Research Computing

Contents

Chapter 1

Introduction

This resource contains teaching materials for an overview course in **research computing**.

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 files hosted in the repository consist of the presentation slides in Markdown format as well as transcripts to go with those slides. The transcripts are posted as wiki pages in ASCIIDoc format and are also offered as PDF and EPUB eBooks.

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(http://en.wikipedia.org/wiki/Peripheral) 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.

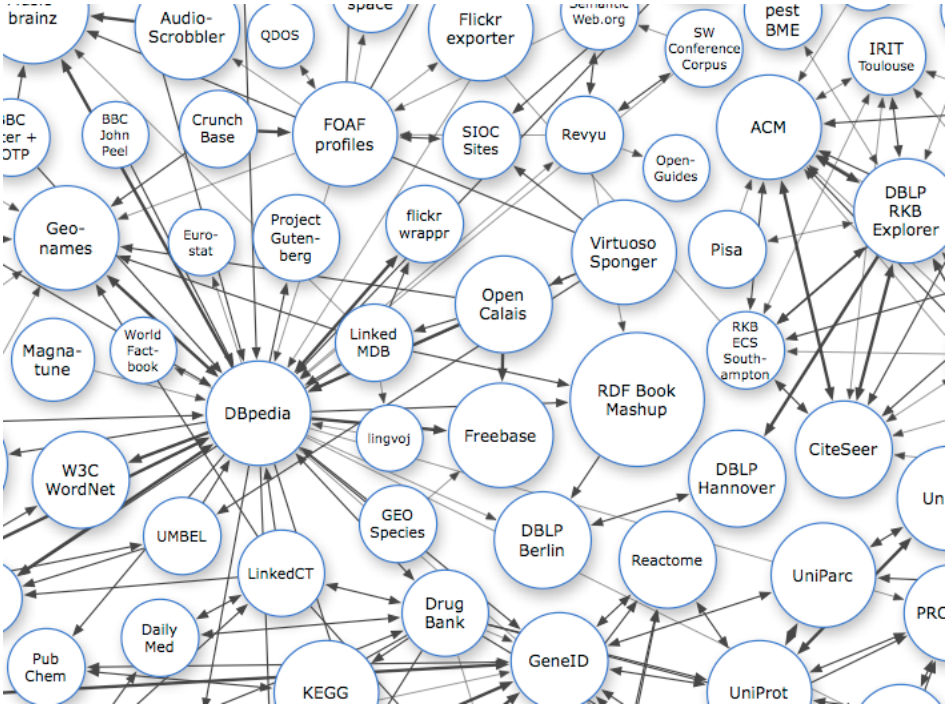

Figure 2.2: Linked Data - Image: linkeddata.org, CC BY-SA 3.0

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 *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.

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

[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/.

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.

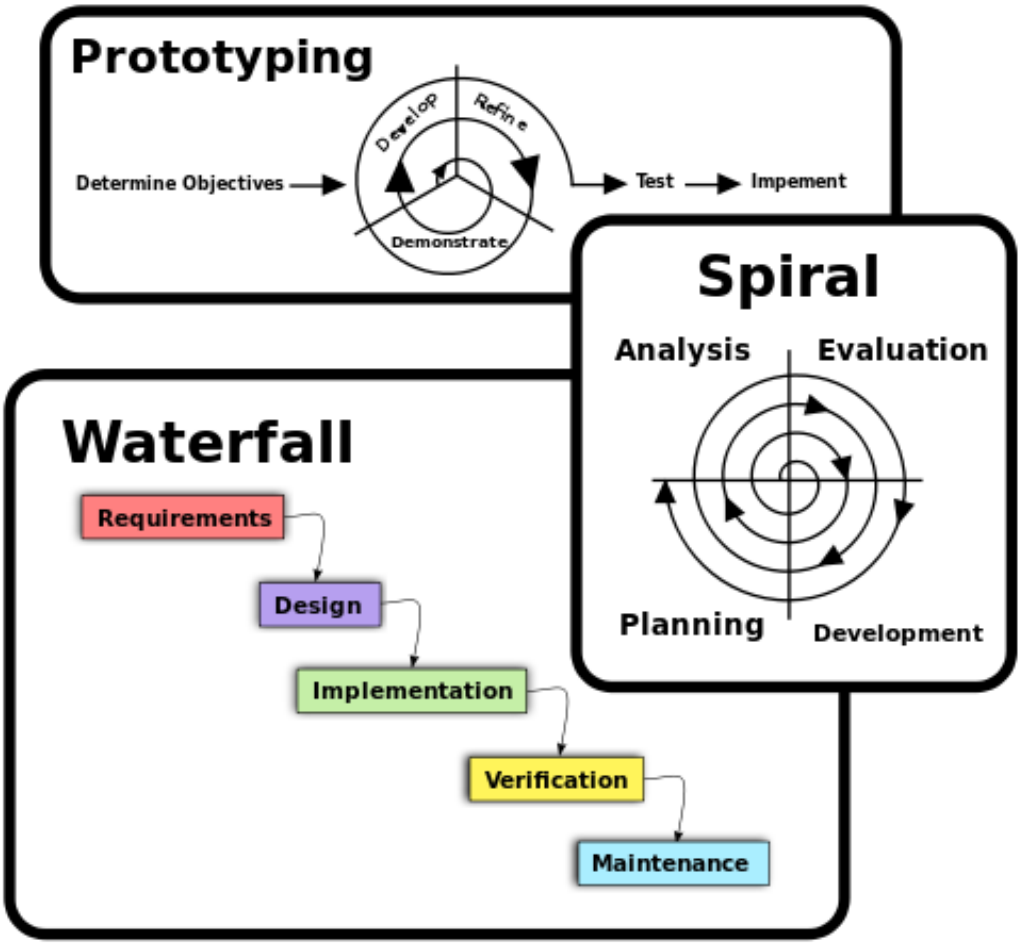

Figure 2.4: Three software development patterns mashed together - Image: Beao, Paul Smith, Public Domain

---

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.

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

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

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.

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

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 gaining approval from stakeholders.
- Analysis
  - *Analysis* includes a detailed study of the current system and clearly identifying 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.

[6] *Waterfall model*, Wikipedia, CC BY-SA 3.0

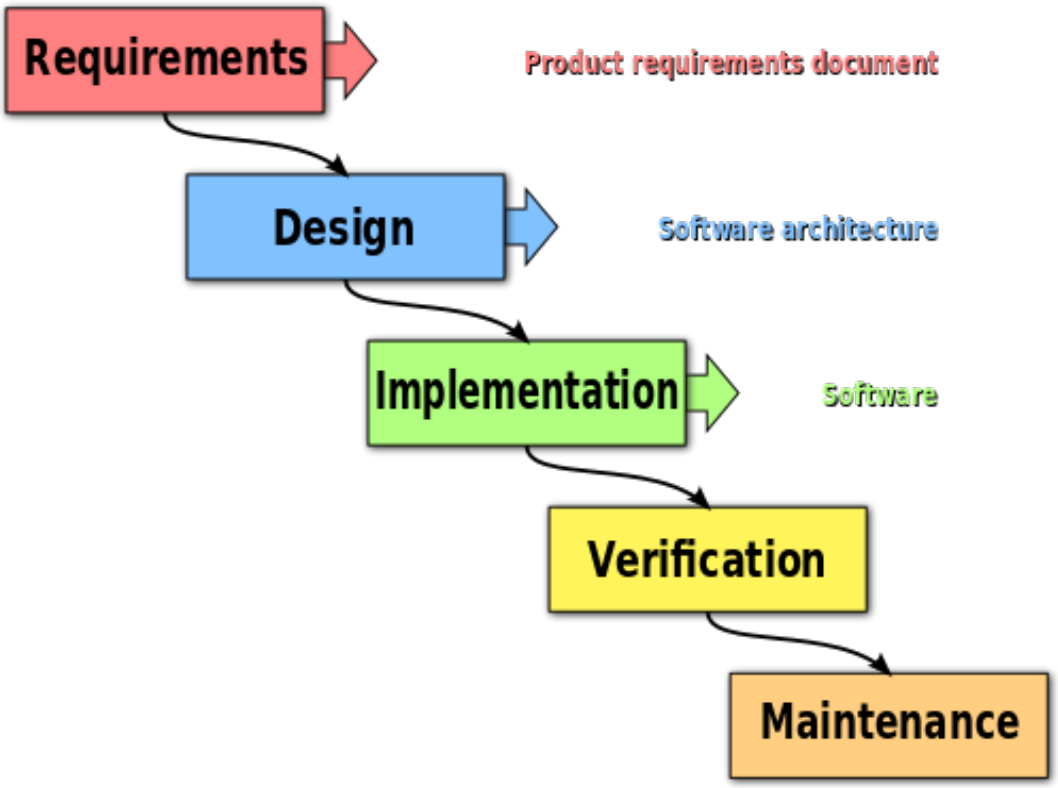

Figure 2.6: Waterfall model - Image: Peter Kemp / Paul Smith, CC BY 3.0

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, but very popular, especially 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

Smaller teams that can meet regularly, ideally face-to-face. Working in pairs, with one person coding and 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]

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

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 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.

These interoperability aspects will include file formats, communications protocols, security and encryption.

All of those are 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 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!

[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 the 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]

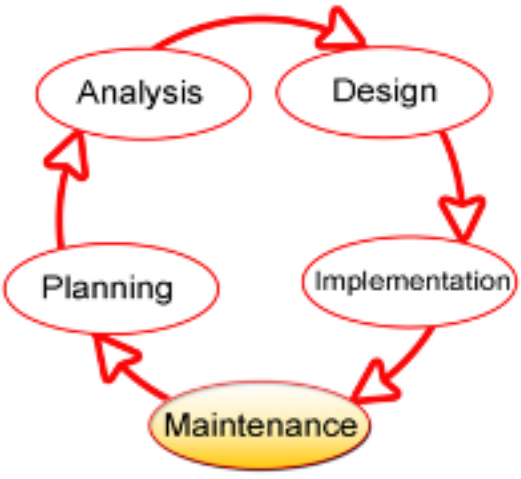

Figure 3.1: SDLC - Image: Dzonatas, CC BY-SA 3.0

[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."footnote:_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"[2]
- *Analysis* means "to take apart"[3]

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.[4]
>
> — 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[5] has its own series of phases. We will look at each of these one by one.

[2] *System*, Wikipedia, CC BY-SA 3.0
[3] *Systems analysis*, Wikipedia, CC BY-SA 3.0
[4] *Systems Analysis and Design for the Global Enterprise 7th ed.*, by Lonnie D. Bentley, as quoted by Wikipedia
[5] *Systems analysis - Information technology*, Wikipedia, CC BY-SA 3.0

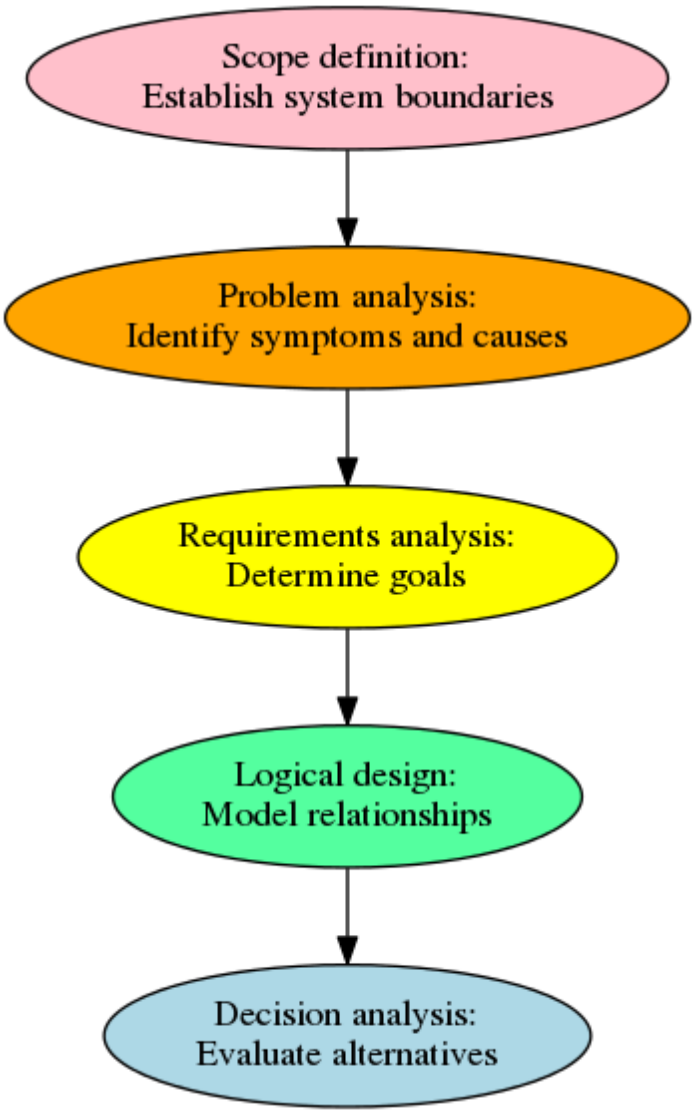

Figure 3.2: Systems Analysis Phases - Image: Brian High, CC-BY 4.0 International

---

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*?[6]

> **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)*

[6] *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*.[7]

> **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[8] 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?*

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

[8] 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" ...

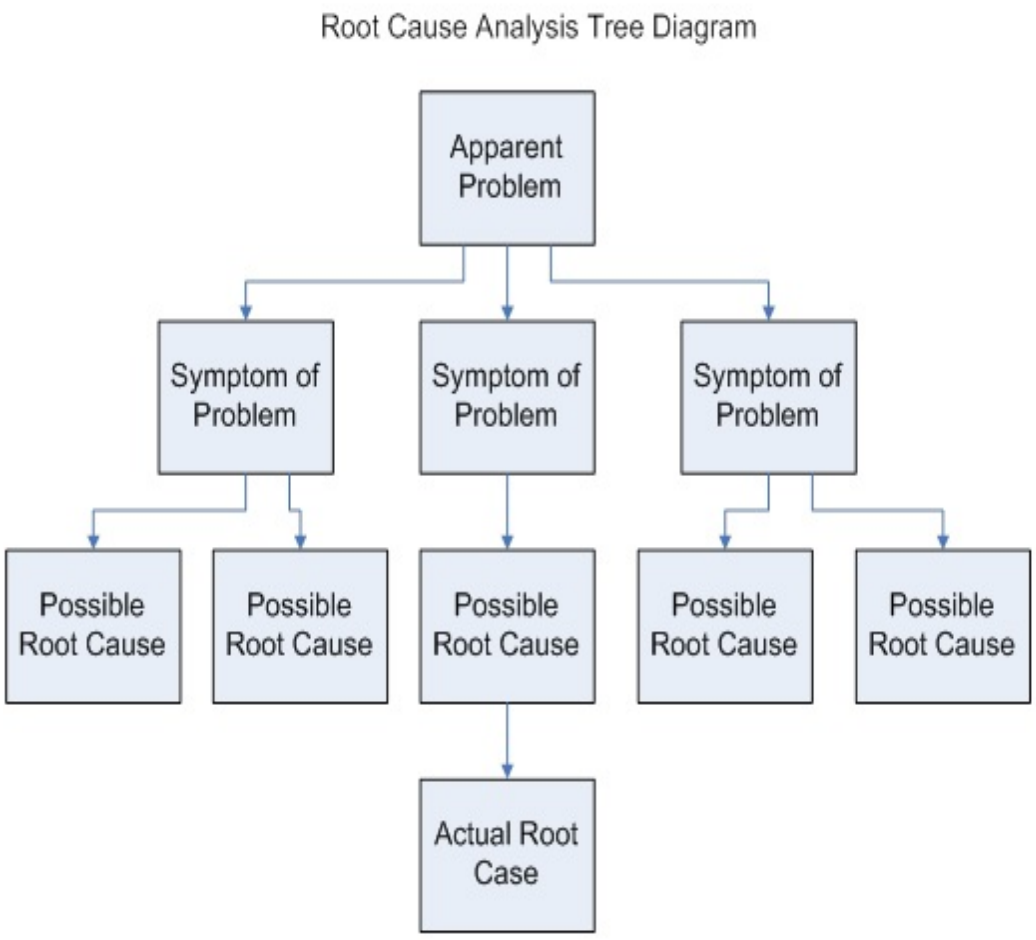

Figure 3.3: Root Cause Analysis Tree Diagram - Image: KellyLawless, CC BY-SA 3.0 Unported

---

... 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.[9]

This method may also be used in Requirements Analysis.

3.9 Requirements Analysis

- Elicit, Analyze, and Record (EAR):

[9] 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.[10]

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.

[10] *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 forms 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.

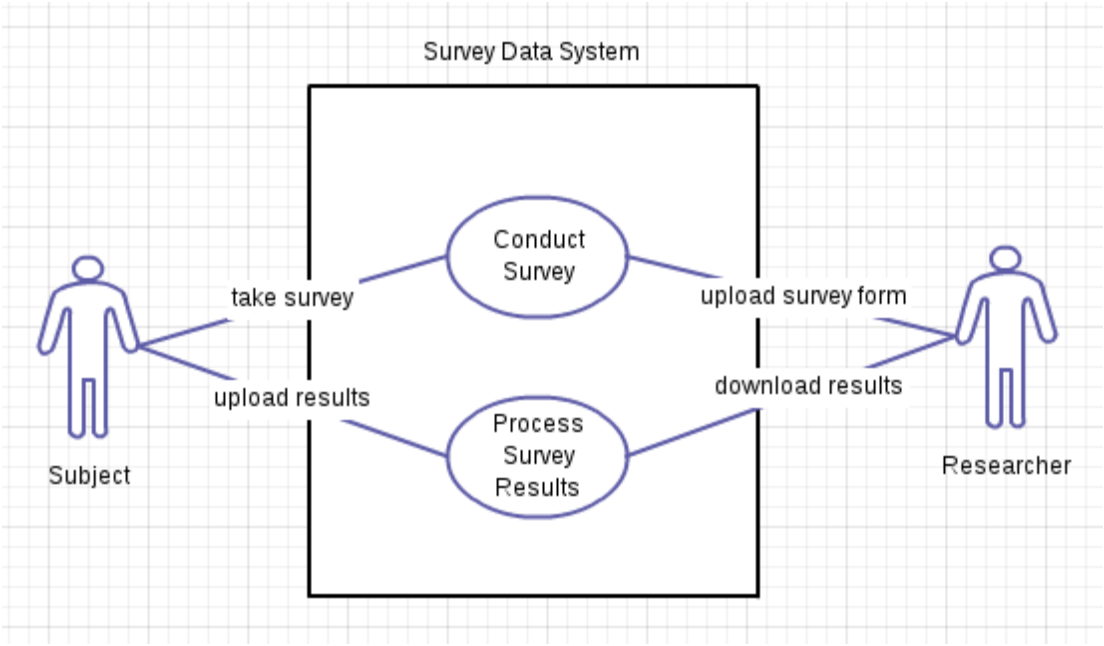

Figure 3.4: Research Survey Data System - Image: Brian High, CC0 1.0

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*.footnote:_Systems design_ - *Logical design*, [Wikipedia, CC BY-SA 3.0] 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 function 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 systems 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.

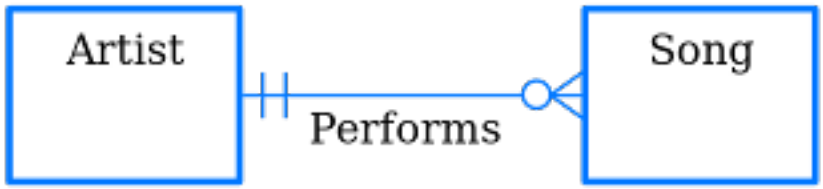

Figure 3.5: ERD artist performs song - Image: Bignose, Public Domain

---

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[11] (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.

[11] See also: *Data Management*, UW Canvas.

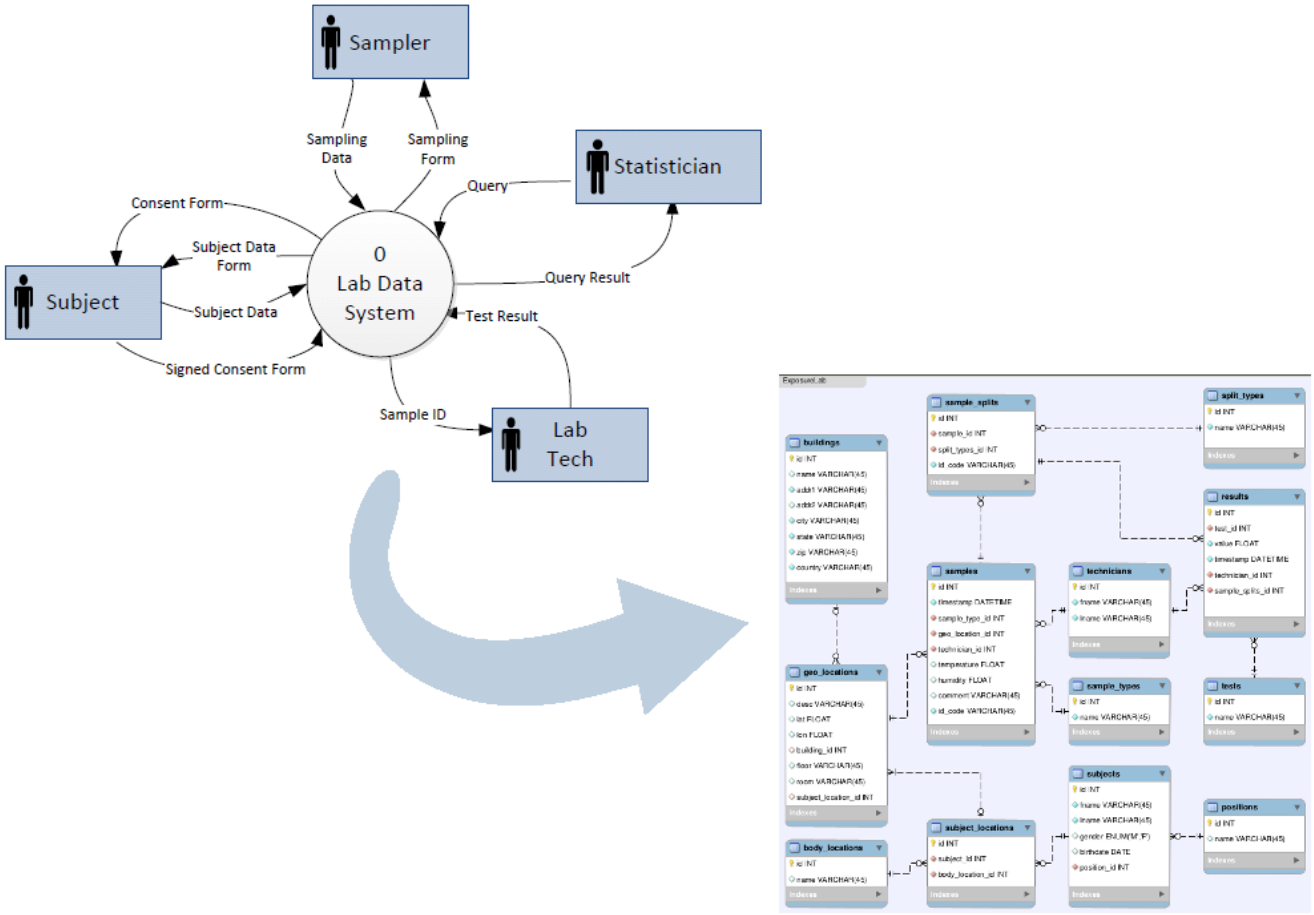

Figure 3.6: DFD and ERD - Image: Brian High, CC0 1.0

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.[12]

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.

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