run on networked computers is the notion of "layers", We will review what is generally called a "layered model".

Other important elements in computer networking include software and hardware components, network topologies, network speeds, and a number of other elements.

4.4.1   Protocols, Standards and Governing Bodies

In 1865 — the age of telegraphy — the establishment of the International Telecommunications Union (ITU) signaled the recognition that worldwide communication would not grow and improve unless countries cooperated in defining standards and protocols that would allow national telegram and telephones to interoprate. The ITU is still very much alive today with governing bodies spanning landline telephone communication, wireless telephony, satellite communications and many other areas.

Question

What is ICANN? IANA?

It was not always a given, but now that we are well into the age of the Internet, it is clear that the governing body that has the most to do with how networking and the Internet as we know them operate is the Internet engineering task force or IETF. The IETF is the governing body that has custody of the overall suite of protocols that define the way that the Internet works. Most notable in this realm is the suite of Internet protocols that go by the name of TCP/IP. While not necessarily part of the Internet, many other organizations and institutions operate networks that use the same TCP/IP suite to implement their networks. The core protocols of this suite include transmission control protocol and Internet protocol both of which are featured in the suite's name and which govern routing on what we now call an Internet-based network. The suite includes under its umbrella many other protocols that support operations on an Internet-based network these include protocols to look up names you may — may recognize DNS in that role — and things like time synchronization handled by the protocol NTP or network time protocol.

TCP/IP protocols and their changes and enhancements are embodied in documents call RFCs, or Request For Comment. In technical discussions, it is not unusual to hear networking types use phrases like "RFC 1918 addressing" to reference a specific technical subject.

Question

How many different protocols are under the TCP/IP umbrella?

Many factors are considered in the development of these complicated protocols including what degree of reliability is required what performance is required. Some protocols may include elements that are unreliable by design and are simply called unreliable. It is not an insult but a very practical decision of trade-offs to allow a certain degree of unreliability in the name of better performance and the the converse can be true.

To the amusement of some, many of the key protocols begin with the term "Simple" when they are hardly that.

Before looking any more closely at protocols it will help to back up a bit and look at other defining aspect of "inter-networking" in the form of the layered model mentioned earlier and at packet switching. Let us look at the basics of packet switching first.

4.4.2   Packet Switching, Routing, and Metrics

Imagine, if you will, Charles Dickens sitting at Gad's Hill, Kent writing *A Tale of Two Cities* one paragraph at a time. He's doing it this way because the only way he has to send his prose to the publisher is on the back of a stack of postcards one card at a time. At a very basic level this represents how a packet switching network operates, by reducing a larger message or text to small postcard sized packets which are then transmitted individually.

Now, you might ask, how will Dickens' postcards get to the publisher in London if one of the bridges over the Thames is out of service, or if a certain post office along the route is closed for the day or has burned down. The logical answer might be that the postcards should be sent over a different bridge and through a different post office. This can be seen to represent another important component of packet switched networking, that of "routing".

Now put yourself in the shoes of the publisher. Hundreds of new postcards have arrived from Dickens but they have come across different bridges and through different post offices and it's not really obvious what the order of these postcards are is supposed to be of whether any postcards might be missing. Some of the postcards are obviously damaged or smeared. In fact the first part card reads "A Tale of [illegible] Cities" and the second "it was the winter of our discontent". What is the publisher to make of this text?

These issues are all something that packet switched networking must address. Routing, address resolution, sequencing, and checking for completeness.
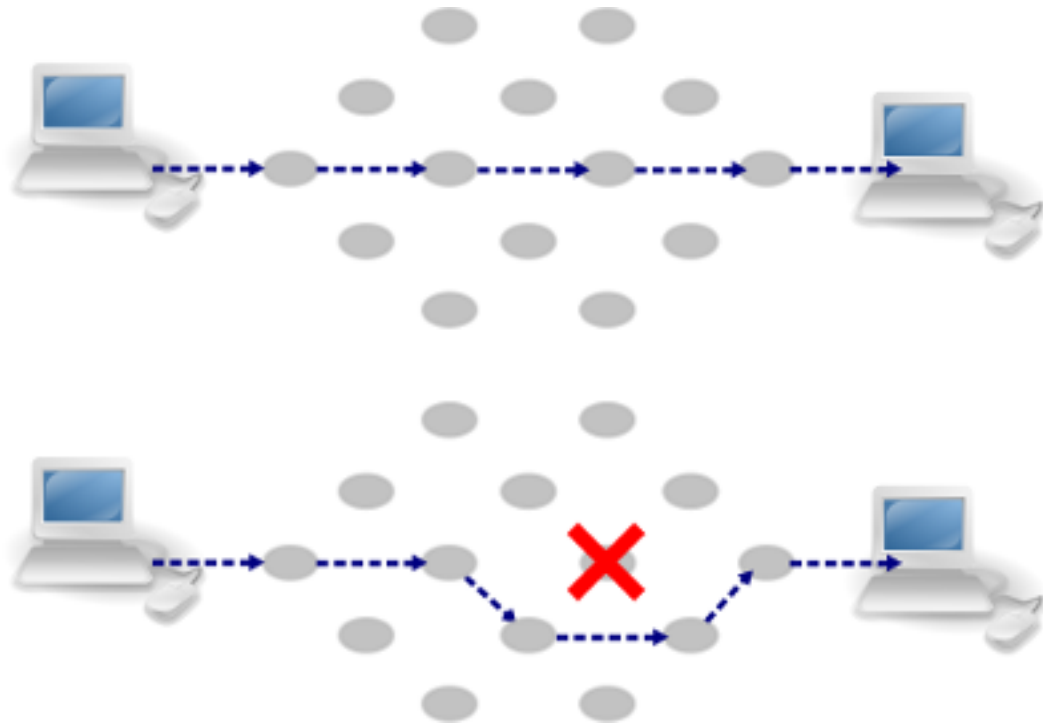
Figure 4.4: Routing example - Image: Pluke, CC0 1.0 Public Domain

This diagram is a very simple example of network routing that simply shows how traffic may choose another path through the network in the event a device along the path has failed. The protocols that we will talk about a little later are essential to making sure that, regardless of the route, the data transmission is successful and copes with any variables along the way.

There are quite a number of other elements and terms that are essential to routing but I'll mention just a few of them here including hops, latency, congestion, queuing and metrics. Let's take a quick tour through all five terms.

"Hops" are mostly an indicator of how many network routing devices are given packet must pass through to reach its destination.

"Latency" is a measure of call it lateness — delayed transmission (and its reception) over the network — especially over long distances.

"Congestion" is more or less a measure of just how much traffic is on the road so to speak. Think of daily traffic on Interstate 5 as it ebbs and flows and slows in response to congestion. Inter-networks aren't much different.

"Queuing" is essentially a positive component of most networks that allow data packets (or cars!) To get where they are going by imposing certain rules of queuing and by implementing controls to enforce those rules. Think of the light controlled on ramps on Interstate 5. Of course for queuing to be effective you must have someplace for packets or cars to queue up like the entrance ramp off of 45th street. In the network world devices that participate in routing traffic include queuing capacity by design so that packets have a better chance of reaching their destination expeditiously even if there is some delay involved.

"Metrics" are something that are implemented in routing protocols that consider a lot of the other factors we have just discussed — queuing, congestion, latency, hops — and make calculated decisions about the best way to send packets on their way. If you had to drive from the UW campus to Edmonds right now, you would have a choice of routes. Take I-5 Or Highway 99? And you would likely consider pretty much the same list of factors as these "routing metrics".

4.4.3   Resolution of Names and Addresses

Protocols and jargon beget more protocols and jargon. If we suggested that you go to 128.95.230.32 to work on RStudio, it might be hard to remember your intended destination. Instead, we are able to say go to "phage" or "phage.deohs.washington.edu". This is made possible by a resolution protocol system called the domain name system or DNS with which many of you are likely familiar.

Further, resolution is required not simply to that IP address, but to an actual hardware device in the computer and to something called a MAC (media access control) address. This MAC address is something that is equally unfriendly to humans consisting as it does of a string of hexadecimal numbers. Resolving an IP address to a specific computer and its MAC address required another protocol called ARP or address resolution protocol. But we may be drifting into the realm of that protocol called TMI.

4.4.4   The Layered TCP/IP Model

To bring more conceptual order to this complicated set of protocols, a layered model is used as a reference. Take a look at this diagram. While there are other layered models in the communications world, the TCP/IP model aimed for some simplicity in comparison to others, with only 4 layers (other popular models such as the OSI Model have more). Probably the most frequently referred to protocol in the suite is IP or the Internet protocol which deals with what would be called an IP address and which is also fundamental to the operation of routing across an IP network.

It is not so important for you to remember a lot of details about the great number of protocols employed, but we review this information with an eye towards helping you learn how to troubleshoot problems you may have using a networked computer. More on that later.

4.4.4.1   Data Flow of the Layered TCP/IP Model
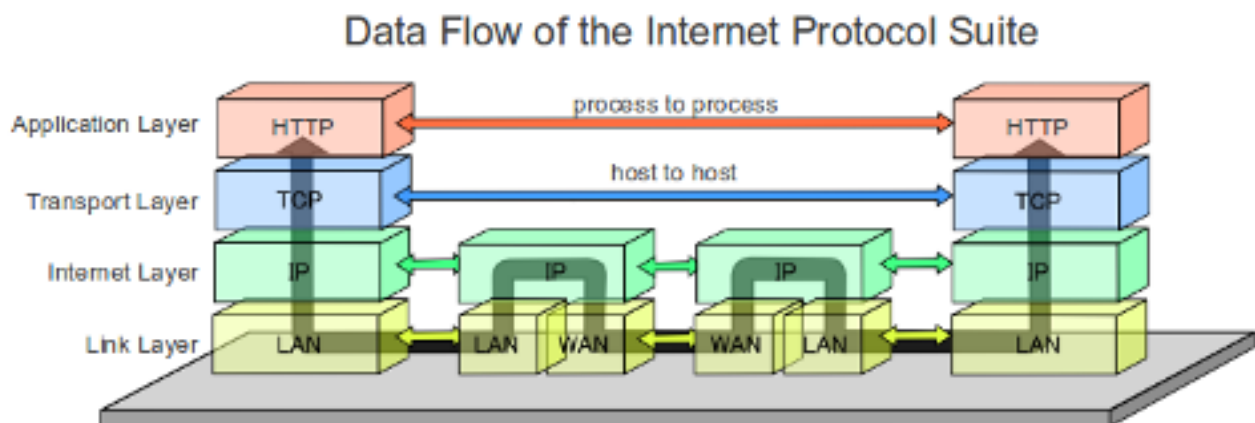
Figure 4.5: TCP/IP Model - Image: renepick/Rob-nowman, CC BY-SA 3.0

Let's look the flow of data through the layer TCP/IP model. You can see that data originates in the highest level of the model, the application layer and is handed off to the transport layer then next to the Internet layer and then down to the link, or physical layer. After passing across a network, the data is then sent up through the layered model in reverse order. This is how all Internet communications operate.

4.4.4.2 Layers, Protocols, and Encapsulation

The importance of the layered model operation makes a bit more sense when you consider another fundamental operation of IP networks, that of encapsulation. While some of this attractive diagram has not been translated from the Dutch it should be pretty straightforward to see where we have an application layer where information is transformed into packets of binary data for encapsulation by the transport layer then again encapsulated into the Internet layer format and so on until the data is finally encapsulated for transmission over the link layer of data network.

Using our Charles Dickens example, it would be as if Dickens ran out of postcards but discovered a cache of envelopes. Think of encapsulation as a layer of envelopes around chapters of his long tale.
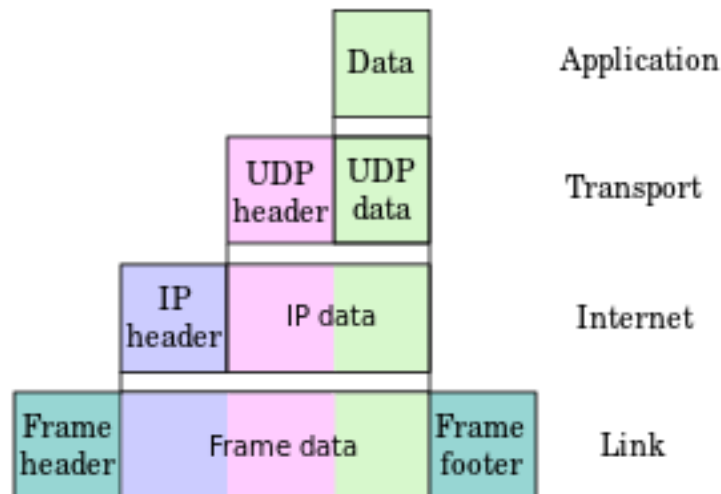
Figure 4.6: TCP/IP UDP Encapsulation - Image: Cburnett/Kbrose, CC BY-SA 3.0

This diagram also offers a very clear depiction of the protocols associated with each layer.

4.4.5 Types of Networks

Let's quickly review a few other key elements of contemporary computer networks.

The term LAN for "local area network" means pretty much what it says it is the local network that you are connected to and routing does not necessarily occur except to make connections to other LANs. Connections to other computers on the local area network are handled more by protocols like ARP.

The term WAN for "wide area network" refers to a collection of networks tied together across a wide area and where elements like routing become more important.

The distinction is not often absolute.

Question

Is the UW network a WAN?[a]

---
[a] http://en.wikipedia.org/wiki/Wide_area_network

4.4.6 Network Topologies

Network equipment can be organized in different so-called topologies. Common examples include mesh, hub and spoke (star), and bus. The Internet can be considered the biggest example going of a mesh network. Closer to home, a star topology is more likely to be found on a local area network (LAN).

- Mesh, Star, and Tree are arguably more common in this era
- Complex networks can combine elements of multiple topologies.
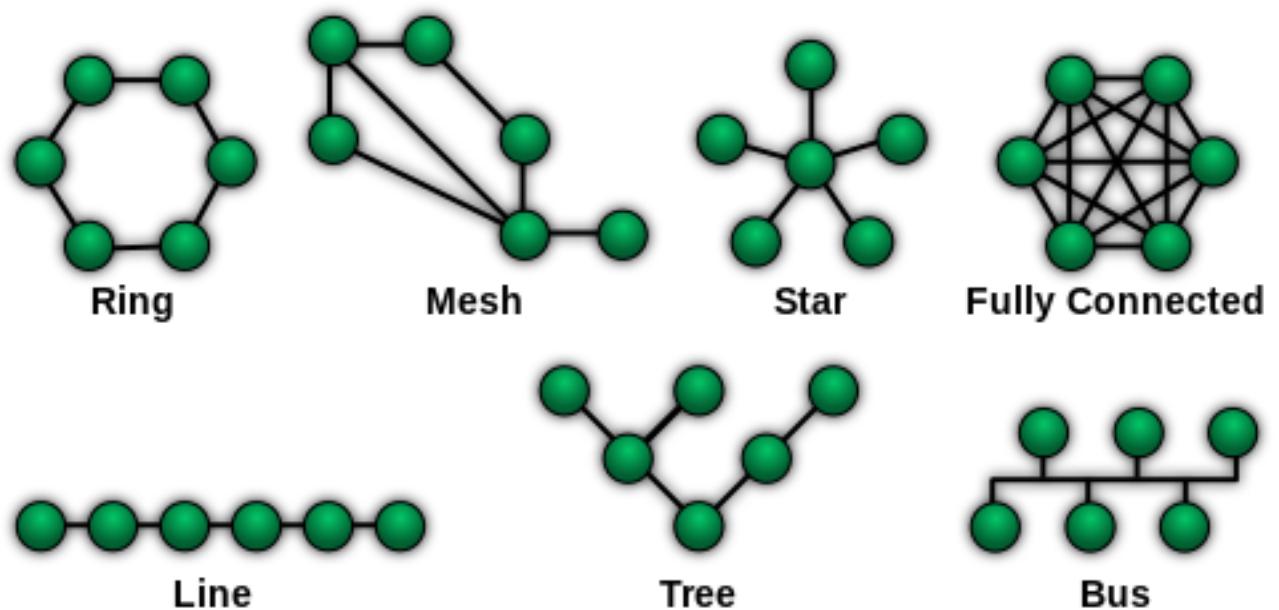
Figure 4.7: Topologies - Image: Malyszkz, Public Domain

4.4.6.1 Link Layer Technologies (Ethernet Rules!)

Figure 4.8: 10Base2 - Image: Dflock/Zcrayfish, Public Domain

At this point in the history of computer networking it could seem that Ethernet is the only networking protocol that ever existed to supply layer 2 networking and switching. While that is not the case, for practical purposes it might as well be considered so. Competing protocols have come but mostly gone.
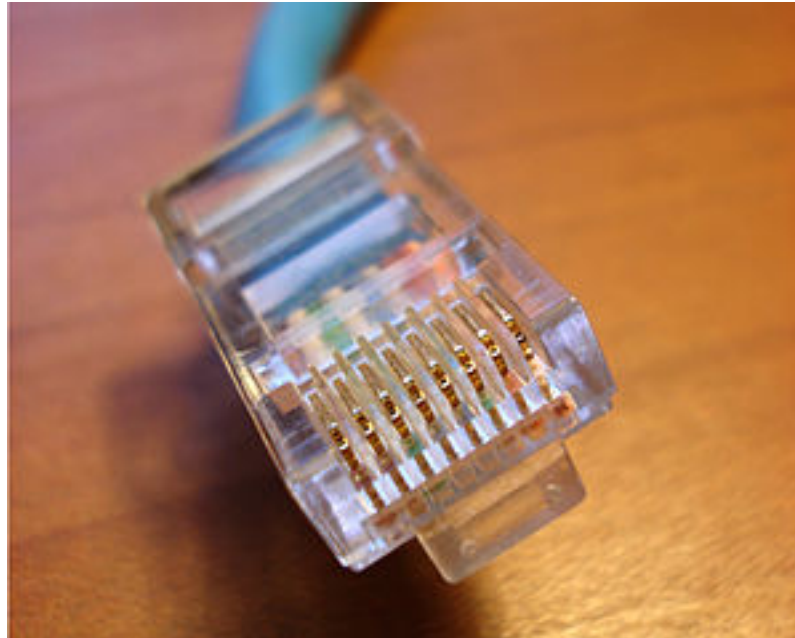
Figure 4.9: RJ-45 plug - Image: Justin, CC BY 2.0

Ethernet has continuously evolved, from a bus-based system in the 1980s to Ethernet utilizing twisted-pair cabling and the familiar RJ45 connector, and can connect over a variety of copper, fiber, and wireless/radio media. All brought to us by another important standards body, the Institute of Electrical and Electronics Engineers.

Figure 4.10: copper and fiber - Image: © BrokenSphere / Wikimedia Commons, CC BY 3.0

While not restricted to local area network technology, network speeds are referred to in bits per second — bps — with variations based on the speed of the network. Probably most familiar is the term megabits (million bits) per second — Mbps. Speeds have progressed from 10 Mbps, past 100 Mbps, and now range into "gigabit" or billion bits per second — Gbps.

Take note that the letter B in"bps" is expressed in lower case where a bit is simply one binary digit. Contrast this with "MB" for megabyte where a byte is an unit consisting of 8 bits. This distinction becomes important when trying to calculate the time needed to transfer data over a network where the network speed is expressed and measured in some number of bits per second but the data to be transferred is stated in megabytes

4.4.7 Primary Network Devices

So, what are the devices that make these complex topologies work?

Two types of devices, perhaps familiar, provide the functionality needed for a packet switched network. The first would be a "switch". The second is known as a router. The former is typically implemented to support a LAN while the latter is intended to provide routing services for a WAN.

Another device with which you are probably familiar is a wireless access point.
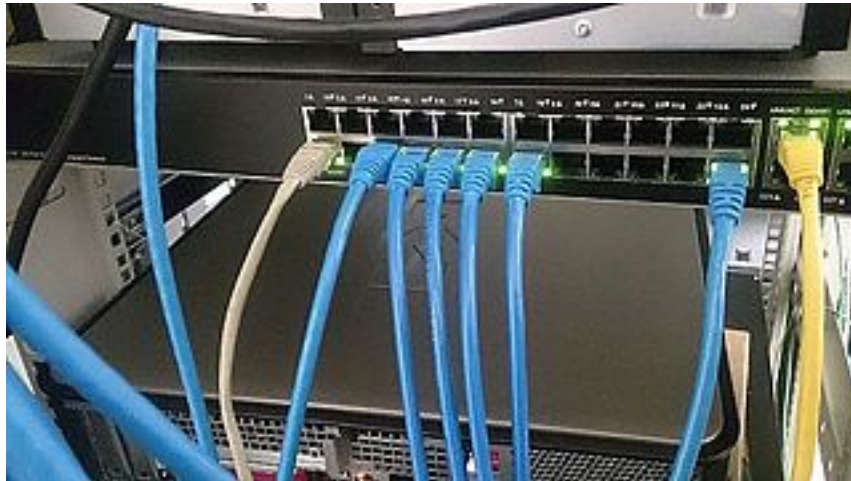
Figure 4.11: Switch - Image: ProjectManhattan, CC BY-SA 3.0

By design these newer devices may combine functions and be hard to distinguish solely by appearance or size.

4.5 Using the Network Effectively

First keep in mind that a network is only as good as its weakest or slowest link. Network service can be a performance bottleneck in some situations.

A wired connection to your computer is almost always more reliable than a wireless network connection.

In the case of bottlenecks remember that read and write speed to and from a local hard drive will typically always be faster than the speed of read and write to share or folder on the network.

What are some troubleshooting techniques?

For one start with the component closest to you and work your ways outward. Troubleshooting what seems like an Internet problem may be nonproductive if it is simply the case that someone has unplugged your network cable from the wall.

Learn how to use a simple tool called "ping" to assess whether remote network hosts are reachable by IP address.

Get a grip on name resolution and DNS. Knowing whether a problem is simply a failure of domain name resolution can save you a lot of time and aggravation.

Learn how to determine your IP address using tools on your computer or web based tools like what is my IP.com.

Whether you are attempting to solve a problem by yourself or wind up seeking help from another party, knowing some of these techniques will help get you back in action much more quickly.

For more information, please take a look at *Networking* in the computing basics wiki.

Chapter 5

Information Security

5.1 Introduction

You may ask "Why should I care about information security?"

To help answer that question let's try a short, meditative exercise. Please pause for a moment to consider the following questions:

- What information do I care about? Personal information like bank account numbers and credit histories? Documents you have created as part of class assignments? Data you are responsible for as part of a laboratory experiment or research study?

- For the kinds of data that you are responsible for and care about, give some thought as to what might result if that data was:

  - Lost when a laptop was stolen
  - Obtained by a criminal
  - Disclosed to the world over the Internet
  - Found to have a host of errors
  - Subtly altered in a way that would not be noticed for 5 years

Okay, if you have given that a few minutes thought, let's continue...

5.1.1 Outline

Some terms may be meaningful to you yet, but in this session we'll cover material in 5 main areas:

- Part 1: Some philosophical foundations of information security

- Part 2: Risk assessment

- Part 3: Security controls

- Part 4: Encryption

- Part 5: Security best practices

5.2   Part 1: Philosophies and Foundations

5.2.1   Why Security?

So why is security in general a good thing? Some folks manage to form the impression that people who are enthusiastic about "security" are people with some control issues. While I'm sure you can find some security enthusiasts who might be described as "control freaks", a more fair picture is that mature security professionals are not interested in control anywhere near as much as they are committed to the pursuit of risk management. So we see security and Information security, not as a control-freak *end*, in and of itself, but rather as a *means* of identifying and dealing with risk.

You might still ask "How does security matter to me? Why should I care?" With luck the introductory meditation will make it less difficult to identify examples of how security, or the lack of it, can have a profound impact on your work and career. We'll see a few examples in the next sections.

5.2.2   Working for the CIA

Information Security, often referred to as "InfoSec" uses a number of different frameworks to pursue its desired result. The most commonly-used InfoSec principles revolve around the triad "CIA", standing for confidentiality, integrity, and availability. There are more complex expressions of this concept but for our purposes will stick with CIA.

Confidentiality should be really self-explanatory: only parties who should have access to information get it.

Integrity means that the accuracy and consistency of data throughout its life-cycle is assured. It means that data can't be modified in an unplanned or unauthorized way and that your data is correct at least with respect to your expectation of what the data should be.

Availability is a lofty term for making sure that you can get your data — that your data management system is built and designed in such a way that it will continue to make your data available to authorized parties and mitigate against threats to availability whether they be computer failure, network failure, a malicious exploit, or something else.

5.2.2.1   What does CIA address? (Examples)

It should be easy to think of some reasons why these three concepts are important. Let's take the example of a research study that you are working on. It has been funded to the tune of $5 million over five years, and there is the possibility of renewal. By chance, the study collects and stores some private health information (PHI).

Possible results of a failure to assure data confidentiality should be obvious these days. Harm to study subjects, lawsuits against the sponsoring institution, disciplinary actions, loss of funding. . . The list goes on.

Integrity, and a failure to maintain it, are probably easiest to think of with an example like a large study that spent millions of dollars, collected lots of data, but then discovered at some point that the data was more or less corrupt, in-correct, inconsistent, and invalid. Possible harms? Incorrect conclusions making their way into the scientific literature would be one. Withdrawal of funding would be another on top of damage to professional reputations.

Availability would seem more straightforward, but also requires considerable thought and planning. What good is confidentiality and integrity if all of your study data was lost in a fire or stolen from your laptop in an airport lounge? How good are your backups?

CIA is a mainstay of good Information Security practice, but there are many additional elements that form a more complete InfoSec framework.

5.3   Part 2: Risk Assessment and Management

5.3.1   Frederick II the Great

"He who defends everything defends nothing."

— Frederick the Great

There is a quote, attributed to Frederick the Great, that posits "He who defends everything defends nothing". The saying is popular among some security professionals. Frederick was talking about war and the need to manage limited resources and logistics to achieve a strategic victory, so the application to InfoSec can get pretty strained. Still, it does have bearing in information security as the resources that any one organization can bring to bear to secure the organization are limited. A contrary assertion — "He who does not defend everything defends nothing" — also matters a in the InfoSec world, where the failure to secure even innocent–looking assets could lead to a much greater compromise of the organization's information. This is something we'll talk about when we cover the concept of "privilege escalation".

5.3.2   Allocating Resources for Information Security

So what is one to do? How do you decide where to spend your finite, limited resources and time to protect your data and information?

One cornerstone of good information security practices is that of risk management. While it might sound rather dull, a risk management approach is key to making decisions that optimize the application of your resources to secure your information and the organization.

5.3.3 Risk Assessment Methods

The field of risk assessment covers a lot of territory and can have many different emphases. This can range from a billionaire commodities trader trying to assess the risk of a small drop in the price of precious metals all the way to you trying to judge how much insurance coverage to obtain for your car or home. Let's say you have only 2 valuable possessions, a brand-new laptop that cost $2000 and a used 70-inch LED TV that cost $1200. You might approach the task of determining the the right insurance coverage by calculating something called "Single Loss Expectancy". SLE is calculated in the formula:

$$Single\ Loss\ Expectancy\ (SLE) = Asset\ Value\ (AV) \times Exposure\ Factor\ (EF)$$

To arrive at SLE you would calculate the asset values of the Laptop and the TV and the so-called "exposure factor" (how much of this stuff some burglars might haul off).

Okay, when you look at these two assets you realize that while the laptop would be a cinch to pinch, it would be nearly impossible for thieves to get your TV out of your 3rd-floor walk-up. So, while your total Asset Value is $3200 but your Exposure Factor would be .625 reflecting that the laptop would likely be stolen while the TV remained. So his Single Loss Expectancy is $2000. This is a very simple example and might seem a bit circular, but it is the basis for more complicated estimates.

- Asset Value = Laptop ($2000) + TV ($1200)

- Exposure? Laptop, YES. TV, NO.

- Exposure Factor = 0.625

- Single Loss Expectancy = $2000

"Exposure" is not limited to a specific object, but can be an expression of likelihoods and probabilities. Example: You might use the annual probability of 8.0+ earthquakes in Seattle to estimate the chance of your TV being destroyed by a natural disaster.

5.4 Part 3: Controls

5.4.1 What is a Control?

To paraphrase conflicting definitions, "Security controls are safeguards or countermeasures to avoid, counteract, minimize and/or recover from risks and threats related to security".

5.4.2 Types of Controls

A successful information security and data management plan will address specific identified risks by means of specific measures or "controls". These can be looked at in a few different ways. One main way is the type of control. Several important types along with some examples include:

- administrative (for example written policies that are enforced)

- logical (required computer accounts are the most common example)

- physical (door locks and access cards for example)

Another primary way to look at security controls is whether they attempt to prevent the occurrence of a risk event, reduce the impact of a risk event if it does occur, or restore your systems and information when a risk event occurs with full impact. So, we have controls that are:

- preventative

- corrective or mitigating

- restorative

Table 5.1: Example Controls

5.4.3   An excercise in Risk Mangement Calculation and Control selection

Let's look at some domestic burglary risk management and decision-making about controls. We'll use the previous scenario where you want to secure your laptop and the information on it from theft.

Having a German Sheppard patrol the front yard would definitely be a preventative control, as might the "Protected by ADT" event if you didn't have an alarm. But keeping a mean guard dog has its own risks.

A real alarm system for your house would also be preventative, but, also, since it could summon the police before the perp escapes with your laptop, it could also be a mitigating/restorative control. But an alarm would also be expensive and it raises the question of continuing cost. Would it be better to pay once for better locks that will last 20 years?

A restorative control might be harder to identify. While you might back up the data on your laptop as a restorative control, the chances of ever seeing that laptop again are tiny and the main restorative control would likely be homeowners

insurance.

5.4.4 How does this apply to Information Security?

One truism that most information security professionals would agree with is that "There is no silver bullet.", meaning that there is no single technology or technique or philosophy that will adequately protect information in a complex world. They recognize that somebody might give your faithful guard dog some tranquilizers or enter your house through a window in the back

Most every decent information security plan will require a multifaceted approach that combines preventative/defensive methods along with procedures that mitigate the adverse effects of security exploits and breaches. From time to time, you will hear someone say something like "We are okay. We have a firewall!" Fixation on any single measure or technology is a recipe for future information security disaster. Successful security practice is implemented in multiple layers, or a "layered defense".

With other topics remaining to be discussed we may be forced to give risk management short shrift. It is an area that could consume hours days months years, and to which some people have devoted their entire careers. That being said, at the beginning of any project and more usefully before it is even launched (or funded!), It is essential to devote some time to an assessment of the projects risks so as to develop adequate controls.

This is not cut and dried. While it might be easy to identify a risk such as losing data on a laptop that gets stolen, other risks might be less tangible yet important. For example, what if the lost data on that laptop leads to the cancellation of a project or an unsuccessful grant renewal?

Paradoxically an asset can be a risk. For example think about the very bright data analyst you hired for the project and who is the only person in the project who truly understands some of your statistical methods. The data analyst is a great asset to the project until they take a new, highly-paid position in Paris. In impersonal terms, they were always a risk, really.

One risk is that you could spend too much time on risk assessment!

5.4.5 Typical Risks and Threats in Information Security

There is a danger in saying "typical" in this context, as risk profiles can cover a lot of different areas across different environments. A company located in the Swiss Alps may need to account for the risk of avalanches while their subsidiary in the Philippines will need to pay attention to the dangers of the Monsoon season.

In this day and age, however, most of the threats that we see highlighted on the nightly news are ones that are specifically malicious and leverage access to computers over the Internet. Many of these threats may target a range of networked computers including general-purpose computers along with more specialized machines such as those operating Web servers. Threats will also vary based on whether a computer system is the primary target of criminals, say a bank system or a large retailer, or whether the threat is intended to have a more general target and result, like pulling PCs into a so-called "Bot-net"" or getting Web servers to serve malicious material and scripts, An attack against a high value target like a retailer could combine all of the above to execute an attack and compromise the retailer's critical systems.

5.4.6   Common Internet and Web-borne Threats and Terms

Let's quickly review a number of common threats and terms:

- Social Engineering

- Trojans

- Phishing

- Spear Phishing

- Brute Force Attack

- Escalation of Privilege

- Advance Persistent Threat

- Zero-Day Exploits

- SQL Injection

- DoS/DDoS

5.4.6.1   Threat: Social Engineering

This refers to using psychological mean to manipulate people into performing certain actions or divulging confidential information. It is the crux of many gumshoe detective novels and is not confined to information security in the computing and systems sense. Think of a detective telling a receptionist "Your boss said I need to fix his telephone right away!" when really all the detective wants is to look in boss man's file cabinet for evidence. It is a lynch pin of man other threats and techniques.

5.4.6.2 Threat: Trojans

A Trojan (from Trojan Horse) is a exploit most commonly delivered over a network (bit not always. USB keys have been used). The key to a Trojan is that it does something different than what you might be made to believe and it requires an action on the part of the recipient to open the hatch, more or less, and let all of the Greeks out.

5.4.6.3 Threat: Phishing

This should be familiar. The technique of sending formatted emails made to look like they are from an institution like a bank. All in order to trick people into divulging personal information like passwords and credit card number. You've seen them: emails saying that your checking account at Wells Fargo is being shut down unless you CLICK HERE. But you don't have a Wells Fargo account!

Spear Phishing is a variation that is much more targeted. Where the perpetrators know that a group of people have certain assets, positions or information and craft a much more customized message with more realism. An example might be an email to all of the stockbrokers at Bank X from the bank president inviting to sign up for the annual meeting in Hawaii (which does exist).

Threats can also be mixed. like Spear phishing emails that also bear a trojan.

5.4.6.4 Threat: Brute Force Attack

Imagine you forgot the 4-digit code to the bathrooms in Roosevelt. Everyone else has gone home. So you start entering a sequnce of 4 digit codes as fast as you can to see if one matches. That is brute force. Thankfully, brute force attacks can often be spotted and mitigated, but not so when they are used off-line to try to discover actual passwords from encrypted password hashes

5.4.6.5 Threat: Escalation of Privilege

This is the technique, often exploratory in nature, by which a perpetrator gains increased privileges on a computer system over time. We've made an analogy for this in a separate short feature.

5.4.6.6 Threat: Advanced Persistent Threat

An APT is a combination of sophisticated techniques being used in an attack to the point that the threat elements persist on the system after an initial attack and may be difficult or even impossible to remove.

5.4.6.7 Threat: Zero-Day Exploits

A so-called "zero-day" is an exploit against software and/or systems that becomes know to the word at large before the author, publisher, or manufacturer has even one day to try to fix it and issue a patch. So, a zero-day

5.4.6.8 Threat: SQL Injection

"Sequel" injection is an attack method aimed specifically at relational database systems and most often through forms on Web sites that use a SQL-based data story. They work when an poorly-designed/coded Web page or form allow an attacker to append SQL commands to a Web site URL such that the injected query gets processed. This is just one of a number of Web-specific attacks, but perhaps the most common.

5.4.6.9 Threat: DoS/DDoS

These stand for Denial of Service and Distributed Denial of Service. The latter has become increasingly common, whereby a large number of computers are employed to overload another computer by sending vast numbers of requests to that computer over an Internet.

5.4.7 Typical Controls in information Security

- Access Control

- Physical

- Logical

- Network

- Software

- Anti-Virus

- Browser configuration and add-ins

- Alarm Systems and Monitoring

- Intrusion Detection (physical and logical)

- Server hardware health

- Environmental conditions (temp, smoke, fire)

- Redundancy

- Backups

5.4.8 Access controls

- Identification

- Authentication

- Authorization

A key element to securing data is the concept of access control. This includes the requirement that a party successfully identify themselves and that is followed by a requirement to authenticate this identification. Authentication can take a number of distant different forms including a stored secret that is known only to you or device like a smart card. You are probably familiar with recent discussions about a move to so-called "multi-factor" authentication systems in payment card systems and ATMs where a combination of password plus physical device are required.

Authorization relies on successful authentication but provides a different service – that of granting rights to individual parties based on membership in defined groups or roles. A very simple example from our department: to edit some particular pages on the department's website, you need to be a member of the group "web team"..

5.4.9   Further assurance for your stored data

Remember that controls can operate in different realms, sometimes at the same time.

For example backups may not be a completely successful preventative tool in deterring the risk of some of your data – let's say that a the day's worth of your data was lost due to an equipment failure for someone's failure to save their work. However, those same backup controls could be an excellent preventative control when looking at the risk of losing your business or a grant renewal or contract.

Typically backups are tailored to capture data that is still in active use in a business. At some point it may become prohibitively expensive to continue to back up data that is not in active use. This is the point at which a business would consider storing legacy data in some type of lower-cost archive. Archives can be critical control measures depending on the circumstances including things like regulatory and contract compliance.

We should mention in passing that some systems, like ours, have the ability to take so-called "snapshots" of stored data at desired intervals. This provides a mechanism to do a previous version of a stored file and more or less "rollback" the document or data. Some advanced data systems have this rollback built in and use concepts such as journaling to be able to look at state of the data at any moment in time.

Probably much more important and of much more utility to average folks managing average data is the concept of versioning and version control. version control is useful on a number of different levels. Most commonly associated with the control of programming source code, version control is essential especially when multiple parties are collaborating on the same programming tasks. The version control offers great benefits even for an individual user, and even managing the versioning of things like documents or evolving data sets. Formal version control systems use a defined repository to store versioned data. We have implemented one here in the department using the software called "git" and that repository is integrated with our Redmine system.

- Backups

- Snapshots

- Archives

- Version control

5.5   Part 4: Encryption Technologies

Wikipedia says "In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it." That sounds about right.

cryptography itself goes back in some form to ancient times but "crypto" and encryption based on sophisticated mathematical algorithms really started to emerge in the 20th century. Late in that century the emergence of the Internet helped propel the use of encryption and standards for its use.

Modern cryptography is based on so-called "keys" of differing types.

5.5.1   A Key

Here's an example of a pretty much randomly generated key:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info:DES-EDE3-CBC,568B9A3A3399B91F
```

```
tE+eI4cCX5iAHL34MEnTV+AmA+iWmRx+RRUWF5aJ4EX+A/zWNnIexwJEL0aWecA/
PQzqkEj5b22MGD/Y4sVmuQPMaJFpEGwHpn1voT+uUhAzC5ne3njCQtaPZF6XIRh6
36tTELNI6uJfd6o/VNC+ya4HfvI3iQlMZn6IP0wrEMSDk/1a3dc90POXFXERgkS0
N020tQ69zRkJnw1IMGAkIXqOOjMlkBARFMW94HWkfiZ0vA+v6mg34MdUQqln0ibc
y9xLaI2XakyIZN65z+ZzU3VPqAnDSN/vOUPuzr6PJSLg1UkKL6u8oxaZZZgUEIyB
G1TYbA9sLrScONJF8eBW/fI+7yK/a0wWnzsCJ59zNeno6Dg+6jFasJmjMhYpWOjm
uX99QJWhIaE4evI75h0vGSc/psTw2X4ppYyj6TnbORc2+HiIoNpKyeq3ovcRpPmN
x97fGBYI6GzaCF1u7q2EN85IVaydCCNLzA4p3NJPriw0M21sGu+MXrqPKKtd46O6
t27pYf/9Gm1QtOkwpOyKn2pVVEKGZoFfxKi+gYxyrJFUBtVpuhs+jW35IA7mCUeo
IS/0vtPU0vlQs4xsz7yOv4h4iozPCmzKSXvQ0J4Az6z/rsrwjcoS3f6bwWVLzMaM
kYqQpT4h0PdCtHBygBQFpPnoc6ocsZmGIkzibOJ3z0EVncMFTKHTtMKMYqFuJRgo
Xq+WsYqlOrBfusGLt1ReGJ0fVPQmWAbCDvoEn4BSfv8nQZqwFFxH1ev1YD1E/nYg
e84JT4dnzAzZ/k1I9TlZOvzAn3+2qP33CWXgMoful1bqr9oSvSX9Pw==
-----END RSA PRIVATE KEY-----
```

All keys appear to be this meaningless!

5.5.2 Main Uses of Encryption

These uses generally fall into two main classes:

- Encryption of data in motion - in transport "streams"

- Encryption that is more or less file-based

5.5.3 Encryption software

5.5.3.1 Stream/Session based:

- Protocols and tools like SSH, SFTP (SSL/TLS)

- Secure Web using SSL and represented by Lock Symbol

- Digital Certificates mostly for Web sites but also applications

- Negotiation

5.5.3.2 File based:

- 7-Zip

- TrueCrypt

- PDF

- PGP

5.5.4   Other Encryption Elements and Factors

• Codes/cyphers: the mathematics continue to evolve to defeat code breakers

• Keys: Length is a "key" factor in strength

• Standards include NIST standards like

• DES

• 3DES

• AES

5.5.5   A Public Key Encryption Diagram

A breakthrough in cryptography led to the technology referred to a Public Key Encryption. A diagram shows how users can make available their public key available to a friend or the whole world while retaining the complementary private key that is required to crypt and decrypt.
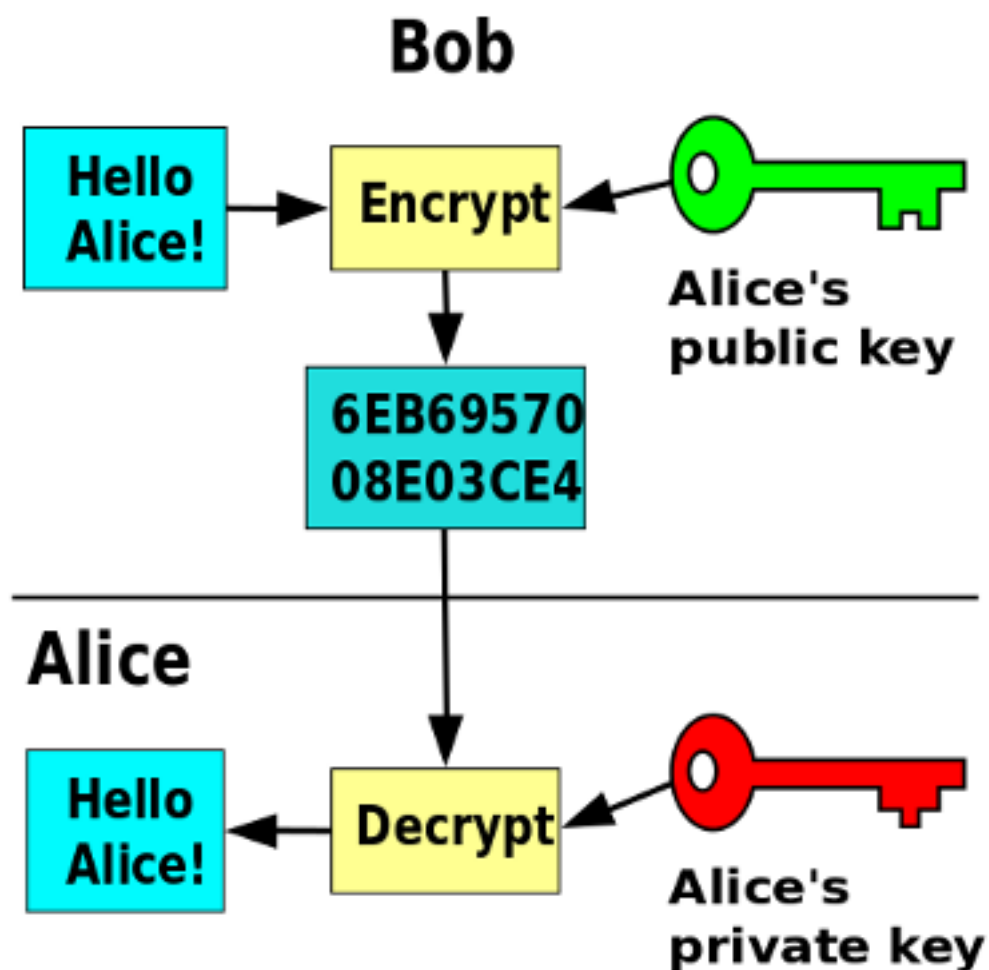
PGP is a popular software that employs this public key method.

5.6 Part 5: Security Best Practices

5.6.1 Some Basic Security Tips

- Security on personal computers and devices is easily subverted.

- Internet-connected devices are under attack day and night.

- Anti-virus software is not updated fast enough to keep up.

5.6.2 Best practices

- Least Privilege

- Default Deny-All Policies and explicit permissions

- Characterize and understand the baseline security environment and leverage it. Don't reinvent and risk possibly making things more complicated.

- Safeguard private keys!

- Subscribe to relevant security announcement lists.

- Use widely adopted industry standards like OWASP.

Web application security: OWASP Top 10

5.6.3 Other links and some recent real-life examples

- Krebs on Security Blog

- Wikipedia "History of Cryptography

- Wikipedia on Single Loss Expectancy

- DDoS on GitHub Purchase Uber Logins Online!

- UW Professor encounters Ransomwear (will discuss in class)

Chapter 6

Instrument Interfaces

Data is waiting to become information. Through instrument interfaces, and data acquisition software you can transform raw data into useful information.

6.1 Input/Output (I/O)

Computers and devices are equipped with a variety of inputs, and outputs. Inputs and outputs have both physical and virtual components. On the physical side, is the various ports, connectors, and cables utilized. On the virtual side, are the various protocols that operate over the physical interface. That said, physical interfaces aren't limited to cabled connections. Wireless both radio frequency and optical are growing in popularity. Regardless of the physical medium most are digital, however some, such as audio, are analog.

6.2 Peripherals

Peripherals are not limited to monitors and keyboards. They also include sensors, and complex instruments. Each of these devices connects via some interface which could be wired or wireless. Be aware though, that a peripheral may be available in more than one interface type or may be equipped with multiple interface options. For example, monitors often equipped with both analog and digital inputs.

6.3 Instrument Interface Technologies

An input/output card expands a computer's I/O options. These can range from a basic USB card, to a GPIB controller. Something to keep in mind, laptop computers tend to be more limited in their I/O expansion options. If you need more than USB and Ethernet capability, a desktop computer is recommended.

Figure 6.1: Chassis Plans 8011 Digital I/O Card: Lippincott, CC BY 3.0

A bus technology allows multiple devices to share a common medium. With bus technologies, you can share a single port on your computer with multiple devices via a bus specific hub or in some cases via daisy chaining devices.

Hewlett-Packard Interface Bus evolved into the General Purpose Interface Bus. It's a standardized interface bus used by a wide array of scientific instruments. One nice feature of GPIB, is the ease of converting it into another interface technology such as Ethernet or USB.

Figure 6.2: IEEE-488 Connectors: 1-1111, Public Domain

Ethernet is the most common network interface technology. So common, that most computers built today include Ethernet on-board. Many modern instruments have native Ethernet support, enabling them to be accessed directly from a desk, instead of via a computer located next to the instrument.

Serial and Parallel are both "legacy" interfaces. Their popularity has been in a decline over the past decade, largely being replaced with USB and Ethernet. Due to their decline, they aren't typically seen on modern computers.

USB is a fairly universal bus technology. It's used to connect everything from keyboards to GPIB interface controllers. It's found on every computer made in the past 10 years, making it a safe choice when compatibility is needed.

6.4   Remote Data Acquisition

Sensors exist for most everything, from temperature and pressure to particulate detection.

Hackable devices have become popular in the past few years. They are designed to make development of custom devices and sensor packages more accessible to a wider user base. In the past, where a commercially built sensor package would be used, a hackable system may be a suitable replacement.
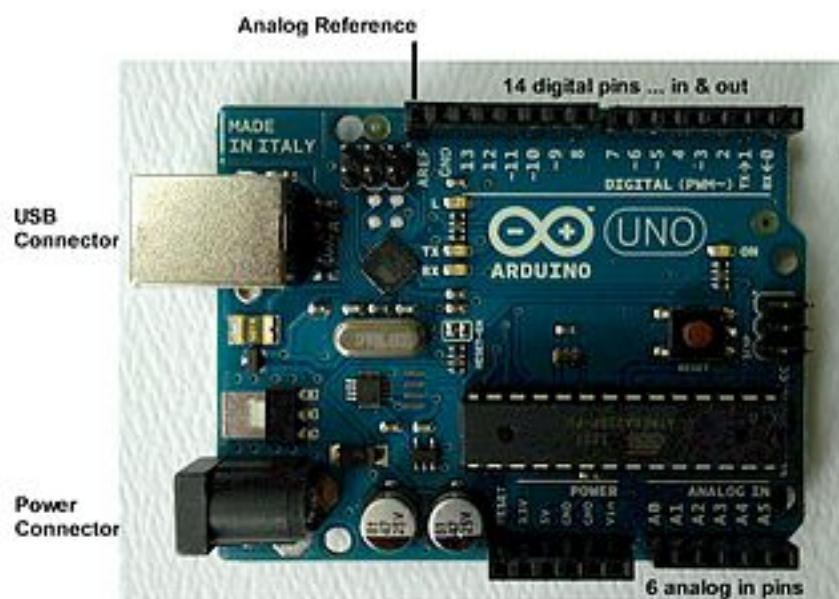
Figure 6.3: Arduino Uno: 1sfoerster, CC BY SA 3.0

GPS enables you to quickly and accurately determine location, altitude, and time. By integrating GPS into your field data acquisition, you may be able to partially automate location tracking of sensors and samples.

Mobile devices such as smartphones and tablets can be used for field data acquisition. They can be used to scan barcodes on sample containers and record the location (when equipped with GPS), or used to log notes during an interview.

6.5 Instrument Data Acquisition Software

Instruments require acquisition software. It translates the raw data from the instrument into a useful file format, or provides some additional processing capability. Keep in mind, some software and formats are limited to specific brands of instruments. So, be sure the software you wish to use will work with your equipment and other tools.

LabVIEW is a popular package due to it's unique development language, called G. What sets G a part from others, is the graphical development model. Instead of writing pages of code, you graphically build your acquisition system by dragging and dropping objects on the screen.

ChemStation from Agilent is one of many chromatography packages. Like many instrument acquisition packages, it's a vendor specific software package limited to use with Agilent equipment.

Torrent Suite is a modern sequence analysis package. It's available with an open source license, making it free software. Open source alternatives exist for a variety of proprietary software.

6.6 Legacy Systems

Figure 6.4: Intertec Superbrain: Brighterorange, CC BY SA 3.0

Earlier we introduced the idea of "legacy interfaces" like serial and parallel, which are showing up on fewer and fewer computers these days. Which introduces a problem with both hardware and software used to interface with an instrument.

Let's take for example a mid 90s vintage HPLC, which is likely to use GPIB to talk with the computer. While the basic GPIB interface is supported on a modern system, the instrument may have other requirements. Such as requiring manufacturer specific software that depends on their own interface card. And, the interface card uses the ISA bus, a bus that hasn't been used in computers in almost 10 years.

So, why is this a problem? As the computer ages, it's reliability goes down. In addition, it's compatibility with peripherals and software becomes worse over time. Thus, while the instrument may still do it's job, you may have difficulty transferring data from the computer to other systems, printing, etc. These issues may first appear as minor annoyances, but can quickly evolve into a work stoppage.

You can help to mitigate this situation, by reviewing the requirements for the instrument. For instruments that have a very long life, it's best to avoid ones that depend on proprietary hardware interfaces. So, look for units that use USB or Ethernet. Review the software support policy, most manufacturers require you to purchase updates over time. But, make sure they have plans to support the instrument long term. In addition, you may want to see if any third-party software vendors support your instrument, as they have a greater financial interest in supporting older equipment.

6.7   Instrument Overload

Up until now, we've covered how instruments interface with a computer, software that interacts with the instrument, and the pitfalls of legacy systems. Now, we must turn our attention to the overall resource needs of instruments.

In a lab environment, it's typical to have 1, maybe 2 instruments connected to a single computer. However, in the interest of saving money, and space some attempt to connect far more than that to a single computer. If the instruments aren't being used simultaneously, you can probably get away with it. But, if you do wish to interact with many at once, you're likely to run into issues.

One group actually attempted to connect a couple dozen instruments in a mobile environment to a single laptop computer. From a pure software perspective, their tool of choice, LabVIEW, is designed to interact with multiple devices at once. But, as the number instruments goes up, the amount of CPU, Memory, and Disk activity also goes up. That said, their main issue wasn't with any of those resources, it was with interface technology. Each instrument connected via USB, which is a bus technology. And, like most bus technologies, the more devices on the bus, the slower it gets.

So, when developing a plan involving instruments that require a computer interface, you must consider more than just the physical connection, and the software support. You must also consider if the interface technology can really support that many active instruments at a time. In this example, the lab group ended up purchasing a second computer to divide up the work load.

6.8   Summary

To review. Peripherals are typically external input and output devices, such as keyboards, and monitors. There are a number of interface technologies on the market, the latest instruments can be purchased with Ethernet built-in, while most older devices will rely on GPIB. As for data acquisition, in the field you can use mobile devices like tablets, and GPS to aid your work. And, in the lab, use software like LabVIEW to process data from an instrument.

When purchasing lab instruments or working with older models keep a few things in mind. Older instruments may not work properly with modern computers. In some cases, the manufacturer may have a paid option to make it work, such as a software or interface upgrade. However, that option may be costly, so it's best to plan ahead for that scenario, rather than be stuck trying to keep an old computer alive.

Finally, make sure your instrument needs, don't exceed your computer and interface limitations. If you exceed those limits, you could face issues with reliable recording of data to possible hardware damage. Ultimately, planning ahead is likely to save time, and money.

Chapter 7

Resource Management

Information can be processed most efficiently when the appropriate resources are allocated, and utilized in an effective manner. In order to aid you, we'll go over some methods of determining your resource needs, and techniques to best utilize the resources available to you.

To some this process is a dark art. At it's core, is the process of capacity planning. Capacity planning involves determining the amount CPU, RAM, and Disk your information processing needs. To achieve efficient resource utilization, you may need to optimize your work flow, and potentially leverage technologies like parallel processing. In order to ensure you are effectively using your computing capacity, you'll need to monitor your resource utilization at points throughout your work.

7.1   System Resources

The CPU or Central Processing Unit is the heart of your computer. As the name implies virtually all data processing is handled within the CPU. In simplest terms, a CPU's performance capability is measured in two ways. It's clock speed, that is the frequency the CPU operates at, which is measured in megahertz or gigahertz. And, the number of cores it has. Each core can perform a single operation (or calculation) at a time. So, the more cores you have, the more calculations that can be performed simultaneously.

RAM or Random Access Memory, is very fast, but short term memory. It's job is to hold the data that the CPU is actively working with. If your needs call for large amounts of RAM, you're in luck. It's relatively easy to upgrade, and fairly cheap.

Disks are used to store data for the long term. But, not all disks are created equally. There are two main types of disk, Solid State Drives, and Hard Disk Drives. An SSD is many times faster than a hard disk, but that comes with a much heftier price tag for large amounts of storage. Hard Disks on the other hand, have been around for years. They are cheap even for storing several terabytes of data.

With disks, there are 3 basic ways to utilize them. The most common is a stand alone drive, either internal or external to your computer. The next most common is network storage, where the disks live somewhere else on the network. Network storage is good for large capacity, but rarely good for high performance. The third, is a disk array, which is several disks working together usually in a redundant fashion so data is retained in the event of a disk failing.

7.2   Optimization

In order to optimize your data, and work-flow, you need to identify what resources you are using, identify bottlenecks, and eliminate them.

Every operating system has tools to track resource usage. On Windows, the Performance Monitor (example shown below) is the most helpful. It gives a breakdown of RAM, CPU, Disk, and Network utilization by application. On a Mac, the *Activity Monitor* is the most user friendly method of tracking resource usage. And, in the newest versions of OS X, it has a color coding scheme to help identify if you are hitting the limits. On Linux, you've got many choices, but `htop` is a solid choice for CPU and RAM monitoring. For disk activity, you'll want to use `iostat`.
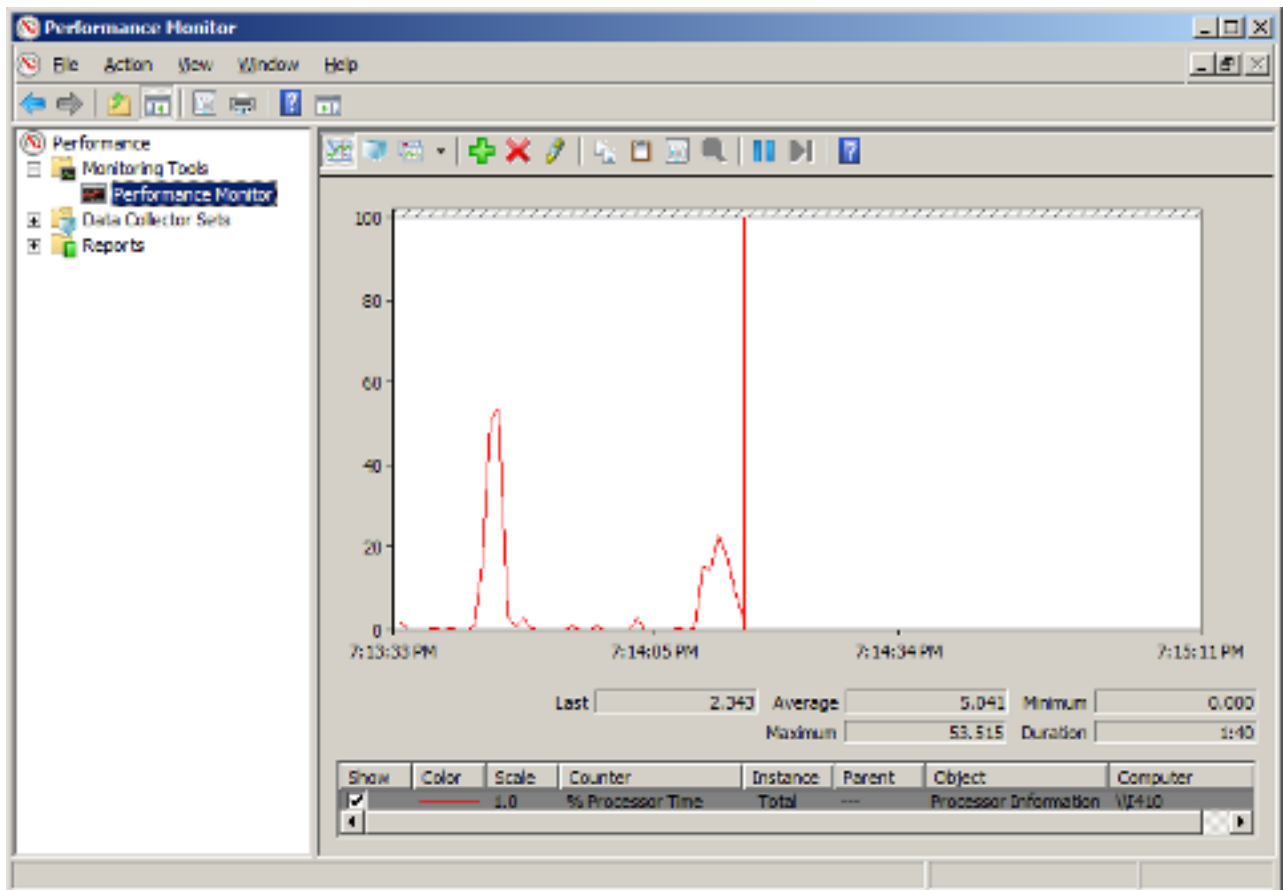
Figure 7.1: Windows 7 Performance Monitor: Microsoft

Once you have determined your usage, you can try to identify bottlenecks. A bottleneck could be caused by your available resources, or your software. If you aren't maxing out the CPU, Memory, and Disk, then the bottleneck is likely within the software itself. However, if you are maxing out a particular resource, then increasing the available resources should help. For example, if your system has a single hard disk drive, and it's being maxed out, replacing it with a solid state drive should speed things up.

7.3 Memory Utilization

Random Access Memory or RAM is a finite resource. The amount you can install into a system is limited by the CPU, motherboard, and sometimes the operating system you are running.

In order for a system to perform efficiently, you must use your available RAM in an effective manner. Which means you need to ensure that each application or processis allocated enough memory. While at the same time you must

avoid exceeding the available amount. When you exceed that limit, your computer will begin to swap or thrash. The result being poor performance, and a possible loss of data.

Allocating memory is made a bit more difficult due to the behaviour of some applications. Software such as R, MATLAB, and Excel by default do all of their data processing in RAM. Which means, you need enough memory to fit the raw data set, and any changes being made to it.

If you plan on working with large data sets, you have a few options. The first option is to purchase enough memory to do all your work in RAM. However, this option can be costly, and doesn't really scale well. Your second option is to break up the data into smaller pieces, and do your calculations in sections. The third option, is to use a plugin or addon for your software that lets you work from disk. This option however isn't available for all software.

7.4   Case Study: Climate Change Project

To help provide context, we will look at a real research project conducted by the University of Washington.

Let's start the scenario with a support request made by a research assistant to the departmental IT staff. Here is the actual request.

```
I'm running a SAS program that involves building a large dataset
from the meteorology files. Each meteorology file is 864 KB, and
I'm merging together about 5,000 of these separate meteorology
files into one giant dataset. So far I've been running this program
since Wednesday, and it's still running. I'm thinking that perhaps
my computer does not have enough processing memory to run this
task efficiently. Also, I'm running this off of my C:/drive, so the
slow processing time is not due to sending information over the
network. I think that if I continue to let my program run over the
weekend, it should definitely be done by Monday. However, I'm
concerned because I will need to run this program again for another
set of about 5,000 files later.
```

This provides sufficient detail to get an idea that this is a resource management problem. Either more memory needs to be installed, or the (685 line) SAS program needs to by modified to use less memory.

After looking further into this issue, we discovered that there were 5335 pairs of files, with each pair consuming 2284 kilobytes.

Let's calculate the memory needed to load the entire dataset into RAM:

```
5335 * 2284 KB = 12185140 KB
```

12185140 KB is about 12.2 GB of RAM. Plus, as the files were being combined in memory, a second (combined) copy of the data was being collected in a separate table, also stored in memory, before being written out to disk. So, the total memory requirement may have been over 24 GB of RAM, (assuming SAS does not store the data any more efficiently than as ASCII text). The computer running the analysis did not have nearly that much RAM installed. So, the main reason this was running slowly was that the hard disk was used as "swap space" when the RAM had been consumed. This "swapping out" of memory to and from disk is extremely slow.

But was it really necessary to load the entire dataset into RAM (twice)?

We ended up writing a short script that used negligible resources and did the job in less than half an hour on a even modest computer. So, "no", we don't have to load all of the data into RAM. We can just "stream" the data, joining and streaming one pair of files to at a time, continually appending the stream of data to an ever-growing output file.

How did we do it? Let's start by looking at the input data formats. Here is a listing showing the first 9 lines of each type of input file:

```
$ head -9 data/data_45.59375_-122.15625
27.950 8.170 0.570 3.100
1.950 6.130 -1.000 3.030
3.925 5.650 0.490 3.020
2.400 5.680 -0.880 2.980
8.200 6.330 0.020 2.990
1.600 5.560 -0.360 3.040
14.650 5.420 0.310 3.410
37.775 5.080 -0.340 3.280
6.575 6.550 -0.500 3.440

$ head -9 force/force_45.59375_-122.15625
1915    1       1       237.83  41.655  79.851  0.1660
1915    1       2       244.82  40.886  77.705  0.1666
1915    1       3       248.43  32.984  84.425  0.1174
1915    1       4       246.30  39.536  79.572  0.1494
1915    1       5       245.51  38.740  81.598  0.1405
1915    1       6       249.16  37.549  81.220  0.1393
1915    1       7       254.88  33.961  84.068  0.1190
1915    1       8       262.68  35.973  82.808  0.1245
1915    1       9       258.02  43.198  79.247  0.1583
```

Here is an example of the desired output, showing the header and first 9 data records of the output CSV file:

```
$ head met.csv
lat,lng,precip,tmax,tmin,windsp,year,month,day,b1,b2,rh,b3
45.59375,-122.15625,27.950,8.170,0.570,3.100,1915,1,1,237.83,41.655,79.851,0.1660
45.59375,-122.15625,1.950,6.130,-1.000,3.030,1915,1,2,244.82,40.886,77.705,0.1666
45.59375,-122.15625,3.925,5.650,0.490,3.020,1915,1,3,248.43,32.984,84.425,0.1174
45.59375,-122.15625,2.400,5.680,-0.880,2.980,1915,1,4,246.30,39.536,79.572,0.1494
45.59375,-122.15625,8.200,6.330,0.020,2.990,1915,1,5,245.51,38.740,81.598,0.1405
45.59375,-122.15625,1.600,5.560,-0.360,3.040,1915,1,6,249.16,37.549,81.220,0.1393
45.59375,-122.15625,14.650,5.420,0.310,3.410,1915,1,7,254.88,33.961,84.068,0.1190
45.59375,-122.15625,37.775,5.080,-0.340,3.280,1915,1,8,262.68,35.973,82.808,0.1245
45.59375,-122.15625,6.575,6.550,-0.500,3.440,1915,1,9,258.02,43.198,79.247,0.1583
```

Here is a simple version of a Bash script which can do the job.[1]

```
#!/bin/bash

echo 'lat,lng,precip,tmax,tmin,windsp,year,month,day,b1,b2,rh,b3'

for LATLONG in `find data -type f | sed -n "s/^data\/data_//p"`
do
    DATA=data/data_$LATLONG
    FORCE=force/force_$LATLONG

    if [ -e $FORCE ]; then \
        paste -d, $DATA $FORCE | sed -e "s/^/$LATLONG,/" -e 's/[ \t_,]\+/,/g'
    fi
done
```

Requiring only a small fraction of the number lines of code, as compared to the SAS version, we can combine all of the files in just a matter of minutes.

---

[1] This Bash script will also work when run in Bash's POSIX mode.

> **Aside: Minimal code, maximal ugliness**
>
> At the risk of code obfuscation, we can reduce the lines of code to three:
>
> ```
> echo 'lat,lng,precip,tmax,tmin,windsp,year,month,day,b1,b2,rh,b3'
> for D in data/data_*; do L=${D/#data\/data_/}; F=force/force_$L
> [ -e $F ] && paste -d, $D $F | sed -e "s/^/$L,/" -e 's/[ \t_,]\+/,/g'; done
> ```
>
> We don't recommend this approach, however, as it is now very hard to read!

We can run our script using these Bash commands:

```
$ cd /path/to/met
$ bash metmerge.sh > met.csv
```

How does it work? For every (space-delimited) file in our data/ folder, there is a corresponding (tab-delimited) file in our force/ folder sharing the same location coordinates. We join matching "data" and "force" files, sequentially, line-by-line, by matching paired files on location (latitude and longitude), as found in the file name of each file. (Here is an example of a "data" file name: "data_45.59375_-122.46875".) As we process files, we continually write the combined output "stream" in CSV format to a text file.

A Python version running on a decent server runs about twice as fast as this Bash script running on an old laptop. So, the total run time can be reduced to about 15 minutes, using a single Python process (CPU core), and less than 25 MB of RAM (peak).

```python
#!/usr/bin/python

import os
import sys

hdr = "lat lng precip tmax tmin windsp year month day b1 b2 rh b3".split()
print ",".join(hdr)

def process_latlong(latlong):
    latlong_out = latlong.replace("_",",")
    data_file = open ("data/" + "data_" + latlong, "r")
    force_file = open ("force/" + "force_" + latlong, "r")

    for data_line in data_file.readlines():
        data_out = data_line.strip().replace(" ",",")
        force_line = force_file.readline()
        force_out = force_line.strip().replace(" ","").replace("\t",",")
        print latlong_out + "," + data_out + "," + force_out

    data_file.close()
    force_file.close()

dir_list = os.listdir("data/")
for fname in dir_list: process_latlong(fname.strip("data_"))
```

7.5 CPU Utilization

With modern CPUs having multiple cores, parallel processing is the only effective way to utilize all of the CPU power available. In order to utilize it, your data and work-flow may need adjustment. With parallel processing, your data is divided into pieces, and calculations are done on several pieces simultaneously.

Thankfully, there are some well developed tools and techniques to help with this. One of the more common is MapReduce which was popularized by Apache's Hadoop. MapReduce is a framework for processing large volumes of data in parallel. Some lesser seen tools include GNU Parallel which is a tool used to run and manage command-line tools in a parallel fashion.

As an example, climate data can be processed in a parallel fashion. The data can be divided up by area, and then computation performed on a per area basis. Thus, instead of doing calculations for one ZIP code at a time, you could process data for 4, 8, or more areas at once.

7.6 Summary

In summary, there are three main components to resource management:

- Capacity planning: Identification and allocation of necessary resources.

- Utilization monitoring: Verifying you are using the resources you've allocated.

- Bottleneck resolution: Identification, and correction of performance bottlenecks.

Chapter 8

Data Files

8.1 Learning Objectives

This module will help you:

- Understand the difference between various file types.

- Learn how to name files to facilitate opening in suitable software.

- Understand common character encoding standards and how to work with them.

- Learn how to characterize a document's structure to identify processing strategies.

- Become familiar with common data file formats and their advantages and disadvantages.

- Understand the importance of good table layout and how to achieve it.

8.2 Data File Types: Binary vs. Plain Text

There are essentially two main categories of digital file types, *binary* and *plain text*.

If a file is binary, the file just contains "zeros and ones". While this is technically true of *any* digital file stored within a binary computer system, the contents of a binary file will not necessarily conform to any *standard* character encoding system. The binary format may be highly efficient for storage or processing, but is essentially *opaque*, in that by simply looking at a binary file's contents, you can't really know what the format is or how to read it.[1] Examples of binary files are database files, most multimedia files, and compressed files (such as zip files).

Plain text files, on the other hand, are composed of *characters*. Typically they are ASCII or Unicode characters represented by one or more bytes, where a byte is (generally) 8 bits. A bit can be considered either *zero* (off) or *one* (on). Plain text file formats are usually open and standard. Examples are web pages (HTML), XML, and CSV (comma separated value) data files.

8.3 File Name Extensions

Filenames generally have an extension, which is the part at the end ("suffix") of the filename, consisting of the last dot (.) and the characters that follow it.[2]

---

[1] Actually, byte sequences called magic numbers or file signatures may be used to identify file formats, but their use is not completely standardized or universal.

[2] *Filename extension*, Wikipedia, CC BY-SA 3.0

Examples of binary filename extensions for images are `.png` and `.jpeg`. To launch "executable" programs on Windows systems you will often launch an `.exe` file. The `.dmg` ("disk image") filename extension is used on OS X. Common extensions for binary data files are `.xls` and `.sas7bdat`.

Plain text file formats for data files include `.csv`, `.tsv`, `.txt`, `.xml`, and `.json`, among others. Program source code is usually stored in plain text files, with extensions such as `.R`, `.py`, `.pl`, `.c`, `.sh`, `.bat`, and `.do`.

The extension is used to determine which "default application" should open it. Within the operating system, the extention is mapped to default applications. Mappings such as these are called file type associations.[3]

8.4   Viewing Binary and Plain Text Files

When viewing the *raw* contents of files, whether they are binary or text files, we will often make use of a *hexidecimal dump*.

Hexdecimal is a base-16 number system with digits 0-F:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Whereas binary has two possibilities, 0 and 1, hexadecimal has 16, including the ten decimal digits plus the letters A-F.

Let's "dump" files in "hex" with hexdump...

```
$ hexdump -C -n 64 filename
```

Where the options we are using is this example are:

- `-C` = display in hex and ASCII

- `-n 64` = show the first 64 characters

- `filename` = name of file to view

In this example, we will view the first 64 bytes of an SVG image file. The file format stores information about the image in text, even though the file is displayed as a graphical image. Our filename is `pie.svg`.[4]

```
$ hexdump -C -n 64 pie.svg
```

```
00000000  3c 3f 78 6d 6c 20 76 65  72 73 69 6f 6e 3d 22 31  |<?xml version="1|
00000010  2e 30 22 20 65 6e 63 6f  64 69 6e 67 3d 22 75 74  |.0" encoding="ut|
00000020  66 2d 38 22 3f 3e 0a 3c  21 44 4f 43 54 59 50 45  |f-8"?>.<!DOCTYPE|
00000030  20 73 76 67 20 50 55 42  4c 49 43 20 22 2d 2f 2f  | svg PUBLIC "-//|
00000040
```

We see a column of numbers on the left which show the character numbers in hexadecimal. Each line shows the hexadecimal number for each of 16 characters, in the center of the output, and on the right is the text equivalent (in "ASCII") of those character numbers.

We can see that this is an "XML" document with a version number, and the character encoding is shown as "UTF-8". The document type ("DOCTYPE") is "svg". All of this is contained in XML tags, similar in structure to HTML

---

[3] file association, Wikipedia, CC BY-SA 3.0

[4] While OS X, Linux and Unix systems come with `hexdump`, Windows systems no longer come with something comparable. Instead, you may wish use PowerShell, which does come with recent versions of Windows, download the hexdump.exe utility and use a command like this: `hexdump.exe pie.svg | select -First 5`... though it will not produce exactly the same output as `hexdump`.

(the language of most web pages). The hexadecimal numbers correlate to the ASCII characters because the first 128 characters of the UTF-8 encoding scheme are the same as the ASCII character set. (We go into more detail on this matter later in this module.)

Even if the file was not a text file, and the ASCII printout looked like random characters, we would still be able to look at the hexadecimal dump to learn about the file.

For example, here is the PNG (binary) version of that same image. We will use the same syntax with hexdump, but look inside the `pie.png` file.

```
$ hexdump -C -n 64 pie.png
```

```
00000000  89 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44 52  |.PNG........IHDR|
00000010  00 00 01 2c 00 00 02 26  10 04 00 00 00 13 97 a3  |...,...&........|
00000020  46 00 00 00 04 67 41 4d  41 00 00 b1 8f 0b fc 61  |F....gAMA......a|
00000030  05 00 00 00 20 63 48 52  4d 00 00 7a 26 00 00 80  |.... cHRM..z&...|
00000040
```

We have the same format of output. On the right, we see that the file is identified[5] as a PNG file, as shown in the first few ASCII characters, but all other ASCII characters appear random (meaningless). Dots are shown for "non-printing" characters. Since the file is binary, and not encoded as characters, the ASCII which has been interpreted by hexdump is not very useful for learning anything more about the image. (As the file is not ASCII encoded, the ASCII interpretation is invalid.) We will just have to open the image in a graphics viewer to see what it is. Although both image files would display the same, you can see that there is a big difference between the contents of plain text and binary file formats.

8.5 Character Encodings

We will now take a closer look at the most popular character encoding standards for text files: ASCII, Extended ASCII, and Unicode.

ASCII was the primary standard text file encoding for many years. ASCII was then extended to include more characters, giving us "Extended ASCII". Lately, in the last couple of decades, Unicode has become dominant because it allows for thousands of characters (because it uses more bytes). Whereas ASCII was originally a seven bit encoding system, Extended ASCII just adds one more bit (to make a complete eight bit "byte"), and Uncode uses up to four bytes.

8.5.1 ASCII

The ASCII ("American Standard Code for Information Interchange"[6]) standard was published a long time ago in 1963 and the current version is from 1986 (ANSI X3.4-1986). It was internationalized in 1983 (ISO 646:1983). ASCII originally used a seven-bit character set, so there were 128 characters (which is two to the seventh power).

8.5.2 ASCII Table

Often we will refer to an ASCII table. There is a command that you can use to generate one. The `ascii` command prints all 128 ASCII characters.

```
$ ascii
```

---

[5] The first few bytes of a file are often used to identify the file type.

[6] *ASCII*, Wikipedia, CC BY-SA 3.0

```
Usage: ascii [-dxohv] [-t] [char-alias...]
   -t = one-line output  -d = Decimal table  -o = octal table  -x = hex table
   -h = This help screen -v = version information
Prints all aliases of an ASCII character. Args may be chars, C \-escapes,
English names, ^-escapes, ASCII mnemonics, or numerics in decimal/octal/hex.

Dec Hex     Dec Hex     Dec Hex   Dec Hex   Dec Hex   Dec Hex     Dec Hex     Dec Hex
  0 00 NUL   16 10 DLE   32 20      48 30 0   64 40 @   80 50 P     96 60 `     112 70 p
  1 01 SOH   17 11 DC1   33 21 !    49 31 1   65 41 A   81 51 Q     97 61 a     113 71 q
  2 02 STX   18 12 DC2   34 22 "    50 32 2   66 42 B   82 52 R     98 62 b     114 72 r
  3 03 ETX   19 13 DC3   35 23 #    51 33 3   67 43 C   83 53 S     99 63 c     115 73 s
  4 04 EOT   20 14 DC4   36 24 $    52 34 4   68 44 D   84 54 T    100 64 d     116 74 t
  5 05 ENQ   21 15 NAK   37 25 %    53 35 5   69 45 E   85 55 U    101 65 e     117 75 u
  6 06 ACK   22 16 SYN   38 26 &    54 36 6   70 46 F   86 56 V    102 66 f     118 76 v
  7 07 BEL   23 17 ETB   39 27 '    55 37 7   71 47 G   87 57 W    103 67 g     119 77 w
  8 08 BS    24 18 CAN   40 28 (    56 38 8   72 48 H   88 58 X    104 68 h     120 78 x
  9 09 HT    25 19 EM    41 29 )    57 39 9   73 49 I   89 59 Y    105 69 i     121 79 y
 10 0A LF    26 1A SUB   42 2A *    58 3A :   74 4A J   90 5A Z    106 6A j     122 7A z
 11 0B VT    27 1B ESC   43 2B +    59 3B ;   75 4B K   91 5B [    107 6B k     123 7B {
 12 0C FF    28 1C FS    44 2C ,    60 3C <   76 4C L   92 5C \    108 6C l     124 7C |
 13 0D CR    29 1D GS    45 2D -    61 3D =   77 4D M   93 5D ]    109 6D m     125 7D }
 14 0E SO    30 1E RS    46 2E .    62 3E >   78 4E N   94 5E ^    110 6E n     126 7E ~
 15 0F SI    31 1F US    47 2F /    63 3F ?   79 4F O   95 5F _    111 6F o     127 7F DEL
```

You will see that there is a header showing the command usage followed by an ASCII table listing. The listing is arranged in 8 sets of columns, with each set showing the decimal (Dec) and hexadecimal (Hex) value for each character. Starting from zero (0), the first 32 characters (and the 128th) are the so-called "non-printing" characters, so those are shown with 2-3 letter codes describing the character. The 33rd character is the "Space" so nothing is shown. All other characters are symbols which appear on the standard US keyboard. The punctuation characters and decimal digits are followed by capital letters, more punctuation, lower-case letters, more punctuation, and finally ending with "DEL" (Delete), the 128th character (numbered 127, or 7F in hexadecimal). To have more characters, we would need more bits in our encoding standard, which we will look into next.

8.5.3 Extended ASCII

Extended ASCII, first published as the ISO-8859-1 ("ISO Latin 1") standard in 1987, adds another bit to ASCII, allowing for 191 characters, adding several rows of new characters to the end of the table.

Figure 8.1: ISO-8859-1 (Latin1) - Image: Roman Czyborra

8.5.3.1 Windows Latin 1 (Windows-1252)

Extended ASCII (ISO-8859-1) was extended even further to Windows-1252, sometimes (incorrectly) called "ANSI"[7]

Here is a listing of the Windows-1252 character set. You will see that there are extra characters at the top of this table that we did not have in the Extended ASCII ("ISO Latin 1") character set, beginning with the the relatively new euro sign (€).

Figure 8.2: Windows-1252 (WinLatin1) - Image: Roman Czyborra

8.5.4 Why Character Encoding Matters

Figure 8.3: Mojibake example: garbled Smart Quotes in email reply

If a file is created using one character encoding, but is viewed using another, the characters are likely to display incorrectly.[8] The resulting garbled text is sometimes called mojibake.

---

[7] *Windows-1252*, Wikipedia, CC BY-SA 3.0

[8] Our first example of mojibake which begins this section is from the reply window of Thunderbird when replying to a message from a Mailman email-list post. The original message was sent from a person using the Mutt email client. The message was PGP-signed. A quick internet search for the unintended characters brings up a useful help page. Apparently, this character sequence representing "curly quotes" or "Smart Quotes" is a common problem.
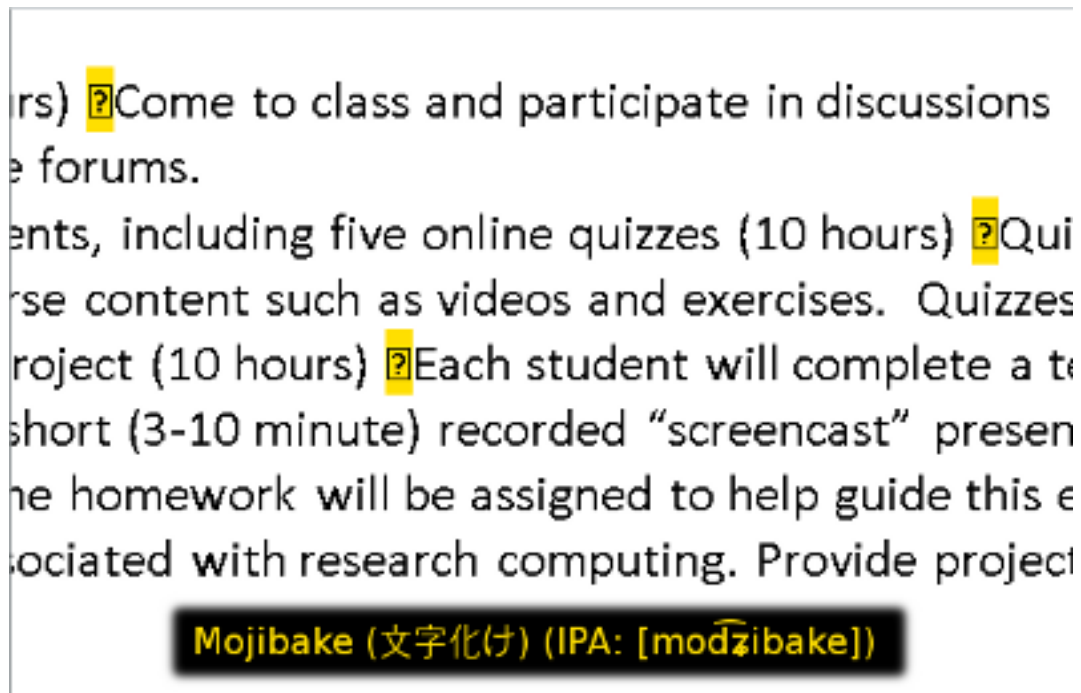
Figure 8.4: Mojibake example: in MS-Word

We can see how differences in character encodings can matter with a few simple examples. Let's first generate a table of characters with Python.

Example 8.1 Printing the Windows-1252 character set with Python

The following Python script will show the printable characters of the Windows-1252 character set when run on a Windows system using a graphical Python interpreter such as *IDLE* or *PyScripter*.
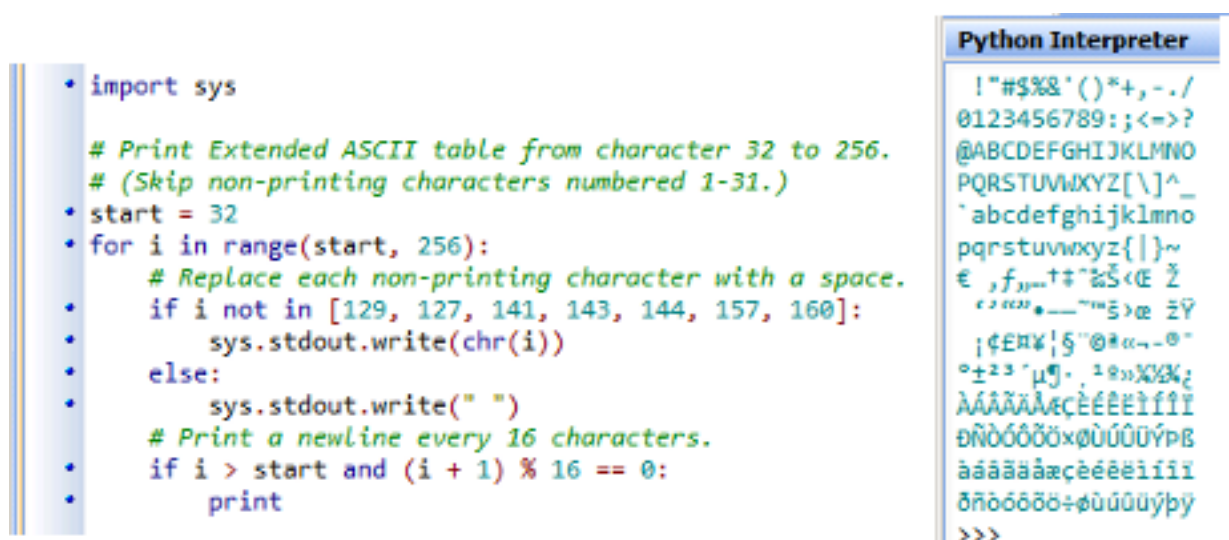
Figure 8.5: Windows-1252 Table Python Script

Here is the full code listing for that Python script.

asciitable.py

```python
# If run on a Windows system in a graphical environment such as
# IDLE's Python Shell, by default, this will print the Windows
# Latin 1 character set, a.k.a. Windows-1252 (WinLatin1).

import sys

# Print Extended ASCII table from character 32 to 256.
# (Skip non-printing characters numbered 1-31.)
start = 32
for i in range(start, 256):
    # Replace each non-printing character with a space.
    if i not in [129, 127, 141, 143, 144, 157, 160]:
        sys.stdout.write(chr(i))
    else:
        sys.stdout.write(" ")
    # Print a newline every 16 characters.
    if i > start and (i + 1) % 16 == 0:
        print
```

Example 8.2 Changing the Character Encoding within your application

We can see the characters properly in a non-Windows environment if we specifically set the character encoding in the application.
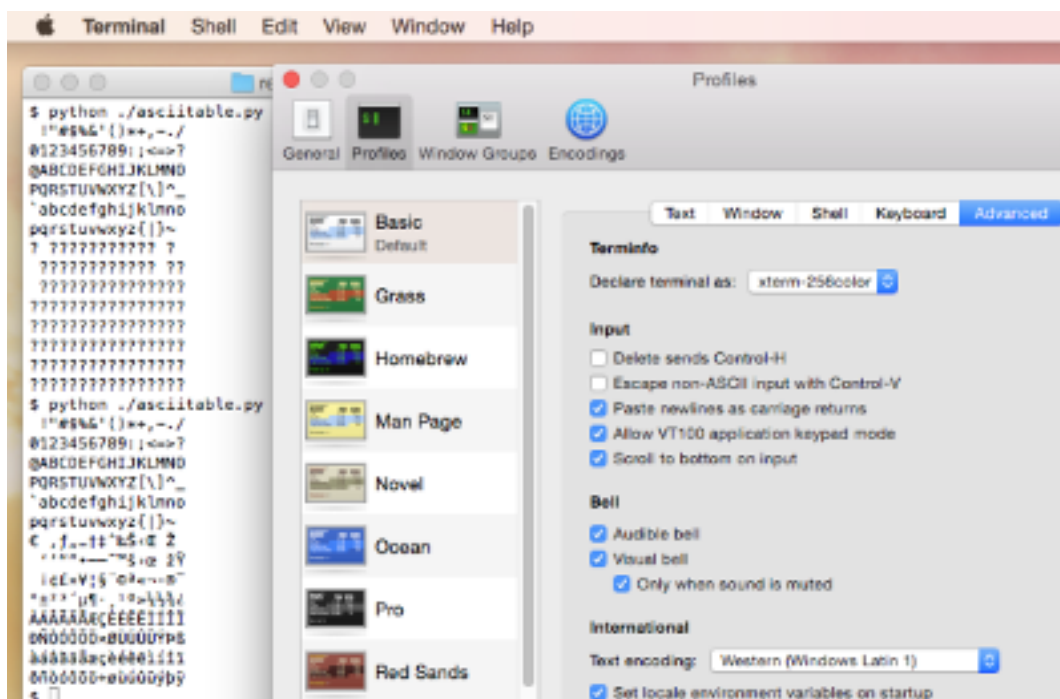
Figure 8.6: Changing Character Encoding in Mac OS X Terminal
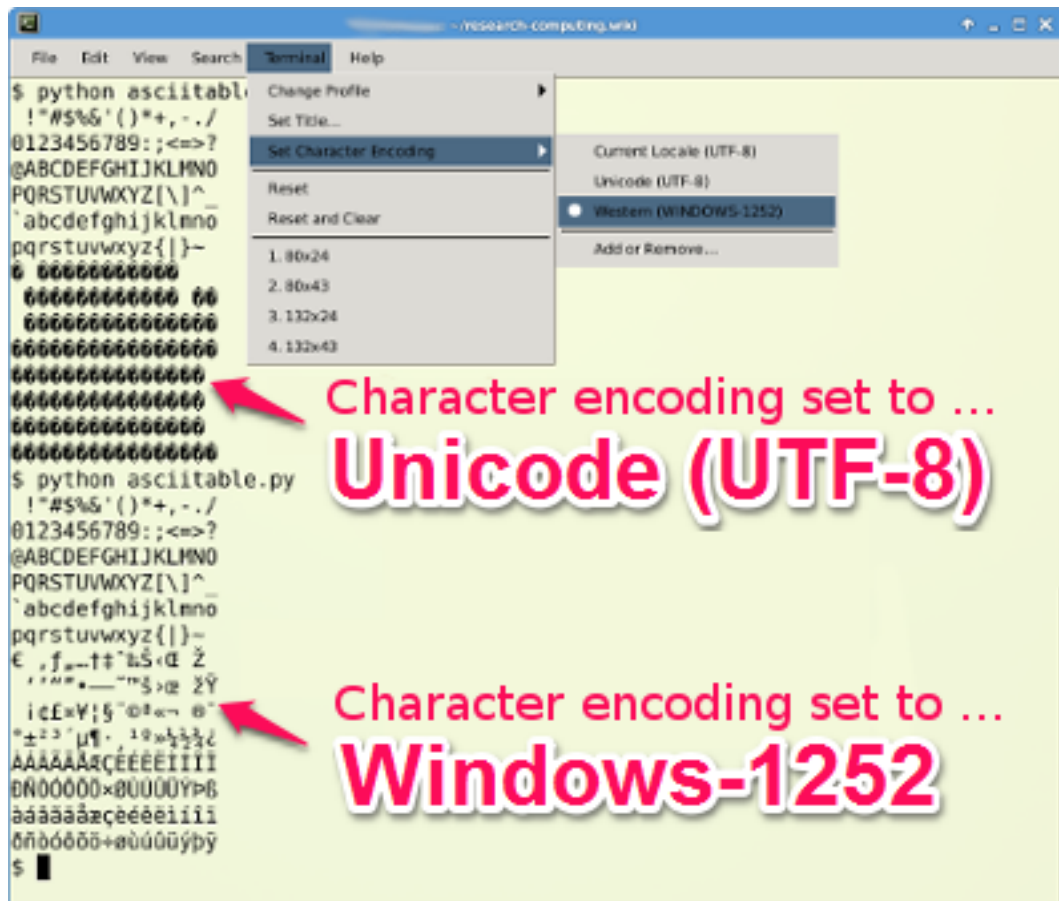
Figure 8.7: Changing Character Encoding in Linux Terminal

However, this is not the default setting. Without knowing the output was encoded as Windows-1252, we might have thought our program had a bug.

So, how can we know the character encoding of "plain text" output? Let's save the output as a file and test the file for it's character encoding.

Example 8.3 Saving the output to a file with redirection

To save program output as a file, we can use *file redirection*. We will run the program on the Windows computer in a *DOS* shell and redirect with the > operator.[9]

```
C:\> python asciitable.py > asciitable.txt
```

Redirection allows us to save to a file, but that file just contains the numeric codes for the characters. There is nothing in the file stating the actual character encoding format. We will have to guess, using the `file` command.

Example 8.4 Testing the file type with `file`

We can check the character encoding and other file properties using the `file` command. This command is available on Unix, Linux, and OS X systems. Here we will run the `file` command from a *Bash* shell.

---

[9] We would use the same `python` command from the *Bash* shell in Unix, Linux, or OS X.

```
$ file asciitable.txt
asciitable.txt: Non-ISO extended-ASCII text, with CRLF, NEL line terminators
```

While this tells us a little about the text format, we still don't know the specific encoding standard used.

As you can see, dealing with various character encodings on different computing systems can be tricky.[10] Is there a universal character encoding standard? Yes, *Unicode*! In the next section, we'll see how we can convert our file to Unicode.[11]

8.5.5 Unicode

Unicode provides an internationalized character encoding **standard**, to "encompass the characters of all the world's living languages".[12]

- Like ASCII, but supports over 110,000 characters

- Unicode standard was published in 1991

- Most commonly used **encodings** are UTF-8 and UTF-16[13]

You can browse the Unicode code charts to get an idea of the many character sets available.

8.5.5.1 Unicode Symbol Example: the Micro Sign

Example 8.5 Encoding the Micro Sign

The character μ, with Unicode[14] **name** "MICRO SIGN" is encoded:

Encodings	Decimal	Hex
Unicode	181	U+00B5
Extended ASCII	181	B5
HTML **numeric** character reference	µ	µ
HTML **named** character entity	µ	

Example 8.6 Typing the Micro Sign

How do you type the μ character into your computer?
Use these character codes:

Name	Decimal	Hex
MICRO SIGN	181	00B5

---

[10] You will find many cases of this issue discussed on help forums such as stackoverflow. There are modules in programming languages such as Ruby and Python which help address these problems. Applications like MS-Word also allow you to set or convert encodings.

[11] For a more thorough treatment of dealing with character encoding problems, see "Bad Data Lurking in Plain Text" by Josh Levy, PhD, which is the fourth chapter of Q. Ethan McCallum's *Bad Data Handbook* (O'Reilly Media, Inc., 2012).

[12] Joe Becker, Unicode 88

[13] *Unicode*, Wikipedia, CC BY-SA 3.0

[14] *Unicode*, Wikipedia, CC BY-SA 3.0

With these operating systems:[15]

- **Windows**: [Alt]**decimal** (using numeric keypad) ... *or* ... **hex**[Alt][x] (does not require numeric keypad)

- **OS X**: for μ, you can simply use [Opt][m] ... *or* ... [Command][Ctrl][Space] ... Search by **name** ... *or* ... use **Unicode Hex Input** (Input Source) and **hex**

- **Linux**: [Shift][Ctrl]**hex**

8.5.5.2 Some Other Useful Symbols

Table 8.1: HTML Entities for Common Math Symbol Characters

Character Name	Char.	Entity	Num. Entity	Hex. Entity
DEGREE SYMBOL	°	°	°	°
MICRO MU SYMBOL	μ	µ	µ	µ
LOWER CASE SIGMA	σ	σ	σ	σ
N-ARY SUMMATION	$\sum$	∑	∑	∑
GREEK SMALL LETTER PI	π	π	π	π
GREEK SMALL LETTER ALPHA	α	α	α	α
GREEK SMALL LETTER BETA	β	β	β	β
GREEK SMALL LETTER GAMMA	γ	γ	γ	γ
INCREMENT	Δ	Δ	∆	∆
GREEK SMALL LETTER EPSILON	ε	ε	ε	ε
INFINITY	∞	∞	∞	∞
PLUS OR MINUS	$\pm$	±	±	±
NOT EQUALS	$\neq$	≠	≠	≠
ALMOST EQUAL	$\asymp$	≈	≈	≈
GREATER THAN OR EQUAL TO	$\geq$	≥	≥	≥
LESS THAN OR EQUAL TO	$\leq$	≤	≤	≤
DIVISION SIGN	$\div$	÷	÷	÷
SUPERSCRIPT TWO	^{2}	²	²	²
SUPERSCRIPT THREE	^{3}	³	³	³

For example, in Windows, you can use the "Num. Entity" column for [Alt] codes such as [Alt]946 for β (beta).

8.5.6 UTF-8: Encoding the Unicode Code Space

As Unicode is an encoding *system*, it depends on various character sets, such as UTF-8 (and UTF-16) for practical use.

UTF-8 (1993) is a variable-length 8-bit character encoding, which means that it can use one to four 8-bit bytes to represent each character. The first group of 128 characters in UTF-8 are the original 128 ASCII characters. This means that software configured to use Unicode will also be able to work with ASCII (ISO 646:1983) files.

The popularity of UTF-8 has increased since it was released. As of 2007, UTF-8 has became more dominant on the web than ASCII itself. UTF-8 is also the default encoding for HTML5 and JSON.

---

[15] *Unicode input*, Wikipedia, CC BY-SA 3.0

UTF-8 and UTF-16 are the standard encodings for Unicode text in HTML documents, with UTF-8 as the preferred and most used encoding.[16]

— Wikipedia *UTF-8*

8.5.6.1 Character Encoding Conversion Example

We can convert a file encoded as Windows-1252 into UTF-8 with `iconv`.[17]

Example 8.7 Converting a Windows-1252 file into UTF-8

```
$ iconv -f windows-1252 -t utf-8 asciitable.txt > asciitable2.txt
$ file asciitable2.txt
asciitable2.txt: UTF-8 Unicode text
```

As you can see, you can use `file` to verify that this is a Unicode file encoded as UTF-8.

Tip

"Normalize" text datafiles to a common, universal encoding format like UTF-8 to ensure characters are displayed with the intended symbols.

8.6 Data Structure

- Structured: Formal and rigorous design

  - Example: Relational database

- Semi-structured: Self-describing, validatable

  - Markup using tags or key-value pairs
  - Examples: XML and JSON

- Unstructured:

  - Multimedia and text document files
  - Any internal structure, if present, is assumed or unreliable
  - Example: email "body" ("header" is semi-structured)
  - May have "implied" structure, like "delimited text"

You can store data in structured, semi-structured, or unstructured formats.

A structured data formal is a rigorous design like you would have in a relational database. Semi-structed is self-describing and can be automatically validated using software. It is validated through markup or key-value pairs. Examples of semi-structured file formats are XML and JSON. By having this self-describing format, programs can be run which can validate the document and make sure that it's in the proper format. Unstructured documents include the majority of documents that you might be using in typical office work, such as multimedia and text document files.

---

[16] *UTF-8*, Wikipedia, CC BY-SA 3.0

[17] `iconv` is another tool originally developed for Unix, Linux and OS X systems, though Windows versions are available and can be found with an Internet search.

You might say, "My documents *are* structured — with paragraphs, sentences and words." The idea is that any type of internal structure such as this is *assumed*. You might not have been consistent in using your "format". Or maybe you and your office-mates do not agree exactly on the "company standard" format's details. Maybe you indent your paragraphs, but your co-worker does not. Maybe you leave one space between sentences or maybe you have two. Perhaps you made a "typo" by having two periods at the end of a sentence instead of one. A person would know this was just one sentence, but a program might see two. (A normal sentence and one with no words, but just a period.)

There is really no viable way to validate these kinds of documents, because there is no assurance that they conform to any specific and sufficiently detailed standard, so they are simply called "unstructured". They may be electronically parsed using heuristics and "artificial intelligence", but not as reliably as is if they were "self-describing" as to the *meaning* (symantics) of each part of the document. A self-describing format like XML would allow you to embed meta-data "tags" to indicate a person's name as an "employee", a place name as a "location", or a postal code as such and not just a series of seemingly random digits.

Another example of unstructured text is the body of an email, where you could type *any* text content into the body that you wanted. The email header, however, is semi-structured because an email header *does* consist of key-value pairs, much like a JSON document. So the email, taken is as a whole, is semi-structured, but the email body is unstructured.

A controversial example might be a comma-separated-value (CSV) file. There are delimiters (commas) separating the fields and there is a header line at the top "describing" the values in each column, (much like key-value pairs). So, since this seems sort of like a database table (structured) or at least similar to a JSON file (semi-structured) you might *not* think it is *unstructured*. But the structure is *implied*, not *explicit*. The document does not tell you that it is in any particular format. You are *assuming* this format based on the filename suffix (extension) of .csv, the prevalence of a lot of commas, or the assurance of the person who gave you the file. It might just contain a list of items that does not conform to any particular standard. You have some rows with more or less values, or that don't match the header line (if present). Any sort of structure is applied or assumed, but isn't reliable. You could open a CSV file with a variable number of columns per row, or without a header in many software packages without getting an error or warning. You would have to discover and address any such issues yourself. The CSV data "format" is not (universally) validated.

8.7 XML

- Self-describing[18]

- Structured

- Standard

- Parsable with libraries

- Examples:

  – XHTML
  – KML
  – XLSX
  – p-list
  – SVG

---

[18] *XML*, Wikipedia, CC BY-SA 3.0

Figure 8.8: XML - Image: Dreftymac, CC BY 2.5

XML is self-describing, standard, and parsable, meaning there are automated means (libraries) of reading it in — loading in the values and the keys into a data structure. Examples are XHTML and KML (used with maps), XLSX is the (newer) MS-Excel format, the p-list is an OS X "property list" used for configuration, and SVG is an image file format.

For example, XHTML is an XML-compliant form of HTML that includes an XML version number, with a DOCTYPE specification, and an "xmlns" namespace. The rest of the document looks a like like any other HTML, such as HTML5, but all of the tags are also XML-compliant so that the file can be validated as XML by an automated parser. It is self describing because the structure of the document and the meaning of the elements are encoded in these tags.

8.8   JSON

- JavaScript Object Notation

- Open format (ISO and ECMA standards)

- Human-readable text

- For transmitting data objects

- Attribute–value pairs

- Often used in Ajax web applications

Another self-describing, semi-structired file format is JSON. It is an open format and standard recognized by two standards bodies (ISO and ECMA). Like XML, is is human-readable because it is plain-text, (though some may find it more readable than XML). It is used for transmitting data objects, meaning sending data structures between

programs. It is organized into a nested structure of attribute:value (key-value) pairs. It is often used in Ajax web applications for transmitting data back and forth between the clients (web browsers or locally-installed "apps") and the server (server-hosted web application).

Example 8.8 JSON data structure for a person (John Smith)[19]

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "height_cm": 167.6,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

In this example, we have braces enclosing information about a person. Sub-sections are also enclosed by braces and/or brackets. The "address" attribute has it's value enclosed in braces and contains a list of attribute:value pairs. The "phoneNumbers" attribute has a value which is an array of lists of attribute:value pairs. The "children" attribute's value is an empty array and the value for the "spouse" attribute is "null" which means "no value". So instead of having tags to enclose data values, we have a braces, brackets, and attribute names to provide a nested structure and describe the data values.

8.9   Delimited Text Files

Files formatted with delimiter separated values use:

- Comma (e.g., "CSV")

- Tab (e.g., "TSV")

- Pipe (vertical bar: |)

... or other single character as a separator (delimiter) between values.

The CSV is the best-known of these, but TSVs are still very common, with pipe-delimited files being less common. A benefit of the latter type, however, is that the pipe symbol is the least likely to be found in most typical data values. Unlike the tab, the pipe (|) and comma (,) are visible to the eye.

Taking the example of a CSV, since commas are often found in data values, there needs to be a way to differentate the comma as the delimter in a CSV file and the comma present merely as data. This is done by "escaping" — adding more characters to indicate the special treatment. In a CSV, this is often called "comma-quote-delimited". But what happens when the escape character also needs to be escaped? Now you can see why using a tab or pipe makes the job a lot simpler, because there is far less need for adding escape characters. As you might imagine, the addition and parsing of these characters is more complicated and thus more prone to error (bugs). For this reason, a TSV may be a better choice than a CSV in many situations. Many data-import functions assume TSV by default.

The records (rows) are separated by line-ending characters (newlines):

- Carriage-return (CR)

- Line-feed (LF)

- Carriage-return, Line-feed (CRLF)

On the original MacOS systems, the standard line-ending character was the Carriage-return (CR), on Unix, OS X, and Linux systems, the line-ending character is a Line-feed (LF), and on Windows, both are combined together as Carriage-return, Line-feed (CRLF). As you might guess, (or have experienced first hand), sharing files between these systems can create problems, so various automatic conversions are built into file-transfer applications. You may see a prompt during software installation (such as for Git) as to whether or not you wish for such conversions to take place by default.

For example, if the server is running Unix, and some of the clients run OS X and some run Windows, the technique is often to configure the client software to standardize on the line-ending format of the server. In this case, the server running Unix and the client running OS X would both use LF, but the Windows client would need to convert LF to CRLF when receiving files and then convert CRLF back again to LF when sending files back to the server.

8.10   Fixed-Width Text Files

Instead of using a single character as a delimiter, you can also neatly line up your columns into a format called "fixed-width". The space between columns is filled with as many "whitespace" characters (spaces or tabs) as needed. Because the number of these delimiting characters can vary from one field to the next, this format is a little harder to parse (automatically), but has the benefit of greater (human) readability. The lines (records, rows) are separated by newlines. Since the columns must line up, the fields must be limited in length, often to some arbitrary value. For this reason, this format is not as commonly used as a data exchange format, but is often seen in text-based tabular reports. Since databases often enforce field-length restrictions anyway, a fixed-width format may be used by such systems for data storage.

Here is an example:

```
                  mpg cyl disp
Mazda RX4         21.0   6  160
Mazda RX4 Wag     21.0   6  160
Datsun 710        22.8   4  108
Hornet 4 Drive    21.4   6  258
Hornet Sportabout 18.7   8  360
Valiant           18.1   6  225
```

(Data from mtcars, *The R Datasets Package*, R Core Team.)[20]

---

[20] Extracted from the 1974 *Motor Trend* US magazine. Source: Henderson and Velleman (1981), Building multiple regression models interactively. *Biometrics*, 37, 391–411.

You can see that the columns are neatly aligned and the table is more readable than if each value was sepated by a single comma or space.

8.11 Multi-line Text Files

You should know that there are also multi-line text file formats. Some popular genomics file formats use multi-line records, for example.

If you look carefully at these, you'll see that each record starts off with a header line.

- FASTA[21]

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGTNLV
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLG
LLILILLLLLALLSPDMLGDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGX
IENY
```

In the case of the FASTA file, the record's first line begins with an angle bracket (>) followed by a pipe-delimited string containing various codes, and finally ends with a name enclosed in brackets. After this first header line, you have a few lines that look like random gibberish, but they are meant to represent the biomolecular sequence information.

- FASTQ[22]

```
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACC
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=36
IIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9IC
```

This FASTQ record's header line begins with an "at symbol" (@) followed by an alphanumeric code and a space-delimited list of values. The first value appears to be another code containing underscores and colons as delimiters, but it is hard to be sure from just looking at it. The last value is actually a key=value pair specifying that the string of nucleotides listed on the next line will contain 36 letters. The third line is another header, just like the first line, but starting instead with a "+". That difference can be used when parsing the file to know the difference between the two types of headers.

Unfortunately, the fourth line, which is the "Phred quality" line, contains a quality-character for each nucleotide listed in the second line. A variety of alphanumeric and punctuation characters are used for Phred quality codes.[23] This could lead to parsing problems, especially with fragmented (incomplete) records. What if a particular quality line starts with a "+" or an "@"? A parsing program might assume this is a header line and attempt to read it as a new record.

Are these multi-line text files structured, semi-structured, or unstructured? As before, there is no explicit information in the file telling you which format or standard the file structure would adhere to, and there is nothing self-describing

---

[21] *FASTA format*, Wikipedia, CC BY-SA 3.0
[22] *FASTQ format*, Wikipedia, CC BY-SA 3.0
[23] See FASTQ format encoding, Wikipedia

about the contents of the file, with the exception of the "length=36" part. So, like delimited and fixed-width files, these are actually "unstructured", despite a lot of implied structure, or "structure by convention".

Regardless, unstructured files are still very useful and are usually structured enough to serve their purpose. The only reason we have been drawing these distinctions about structure has been to help us categorize and evaluate the various alternatives — to appreciate the tradeoffs and provide clues as to possible processing strategies.

8.12   Data File Layout: Tidy Data

Now that we have an understanding of the types of file formats, we will discuss a little of what to *do* with data. In particular, we will see that following a few simple *layout* guidelines can help immensely when it comes time for data analysis later on.

Whether your data comes from an XML, CSV, or other file format, you can hopefully view it as a single table of columns and rows, with a single value in each "cell" (row-column intersection). If so, that should be fairly tidy.

8.12.1   A Table of Columns and Rows

We want to structure each data file as a single table of "columns and rows" . . .

Table 8.2: A table of columns and rows

subID	height	weight
1	58	115
2	59	117
3	60	120

. . . to make them easier to import and analyze.

(Data from women, *The R Datasets Package*, R Core Team.)[24]

If you need to, you can link multiple tables together on common "key" columns, once you have imported them into your database or analysis software.

8.12.2   Tidy Data Basic Tenets

The basic tenets of *tidy data* are:[25]

- Each variable forms a column.

- Each observation forms a row.

- Each type of observational unit forms a table

---

[24] *The World Almanac and Book of Facts*, 1975. Reference: McNeil, D. R. (1977) *Interactive Data Analysis*. Wiley.
[25] Hadley Wickham, Tidy Data

8.12.3 Tidy or Not?

Is this spreadsheet[26] *tidy data* or not? Why or why not?

GBD code	GBD cause (b)		Afghanistan	Albania	Algeria	Andorra	Angola	Antigua and Barbuda	Argentina
		Data sources - level of evidence							
		All cause mortality (c)	Level 4b	Level 2b	Level 3b	Level 3b	Level 4b	Level 3b	Level 1a
		Cause-specific mortality (d)	Level 4	Level 2b	Level 4	Level 4	Level 4	Level 2b	Level 2a
		Population ('000) (e)	22,930	3,141	31,266	69	13,184	73	37,981
W000	**All Causes**		484.5	22.1	173.3	0.6	306.6	0.6	281.4
	Lower uncertainty bound (f)		347.6	19.3	158.0	0.5	255.3	0.5	276.6
	Upper uncertainty bound (f)		695.8	28.0	188.4	0.6	434.8	0.6	286.6
W001	*I.*	*Communicable, maternal, perinatal and*	317.2	1.8	56.9	0.0	230.3	0.1	36.2
W002		A. Infectious and parasitic diseases	169.3	0.2	30.2	0.0	143.0	0.0	15.6
W003		1. Tuberculosis	21.1	0.0	0.6	0.0	4.8	-	0.8
W004		2. STDs excluding HIV	0.3	0.0	2.1	0.0	2.3	0.0	0.0
W005		a. Syphilis	0.1	0.0	2.0	0.0	2.3	0.0	0.0
W006		b. Chlamydia	0.1	-	0.0	-	0.0	-	-
W007		c. Gonorrhoea	0.0	-	0.0	-	0.0	-	-
W009		3. HIV/AIDS	0.0	0.0	0.3	0.0	21.1	0.0	1.8
W010		4. Diarrhoeal diseases	41.2	0.0	8.1	0.0	48.8	0.0	0.4
W011		5. Childhood-cluster diseases	15.5	0.0	4.1	0.0	12.4	0.0	0.0
W012		a. Pertussis	6.6	-	0.6	-	4.2	-	0.0
W013		b. Poliomyelitis	0.0	-	-	0.0	-	-	-
W014		c. Diphtheria	0.0	-	0.0	-	0.1	-	-
W015		d. Measles	2.2	-	3.4	-	6.2	-	-
W016		e. Tetanus	6.7	0.0	0.0	0.0	1.9	0.0	0.0
W017		6. Meningitis	8.3	0.0	0.2	0.0	0.6	0.0	0.4
W018		7. Hepatitis B (g)	1.0	0.0	0.5	0.0	0.8	0.0	0.1
W019		Hepatitis C (g)	0.5	0.0	0.2	0.0	0.4	0.0	0.1

World Health Organization / Organisation Mondiale de la Santé — Department of Measurement and Health Information — December 2004 — Table 1. Estimated total deaths ('000), by cause and WHO Member State, 2002 (a)

We have what looks like columns and rows, but you might notice that there are apparently several rows of headers at the top (with levels and sublevels) and a nested hierarchy of "causes" on the left (listed in outline form). It would be confusing to import this table into data analysis software that did not know how to interpret those various levels.

There is also color-coding for various columns and rows, which we presume must carry some additional meaning, but which might easily be lost upon import into our preferred software application.

How would you go about tidying this dataset?

8.13 Wide and Long: Which Table is Tidier?

Once we have a table in a nice column and row format, we may further tidy the data by checking that each column should indeed be a column — or rearrange the table as needed to achieve this.

A table layout can be seen as "wide" or "long". Wide data has lots of columns (variables) and long data has lots of rows (cases, records, observations).

Here is an example in "wide" format:

Table 8.3: Iris data in "wide" format

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
5.7	2.8	4.1	1.3	versicolor

---

[26] WHO

Here the same table has been rearranged into "long" format:

Table 8.4: Iris data in "long" format

Flower.Id	Species	Flower.Part	Length	Width
1	setosa	Petal	1.4	0.2
1	setosa	Sepal	5.1	3.5
100	versicolor	Petal	4.1	1.3
100	versicolor	Sepal	5.7	2.8

(Data from iris, *The R Datasets Package*, R Core Team.)[27] '

The "long" layout may be tidier if you consider the "species" such as *versicolor* or the "part" of a flower such as *Sepal* as observations rather than as a variables. But in many cases, both wide and long formats are fine if they make sense and suit your analytical needs. The main idea is that you want to spend a little time tidying data so that you don't waste a lot of time trying to analyze untidy data.

8.14   Why does tidiness matter?

If "wide" or "long" layout choices seem rather "subjective", consider which layout will better facilitate your analysis.

Because we have "tidied" our example data into a "long" layout, we can easily "facet" our plot by `Species` and `Flower.Part`.

```
ggplot(data=iris, aes(x=Width, y=Length)) +
    geom_point() + facet_grid(Species ~ Flower.Part, scale="free") +
    geom_smooth(method="lm") + theme_bw(base_size=16)
```

---

[27] Anderson, Edgar (1935). The irises of the Gaspe Peninsula, *Bulletin of the American Iris Society*, 59, 2–5. Reference: Fisher, R. A. (1936) The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, Part II, 179–188.
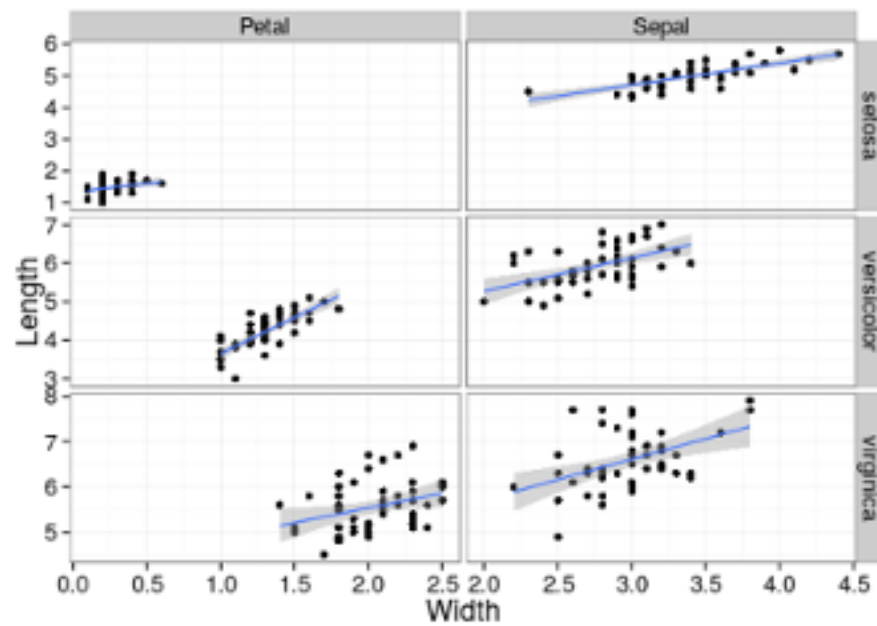
Figure 8.9: Plot using tidy iris data

8.15 Conclusion

In this module, we have explored file naming, types, and organization. We hope that by gaining an understanding of key concepts of data files, that you are better equipped to tackle your data-wrangling needs. We will build upon this knowledge in later data management modules, such as "databases", "programming", and "analysis tools".

Chapter 9

Databases

Research produces data, and information that need to be stored for future review or processing. Databases are one such solution, offering an integrated system of information storage, and management. There are a variety of database systems on the market, each with their own pros and cons.

Databases come in all shapes and sizes. In general, there are three basic types. The *flat file*, which is much like a spreadsheet. *Relational*, which is like a series of spreadsheets linked by common identifiers. And finally, the family of *NoSQL* databases, which have various storage models for different use cases.

Much like the variety of database types, there are a variety of query languages. A query language is how you interact with the database. With it, you can insert new data, update records, etc. We'll go over some specific languages like Structured Query Language later in the section.

- Types

  - Flat
  - Relational
  - NoSQL (various storage models)

- Query Languages

  - SQL
  - CQL
  - Etc.

9.1   Key Terms

Before we can discuss the details of each database type, and their related query languages, we have to get some key terms out of the way. These terms will come up again later, in addition to being found in numerous database books, and articles.

CRUD represents the four most basic database operations. Create refers to creating new entries, or adding new data to the database. Read, as the name implies is retrieving data from the database. Update is simply changing the contents of an existing database record. And, Delete, is the removal of a record.

ACID is a method of ensuring that data is stored reliably. Atomicity or an atomic operation, means either the transaction or query is completed fully, or not completed at all. In other words, if the query fails, the database must not be

changed. Consistency means that database must remain in a valid state. If the database has any data validation rules, all data must be in compliance with them. Isolation is important in multi-user databases. With Isolation, each users transactions must not interfere with each other, so that each user has a consistent view of the database. Durability means the data has been committed to disk. So, that in the event of a power failure, crash, etc. your data is safe.

Normalization is all about removing redundant data. For example instead of storing a customer's contact information with their order. The order would refer to a customer ID number, which refers to their contact information. Thus, multiple orders can refer to the same data.

CAP Theorem is mostly relevant to distributed NoSQL databases. That is databases which are spread across many servers. It states that you can not deliver a consistent database view, ensure that every query is responded to, and tolerate a failure in the system at the same time. In other words, you must compromise in one of those areas, to guarantee the other two.

Database Schema is a formal description of the database structure. It defines the tables, data types, and other rules governing the database.

9.2   Flat File

Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

Figure 9.1: Flat File Model: US Department of Transportation, Public Domain

Flat file databases are extremely simple. They are simply a single table of values, just like a spreadsheet. The actual data format varies, there are proprietary formats like Excel, but open standards like CSV or comma separated values, are widely seen.

Due to their simple nature, they lack a number of features found in more capable database systems. For example, they lack indexing, which means that searching a large flat file could require considerably more time than a relational database. They also lack concurrent write access, since they aren't server based. Finally, updates can be extremely slow, as they require rewriting the entire file to disk after making a change.

All these limitations aside, there are some benefits. Flat files can be very lightweight in resource usage, at least when they are small in size. For databases using CSV or similar formats, you can even edit the database using a basic text editor.

Despite the limitations, there are situations where a flat file is a reasonable choice. For example, recording sensor data from a temperature and pressure sensor. This data tends to consist of timestamped rows with sensor readings, with each file named after the sensor.

That said, it's generally best to not go with a flat file database for storage. If at some point you find that your needs change, converting to a more capable format, could become a rather daunting task.

- Lightweight to a point

- Limited scalability

- No built in concurrent access

- No indexing

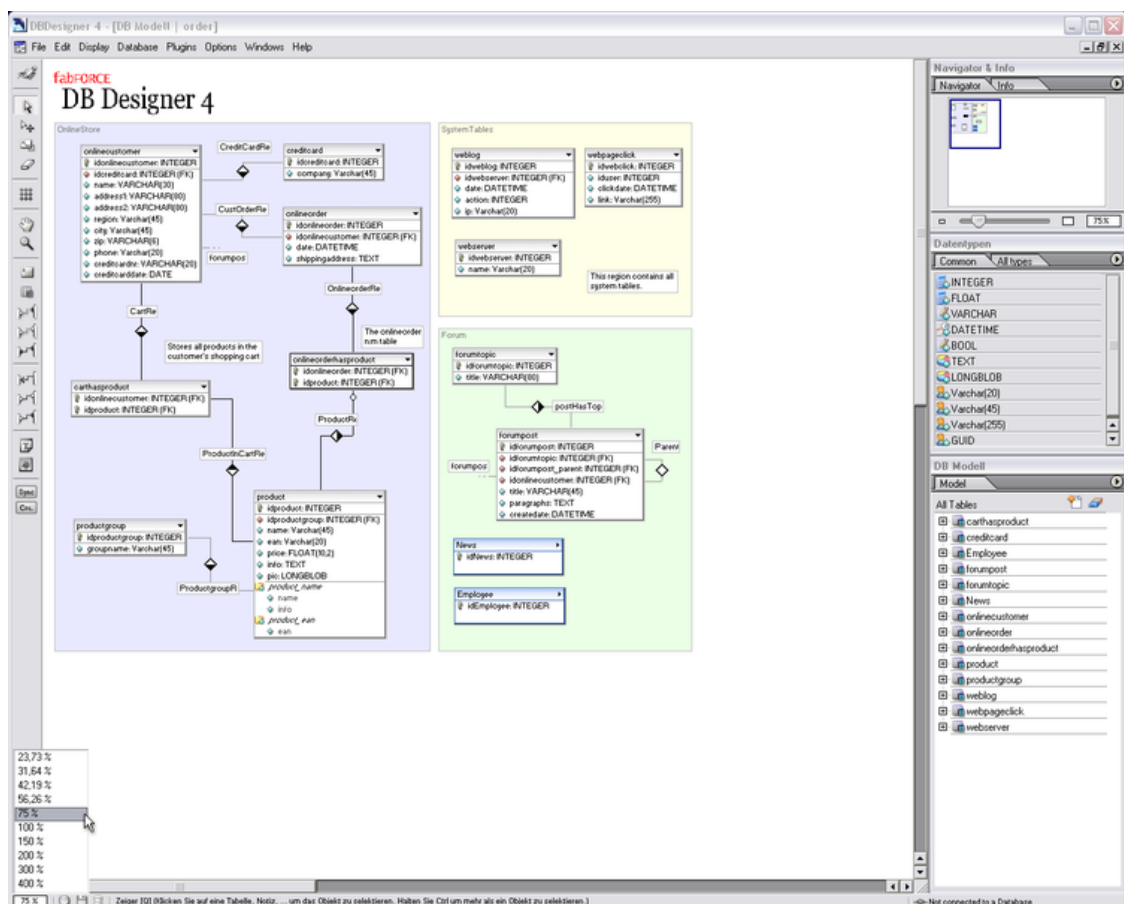- Lacks built in consistency checking

9.3   Relational

Figure 9.2: fabFORCE.net DBDesigner4: MichaelGZinner, CC BY SA 3.0

Relational databases, as their name implies, relate data from one table to another. These relations reduce the duplication of data that is common to multiple records. For example, in an ordering system, you could store your customers within one table, and link their orders to them via a customer identification number.

Relational databases are basically the jack of all trades in the database world. They are highly flexible enabling them to meet the widest number of use cases, such as recording survey results. But, that flexibility comes at a price. You have to tell the database about the data you plan to store. In other words, a relational database requires a schema to define the structure of your data, and how it relates.

Virtually all of them support ODBC or Open Database Connectivity, which is a standardized interface between an operating system or programming language and the database driver. Through ODBC, you can plug your database into a huge number of applications, such as *R* or even *Excel*.

There are variety of relational database engines on the market, some open source, others proprietary. The most common open source database is *MySQL*, which is now owned by Oracle. For those with very small needs, there is *SQLite* which is commonly embedded into other applications like Firefox and Android. Even Microsoft has an offering, in the form of Microsoft *SQL Server*.

Example 9.1 Does climate impact human health?

A database can be used to store any kind of data. One group had the challenge of analyzing a large volume of climate data to compare with health records. The health care data was location coded, in a similar fashion to the climate data that was being used. Through the use of MySQL, they were able to relate health care records by location with the climate data. Thus enabling them to query the data, and look for patterns.

So, which should you choose? My advice is to pick an open source option, like MySQL or *PostgreSQL*. They have low minimum resources making them easy to setup on a laptop for testing purposes. Documentation is widely available. And, more importantly, their free nature means, a lot of folks have taken the time to integrate them into a huge array of tools.

That said, different engines make sacrifices to make gains in other areas. This commonly occurs in the area of ACID compliance. A database may forgo strict data validation, or consistent disk writing to speed up query response time. If you value your data over speed, be sure to pick an ACID compliant database.

In short, if you aren't sure what kind of database to use, an ACID compliant relational database is a good start. It's typically easier to migrate from a relational database to other types, as opposed to converting to one.

- General purpose

- Wide application support through ODBC

- Less duplicate data

9.4 NoSQL

NoSQL or "Not Only SQL" databases serve a variety of niches. Each implementation is geared towards a limited set of use cases, in other words, they don't try to be general purpose databases. NoSQL databases are most commonly associated with Big Data, and High Performance Computing. By Big Data, we're talking about databases in the multi

terabyte and larger range. But, not only are they big, they generally require high throughput either from thousands of concurrent requests or simply doing a query across a few billion rows of data in a short period of time.

Within the NoSQL arena, there are a lot of options. A couple commonly used are Apache's *Cassandra*, which functions similarly to a traditional database. It has tables, which consist of rows and columns. However, the method you query across multiple tables is a bit different than a relational database. *mongoDB* on the other hand, is known as a document store. A document being an object containing a series of values. Where a value could be as simple as a name, to an entire PDF. In addition, objects can be related to each other through their unique IDs.

Much like there are a number of NoSQL databases, each excelling in their own ways, there are a variety of storage models. A storage model is how the data is actually stored within the database. Each storage method is best suited to select use cases, or applications. You'll want to consult the documentation for each database to see what it's most suited for. You may also want to look to sites like BioStar to see what others in your field are using to solve their problems.

With all this variation in data storage methods, there has to be a catch. While many relational databases offer ACID compliance, NoSQL databases rarely do. They make many sacrifices in order to scale massively. The guiding rule with NoSQL is the CAP Theorem which basically says you can't ensure every server shows the exact same data, handle every query, and not lose access to something in the event of a crash.

Example 9.2 Collecting data from remote sensors.

While NoSQL implies Big Data, it doesn't require it. NoSQL databases lend themselves to solving a variety of problems. One group was tasked with performing remote data collection using Internet connected sensors. Each of these sensors sends data back in 30 second intervals to KairosDB. KairosDB is a time-series database, designed for tracking sensor data over large periods of time, and from a large number of sensors. Being a purpose specific database, it's easily integrated with tools used for trend analysis.

So, which NoSQL option to go with? That isn't an easy question to answer. Since each database system varies from the next, you have to closely evaluate your needs. You need to consider what questions you are asking from your data. And, also factor in, what is your data. For example, is it a huge table of values, or millions of documents. Finally, look at the other tools you are using. They may include support for one or maybe two options. In which case, you are either stuck with those choices, or reconsider the tools being used.

9.5   Query Languages

A database isn't very useful if you don't have a way to enter, and retrieve data from it. A query language enables you to enter data, select data based on criteria, change existing data, and even remove it.

The most prevalent is SQL or *Structured Query Language* which is a standardized language implemented by most relational databases. However, it's not alone in that arena. Some databases such as *Oracle*, and PostgreSQL have their own add-ons called *Procedural Languages* which enable you to leverage scripting languages like *Perl* within the database engine itself.

Outside of the relational database world, the languages differ from one database engine to the next. For example, Apache's Cassandra offers *Cassandra Query Language* which has an SQL inspired syntax.

9.6   Structured Query Language

SQL or Structured Query Language is the most widely used database query language. With SQL, you can query for data, create new databases, etc. At the core, SQL is an ANSI standardized language, enabling multiple database vendors to implement it. That said, SQL itself doesn't meet everyone's needs, resulting in proprietary extensions to it.

When possible, it's best to avoid the use of proprietary extensions. Otherwise, in the future should you need to change your database back-end, for example switching from MySQL to PostgreSQL. You may find yourself rewriting a lot of you queries, and adjusting your database schema.
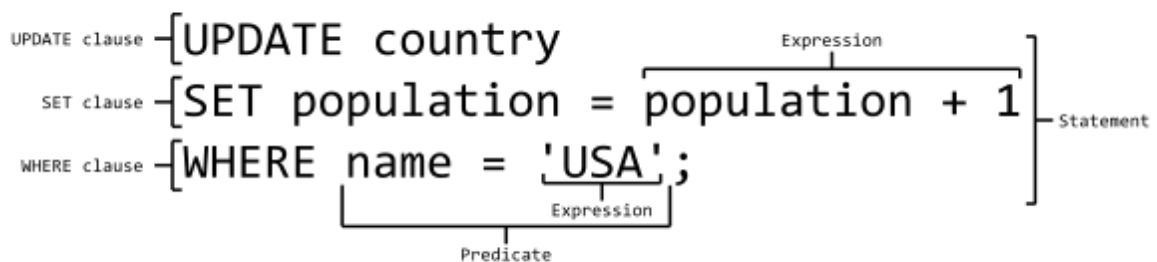

Figure 9.3: SQL Statement Anatomy: Ferdna, CC BY SA 3.0 MIGRATED

So, what does SQL look like? It has a lot of commands, though there are 4 basic commands which you'll want to get most comfortable with. The SELECT command, which is used to select or retrieve data from the database. The INSERT command, which as the name states, is used to insert new data. UPDATE, which is used to change values within the database. And, DELETE, which removes records from the database.

- ANSI Standard

- Proprietary Extensions

- Widely Supported

9.7   Other Query Languages

Among the numerous non-standard query languages, here are a few examples. Apache's Cassandra implements their Cassandra Query Language, which is SQL inspired. Oracle, Microsoft, and others offer Procedural Languages which embed some kind of scripting language within the database. Procedural Languages enable you to complex operations using things like loops and arrays within the database engine itself. By embedding the language into the engine, you gain performance from not having to transmit data across the network. As well, gaining capabilities like having some code executed upon a database insert or update.

In short, it's best to use a standardized language like SQL whenever possible. However, you shouldn't hinder yourself when the database engine offers an option that will improve performance or reduce data processing time.

- CQL: Cassandra

- PL/SQL: Oracle

- Transact-SQL: Microsoft SQL Server

9.8 Summary

We've gone over quite a bit of basics on databases. So, let's review the key points to remember.

CRUD (or Create, Read, Update, Delete) are the basic operations performed on a database. In SQL these operations are represented with INSERT, SELECT, UPDATE, and DELETE.

ACID (or Atomicity, Consistency, Isolation, Durability) compliance controls how data is stored. It ensures that all queries are atomic, e.g. all or nothing. That data stored is in full compliance with any validation rules. And, that database changes are really stored on disk.

Normalization is the process of reducing duplication in a database by relating records, instead of repeating the same information in each record.

Schema is the design and specifications for the database. It defines tables, content types, etc.

CAP Theorem applies mostly to NoSQL. It states that all database servers in a group can't show the exact same data, respond to every query, and tolerate a node failure without compromising in one of those areas.

We covered the three basic types of databases, flat, relational, and NoSQL. With flat files being generally frowned upon unless you have specific reasons for them. NoSQL typically only required when working with big data. And, relational being the most general purpose database you can use.

We've also covered ODBC, or Open Database Connectivity. Which is a standardized scheme for connecting applications, and languages with databases. In the Java world, the equivalent is JDBC or Java Database Connectivity. Regardless of language, these interfaces are your friend.

Finally, we covered some query languages. SQL being the most common query language, makes it a valuable tool to know how to use. If you plan to stick with a single database engine for a lot of projects, it may benefit you to become familiar with it's specific language. For instance, if working with Oracle a lot, you'll probably want to know PL/SQL.