3. REQUIREMENTS GATHERING

Last Mod: 2024-05-05a

The requirements gathering and analysis process is:

- Learn the application domain.
- Gather information about the specific application and how it will be used by the target user(s). I.e. “Requirements elicitation”
- Re-organize that information into a technical form that is most useful to later designers and programmers who will come on the project.

- Document in a requirements spec document.
  - The requirements spec is heavily reviewed by both customer and designers.
  - It often forms part of, say, a $1M legal contract describing the scope of the development.

Table of Contents

3. REQUIREMENTS GATHERING 1

3.1 REQUIRED READINGS 4
3.2 MOTIVATION FOR ANALYSIS 5
3.3 DON'T EXPECT THINGS TO BE FAMILIAR 9
3.4 THE DIFFICULTIES OF ANALYSIS 16
3.5 THE ROLE OF THE ANALYST 21
3.6 KEEP THESE THINGS IN MIND 31
3.7 USE CASES 39
3.8 REQUIREMENTS MEETINGS 44
3.9 APPENDIX A: WHERE TO FIND REQUIREMENTS INFORMATION 47
3.9.1 Written Materials 48
3.9.2 Interviews 50
3.9.3 Existing Forms 54
3.9.4 Screens 65
3.9.5 Visit the Existing System 66
3.9.6 Measurements 72
3.10 APPENDIX B: "ANALYST WANTED, PROGRAMMERS NEED NOT APPLY" 79
3.11 REFERENCES 87

3.1 Required Readings

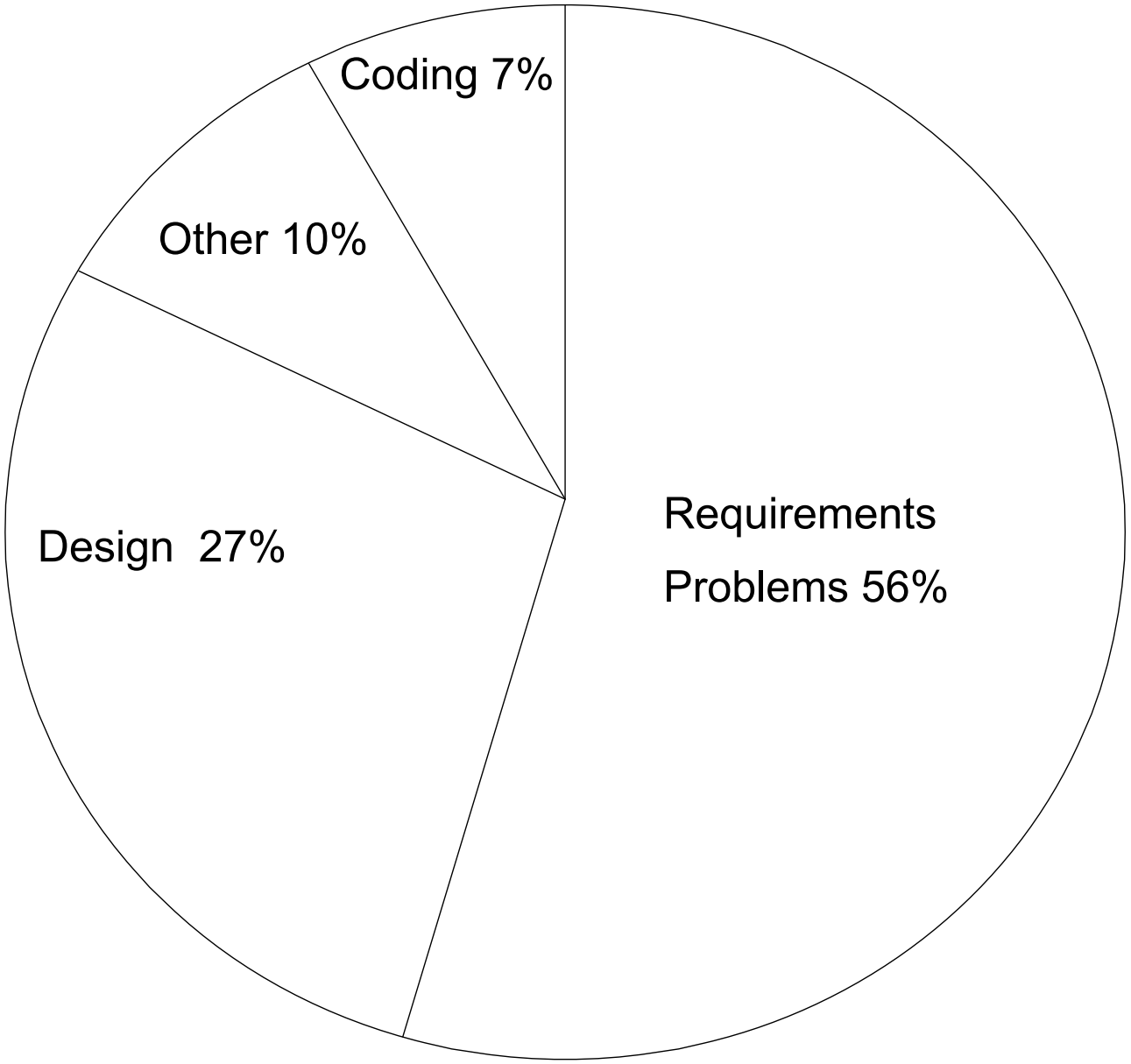
Read the following:

- Appendix A: Where to Find Requirements Information. (A large and important section).
- Appendix B: Analyst Wanted, Programmers Need Not Apply.

3.2 Motivation for Analysis

[Demarco 82] (and others), show the source of errors/problems with large software systems are distributed as follows:

Sources of S/W Dev Problems:

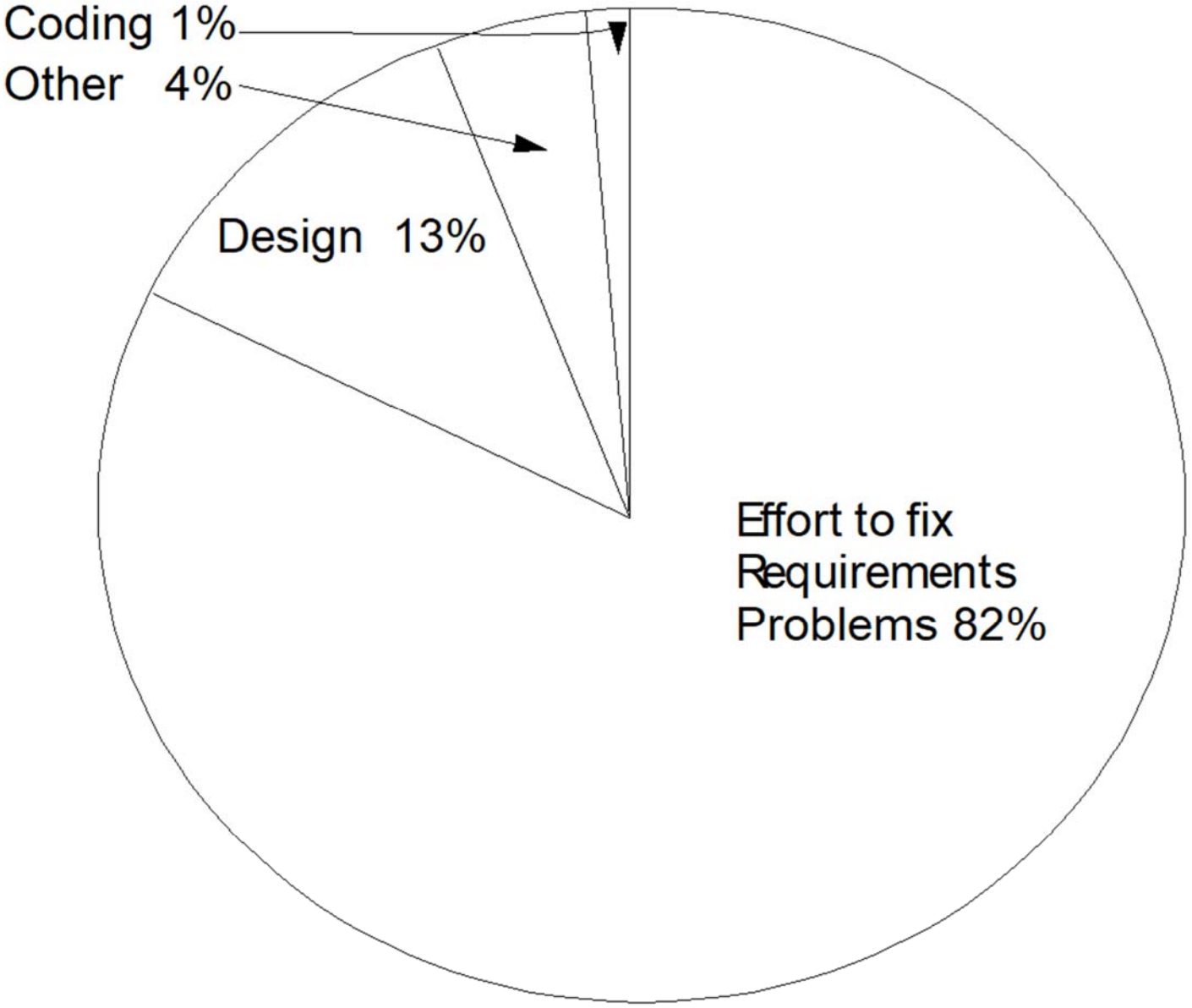


Coding errors are usually easy to fix once we find them.

However, design errors resulting from requirements errors may require a major re-structuring effort.

[Demarco 82] also found the distribution of '**effort**', as measured by labor (**person-hours**), required to fix the nature of the system was:

Effort Needed to Fix Various Sources of Problems



3.3 Don't Expect Things to be Familiar

It's the analyst's job to write a requirements specification that will serve as a single definitive source of *detailed briefing material* for the dozens of designers and programmers.

- Otherwise many programmers drive drive the customer crazy!

Even if the system is a familiar like payroll, there's likely local specializations needing gathering.

E.g. Analysts, designers and programmers are likely not familiar with job classifications and pay levels schemes for an airline.

- How many levels of pilots licences are there?
- Are they classified as 1, 2, 3, or A, B, C, or I, II, III, or “Commercial”, “Senior Commercial”, and “Air Transport”?
- Is the payroll time logged by the pilot measured in 10ths of hours, whole hours, or days on the job?
- What do acronyms such as P.I.C. describing a sub-class of flying hours mean, and what effect do they have on pay?

- Is payroll time for ticket agents, or mechanics measured in different units?
- Is overtime for different jobs handled differently?
- Do staff get paid in the local currency if they are stationed in foreign countries?

This list of questions illustrating your lack of understanding of the airline's payroll systems could go on for pages!

Never assume you understand the system to be designed.

- **It is invariably far more unfamiliar in terminology and intricate in operation than you expect.**

Don’t consider even beginning design, let alone programming, until you have at least started (if iterative development) or completed (if waterfall) the requirement gathering and documenting.

re-engineering

When ‘re-engineer’ a ‘legacy’ system, must:

- Determining the current functioning of the existing information system.
- And how a newer one should function.
- And how the existing data will be converted over for use by a new system!

Also, sometimes you are trying to research and postulate ways that an automated system can *improve* the functioning of a business or institution, not just automate the existing method.

When starting the analysis of an unfamiliar subject area, read:

- Industry and government reference documents,
- Magazines and books on the subject in general (so you can even ask the right questions and not look dumb).

E.g.

- Read a pilot's magazine to find out what payroll issues are being discussed.
- Consult a professional body like the Canadian Air Line Pilot's Association to see if they have any public info they can send you.
- Find a payroll accounting book.
- Ask one of the accounting associations for a reference, or whether they have a library.

3.4 The Difficulties of Analysis

Multiple reasons for pervasiveness of requirements problems:

- The subject area may be *completely unfamiliar*!
  - What the users assume from day-to-day working knowledge ("*Oh, in that special case we* …), the analysts and programmers must find out.
  - Terminology is completely foreign.
    - Do you know the difference between an 'altitude' and a 'flight level'?
- The customers themselves *may not know what they need/want*!

- You might just be told “something must be done to make our shipping department work more smoothly and track items better”.
- Or if the customers ‘involved’ in the specification process are managers, they may not be experts in that field of the business (they are just general managers).
- Or if managers are users, they may understand only their department.

Users may not know what management wants

- The customers may *not be sophisticated enough to envision, in full detail*, all that will need to be handled by the system. E.g.
  - the details of the data attributes and relations,
  - the details of the operations needed, and
  - any data or operation 'exception handling' needed.
- Busy customer managers may *not have the time* to specify their needs in detail.
  - This month's big sale or big disaster needing handling is more important.
  - May express just short-sighted, rather than corporate-wide, or long term, goals for the system.

The result is either a non-existent, incomplete, short-sighted, or inconsistent specification.

There have been many failures in the analysis, design, implementation, and integration of computer systems.

- 15% of all software projects are cancelled.
  - Usually only after spending on 75% of their budget!
- If this were true of office tower or corporate factory construction projects, there would be an uproar!

Note: In Cmpt 370 (93-2) there were 3 students who just returned from coop placements in a huge project (1500 database tables, 100 personnel).

It was cancelled after spending $30,000,000. These students were kind of in shock, still recovering from the magnitude of it all.

3.5 The Role of the Analyst

The analyst plays the same role as the architect in building construction. She:

- Gathers requirements,
- drafts a description of how a building will look, and
- passes that on in a well-documented manner to a civil engineer to do the structural design.

There are occasions when the analyst works for the customer developing detailed requirements that are issued to developers as part of a Request for Proposal (RFP).

Memorize meaning of RFP

Or she may work on behalf of the software development contractor to enhance a brief RFP into a full-fledged specification.

Sometimes on a big project a whole team of analysts are needed just to do the analysis, leaving the designers to lay out the structural plans.

Other times, on small projects, the analyst can, if properly trained and educated, do the design him or herself.

Now, you can't just say to a building architect "build me a new and better factory".

- An architect would immediately say "tell me more detail,
- or tell me where I can get more detail".

She will become a *scout* and go on a requirements-finding *mission* to:

- Examine any current factory that the company has (e.g. lay-out of production lines).
- Refer to operations manuals that describe the workings of the factory, and how it interfaces to the product development, shipping, and marketing departments.
- Research, or even conduct measurements of, volume of storage needed, rate of production of each product, percentage of product needing shipment via truck vs. rail vs. air, etc.).

- Interview the management and workers about the problems and advantages of the current plant.
- Analyze any plans for future expansion that the company has (e.g. different products, higher productions rates, or space needed for different production equipment expected to be installed).
- Investigate proposed sites for the new factory (e.g. how much area, is the land on a slope, which side has road access for the loading bay, is it adjacent to a railway track?).

Only after doing the above would the architect begin to consider the design of the new factory.

And even before the design was formally drawn, she would show preliminary sketches to senior and factory management. Such a review checks:

a) Whether her understanding of the problem is correct.
b) Gives management a chance to notice something missing (e.g. no elevated loading dock, or no elevator for handicapped people).
c) Provides her an opportunity to ask further questions she has thought of since she started her scouting and done her sketches.

Additionally and *importantly*, the analyst acts as a single interface who asks all the questions **once**!

And documents them properly for referral by the designers and programmers whenever they need information about the system to be develop. Since up to 100 designers and programmers may subsequently be working on a project, they would otherwise all be regularly phoning up the customer and *asking a tremendous number of redundant questions*. This would drive the customer crazy.

So: ...

- **Look** at the present,
- **Gather** requirements facts and measures,
- **Create a model** of what the new system features should be, and
- **Communicate it (efficiently via diagrams) to both customer and designers for review**, and subsequent implementation.
- **Write a coherent and definitive description** to provide both:
  - an interface between customers and designers/ programmers, and
  - which may form the basis of the dev contract.

Finally, realize that the analyst may conclude the business is organized and run the wrong way!

- It could be certain departments wrongly have write authority on certain data, or choose new part numbers for products which conflict.
- Or it could be she recommends changing to a system where all parts are bar-code stamped for easy tracking.

In essence, she may suggest ‘re-engineering’ the business, not just the computer system.

3.6 Keep These Things In Mind

While gathering your analysis information, always keep in your mind five things:

- **Watch out for the following anomalies in nomenclature**:
  - **Synonyms**: It is not uncommon for the same object or data attribute to have two different names in two different departments. In the inventory system, a part might have a 'cost', while in the accounting department this exact same attribute is called 'wholesale price'.
  - **Homonyms**: It is not uncommon for the same attribute name to have two different meanings. For instance, 'price' in the marketing department may mean retail selling price, while in the accounting department it may mean wholesale price.
  - **Inconsistencies**: E.g. data formats 24/04/22 vs. 04/22/24 in the US. Rather than a data string, you

many want to store month, day, and year separately so you can display in either country.

- Generally you are trying to determine the **business rules** used in the operation and record storage of the company/institution.
  - It's your job to extract, and communicate definitively (via documentation) what the rules of the operation are. These rules may affect the structure of the data, or the control statements needed (e.g. IF pilot_age >= 50 THEN ...).
  - Sometimes these rules are explicit (e.g. income tax rules for payroll), other times they are subjective (e.g. when a large order, manager must approve the client for credit if the client doesn't want to pay until the end of the month).

- Also, as in the case of income tax rules, you may need to suggest an analysis model that is flexible enough to change every year.
  - That means the income tax rules should be stored in some rules file (rather than hard coded) so they can just be updated (i.e. without changing the design, code, doing a re-build, and shipping a new version of the program to 1000 different accountants in Canada).

- **If the project is large**, you may have to assign several analysts for different subsystems. Some areas that have been suggested in [Coad91] and in [Shlaer92] are:
  - User Interface (UI) Sub-System
  - Data Base Management System (DBMS)
  - Data Communications Sub-System
  - Application Task Management Sub-System
  - Operating System
  - Process IO Sub-System (for automated factories)
  - Trend Recording Sub-System (for data acquisition applications)
  - Alarms Sub-System (for real time systems)

- Always take copies, pictures, or notes of things you encounter.
- And **draw pictures** of the relationships (input/output, data, time, or control relationships) you encounter.
  - Show the diagrams to the people who are giving you the information. Ask if they are correct.
  - *And, ask* **them** *to draw or correct diagrams.*

- Also remember that for each attribute you should note the:
  - **Name** (and any synonyms)
  - **Description** (purpose/use, and possible type)
  - **Range** (size of numbers or length of strings)
  - **Units** (miles per hour or kilometers per hour, radians or degrees, Fahrenheit or Celsius)
  - **Accuracy** calculations are made to, and variables stored to (e.g. pilot's flying hours to 0.1 hours)
  - **Precision** numbers are displayed/printed with (e.g. total hours flown by all pilots in the airline all month printed to nearest full hour)

Precision: to how many decimal places the number is displayed to

3.7 Use Cases

[Jacobson92] introduced the emphasis and importance of “use cases” for software design.

Though there are ‘use case’ diagrams in UML, they are not very useful. We may study them later, as they can be used in customer meetings.

More important is an inventory of *written* use cases.

A use case is of the form:

- Title
- Actor “As an accounting user…”
- Preconditions
- Steps of the interaction. The latter steps are often paired prompts from software system and user responses, until the use case is finished.
- Normal results
- Other variants.

Do I just note the case where a failure occurs as a failure, or would I list what led to it?

E.g.

- Name – “Enter pilot information”
- Pre-conditions - Pilot database table created, and not full.
- Interaction Steps:
  1. Flying school employee clicks Create Pilot.
  2. System polls clearly for last name first, then first name.
  3. User enters last name and first name.
  4. System polls for pilot’s licence number (alpha numeric up to 10 characters).
  5. User enters pilot’s licence number.

6. System checks federal government database for validation of licence number.
7. System signals complete, or error.

- Result: The pilot entry is complete and visible.
- Notes and variants:
  - If error, go to step 4.
  - However, build into step 4 or 7 a method to exit if pilot licence is simply wrong.

Use cases *infect* the software development process:

- They need to be gathered and written during requirements gathering.
- Visual appearance of each needs to be written in to External Design/User Manual, before coding.
- Need to be architected for in design phase.
- Must be implemented in code.
- Form the test cases for the resultant product (possibly for acceptance by customer).

Architecture: deciding what tools you plan on using
Also include multithreading as well to show how the system functions

Thus super important!

3.8 Requirements Meetings

Meetings of all sizes (even just one-on-one) are used to gather requirements.

If doing the initial gathering of requirements, be sure to get customers/potential users to do most of the talking!

- **Ensure that one of talkative-natured analysts does NOT do most of the talking, writing, and drawing.**
  - I have personally experience this!

- It is important the information goes the other way! Always be:
  - **Asking them questions. Like:**
    - **what,**
    - **who,**
    - **how,**
    - **when,**
    - **why,**
    - **where?**
  - **Asking the customers/potential users to get up and write, or draw, on the whiteboard.**
    - **Firmly coax customers Pass/push a pen in their direction.**

Same is also useful when the meeting is to review a draft requirements spec.

Ask customer to draw, annotate, or list business process steps that are MORE correct.

3.9 Appendix A: Where to Find Requirements Information

3.9.1 Written Materials

Usually there exists several kinds of written material to examine for clues:

- There's likely an initial request for your participation in the development.
  - Usually someone in the customer's institution has written a request for improvement describing the problems with the current system. This description could be simply a paragraph in the Request For Proposal, or it could be quite a long description accompanied by diagrams.
- There could be operations manuals used to train new staff about how the company or department does things.

- If an automated system is already used, there may be an old requirement specification, or design documents that would contain invaluable information.
- Or, with an existing system there is likely a user manual describing how to operate that specific software application.
  - In addition, it may have descriptions of the data formats and structures used.

3.9.2 ***Interviews***

Usually on a big project using waterfall development, there is:

- an initial meeting,
- followed by further meetings with both management *and* operational staff, and
- finally review meetings.

The interviews allow you to ask people (or a room full of personnel) for their understanding of the system.

During interviews:

- Remember that the information transfer needs to be from them-to-you!
  - You must encourage and let them do most of the speaking by asking questions, and listen carefully to the answers.
- Take notes or even record the session.
- Pay close attention to understanding the terminology.
- Learn the relation or dependencies between data and operations.

- Shy people will not speak very much unless prompted.
  - If you notice certain people in a meeting are quiet, make an effort to obtain their ideas.
  - Be prepare to allow a little silence to see if people will pipe up and say something.

- Use diagrams and lists on a black (or white) board extensively.
  - Remember, a picture is worth a thousand words!
  - Don't just you draw what it is you are hearing to check for correctness. *Instead, encourage others to get up from their seats and draw/write*.
  - Often you will get valuable insights into structure when people draw arrows showing which objects in the system are related to which others.
  - You'll find out about which order operations take place in from lists or arrows!

3.9.3 Existing Forms

Often forms are used to interact in a formal way with individuals, or record information about incidents or changes that need processing.

Moreover, *forms are often very organized and complete*.

- Consider a student application to to SFU.
- Has whole sections to gather:
  - all the data about you,
  - all the information about your background (e.g. high school, past marks), and
  - all the information about what you *want* (what program, what semester you want to start, whether you want to live in residence).

Note this data is often grouped into boxes. This is a hint, as to:

- what the major objects are and
- what attributes are part of which object/record.

Forms are usually well structured, and in addition they often have formatting information showing the order in which information is later entered into the system (e.g. last name first, or year first “93/04/14”). Additionally, on forms where every character is entered into its own box, you can tell the:

- maximum length of the character string or numbers used by the institution, and
- the number of decimal places of accuracy, and
- sometimes the unit of measure!

In looking at some typical forms, you can also see boolean or enumerated variables. Take a look at

an SFU Course Add/Drop form. There are two enumerated variables, each with 3 possible values:

course_type = (credit, audit, challenge), and

semester = (regular, intersession, summer).

Do you know what an intersession is? If not, ask.

I saw an employment form recently for a limited-term employment appointment, and it had both appointment start date and payroll start date sections.

- What is the difference?
- How can a designer or programmer write a payroll system until this is explained to them?

These two dates also suggest two departments might be involved, each measuring the start date differently.

- So there is some interdepartmental stuff here that obviously needs understanding by the analyst.
- The analyst may even propose the departments change to use one date to make things more consistent, and thus reduce the effort and misunderstandings involved between the departments.

I saw a requisition recently that had a well-defined 5-digit requisition number (that's good to know).

But unfortunately, the value of the goods was to be charged to a 3-part charge code broken into fields labelled fund/account/centre.

- What are these,
- how many digits are they,
- or are they alphanumeric characters?

I looked at some forms used to send packages by courier. In particular, Federal Express has 3 forms:

- One for domestic shipment
- One for international commercial material
- One for international non-commercial material

This is a hint that there are 3 ‘subclasses’ of Fedex Waybills.

- These would ideal candidates to be either variant records, or object-oriented sub-classes.

I also noticed the forms allowed the weight of the shipment to be a number, which was interpreted as either pounds or kilograms depending on which of two boxes was ticked by the clerk.

You will find that most forms have fields for signatures, thus you can find out some routing and approval information.

Many forms are multi-copy documents, where writing makes three copies. At the bottom it often tells you about:

- who gets the white copy,
- who gets the pink copy, and
- who gets the yellow copy.

The analyst can see what *other* departments need read access to the record of the incident this form contains, *should the form ever become computerized*!

In summary, existing forms (and screens) are a veritable gold mine of useful information! Also remember to look at receipts given to customers.

3.9.4 Screens

Much of the previous subsection applies also to any existing screen layouts.

3.9.5 Visit the Existing System

You should actually visit the site and see the existing manual or computer system in operation.

- Watch users.
- Get them to show you all the screens.
- Note down any data formats you see for the first time..
- Obtain screen dumps and examples of printed reports. Remember that the manual you read above may be for a generic, say, payroll system, and is configurable for individual companies. How has this company configured the system?

Get staff to show you how they handle the paper:

- where is it created,
- who has to approve it, and
- what happens to it (e.g. filed, keyed-in, discarded)? In future, you may have to automate this with approvals being given by on-line digital signatures!

Ask staff what the exceptions to the rules are.

- What happens if a form or screen is filled out wrongly?
- What happens to international paychecks?
- Are these exceptions handled by a different flow (think about data flow diagrams and conditional branches)?
- Do exceptions have to be approved by more senior managers?

Note the characteristics of any existing computers, peripherals, and support software the corporation might have.

- They may have a large investment ($ and expertise) in certain equipment and software.
  - They'll be hesitant to change or upgrade.
- Or, you may have to convince them their present systems are inadequate.
  - But you can't judge this unless you record the details of their present systems.

Take note of:

- CPU and speed
- RAM and disk memory
- operating system and version
- Windowing system and version.
  - Most Linux'es use X Windows, Ubuntu Linux uses Wayland windowing system.
- Database management system (DBMS) and version
- Network software and version

- Network data communications speed, and how heavily (50% of capacity?) the network is currently used by other continuing distributed applications.
  - If you've taken Cmpt 371, you'll know that a 10 Megabit per second Ethernet is not capable of 10 Mbps capacity.
- The printer type and print rate.

3.9.6 Measurements

Sometimes measurements of space, rate, and type of each transaction are available from the customer.

Other times the customer doesn't actually have an idea of the magnitude or relative magnitude of things.

Measurements of space are needed not just for specifying future worst case (physical or memory) storage needs. Large storage requires sophisticated (and expensive to program) structuring and algorithms to provide reasonably fast access.

- Give hint whether unsorted data records with linear search is ok, vs. binary search on data that must be pre-sorted, vs. balanced trees.
  - There is no use the programmer doing fancy, doubly-linked trees if they are not needed.

Regarding rates, have you ever wondered how many reservations a day (or per minute) that Air Canada has to handle?

- Do you think a designer would do a different design if the answer was 240 per day for a small airline, versus 24,000 per day for a larger airline.
  - The first could be done on a PC, the latter would probably need software designed for high transaction processing rates.
  - Now most designers could guess what Air Canada's reservation rate is, but some applications are just so unusual that this information is impossible for the designer to guess. It is the analyst's job to gather measurements of these rate statistics.

You may also need to measure:

- Whether certain times of the day Air Canada's reservation rate is much higher than 1000 per hour. i.e. what is the peak rate that must be handled, or the standard deviation of the hourly values.
- What percentage of these are for discount fares.
- What percentage are from overseas offices and other airline systems which can interact with Air Canada's.

- Are most accesses just inserts and deletes with only a few reads, or are inserts and deletes rare and most accesses reads.
  - This information will help the designers and programmers choose between sorted lists and trees.
- What percentage of the reads are random (for enquiries), versus sequential for printing out all the objects with certain characteristics one after the other.

In summary, most measurements are of either:

- Volume storage space currently needed, and needed for the worst case expansion in the future.

- Or rates (e.g. shipments, reservations, message arrivals), both
  - absolute rates (characteristics measured per unit of time), and
  - relative rates of various sub-classes of events (e.g. 7% of registrations are for graduate students).

  This data used by the analyst to construct a good Object Relationship Diagrams (ORDs). And implementers to choose 2NF vs. 3NF database layout, object relationship sparseness, whether to use subclassing.

3.10 Appendix B: "Analyst Wanted, Programmers Need Not Apply"

[Kendall 87] above pointed out analysts need a different skill set vs. programmers.

Programmers notoriously think totally in terms of 'how' the system should be implemented.

- E.g. "Would I use a variant record, or just a regular record and leave some fields blank?"

Inappropriately concentrating on the 'how' aspect has two negative effects:

- It clogs your mind with extra considerations.
  - Huge information systems have requirements that compose several binders full of analysis.
  - Some coop students had worked on a project which had 1500 relational tables/entities in it!
  - Mistakes are typically:
    - oversight,
    - inconsistency, or
    - mis-understanding of the interface/relationship between objects.

  - Humans mistakes happen more frequently as you approach/exceed 7 objects or relationships in a diagram.

- Concentrating on the 'how' can also bias you against features which you may (possibly wrongly) think would be hard to implement.

To propose the best possible system, the analyst should be a 'lateral thinker', *unifying the best parts* of many somewhat wild alternate solutions independent of cost, employment layoffs, or the 'old ways of doing things'.

A big project analyst might never talk to a programmer, nor need to know how to program.

Systems analysts analyze how *any* system works and suggest changes or improvements.

- A "time and movement" analyst may go onto an aircraft carrier to figure out getting airplanes, fuel, and bombs up onto the deck quickly.
  - Should all 4 elevators be devoted at particular times to one type of cargo.
  - Or should one only do the fuel and weapons transfers to the deck?
  - And what are the trade-offs of speed, versus safety against explosion?

So, the analyst typically has the traits of a:

- curious scout,
- lateral thinker,
- solution constructionist, and
- pretend user.

Nonetheless, designs often follow the solutions closely. So an analyst and the designer often have considerable common understanding.

But it's the analyst who tells/illustrates to the designer what data must be stored and manipulated.

If the designer can:

- envision the kinds of problems users have,
- has the determination and people skills to ferret out requirements from sometimes reluctant sources,
- is a good lateral thinker, and
- can propose a better organization and control authority for data and operations,

then he can also take on the role of an analyst.

Similarly, a good analyst, if she is versed in data normalization, user interface mechanisms, and distributed system design, may be capable of

subsequently evolving her analysis models into a design.

3.11 References

[Coad91] “Object-Oriented Analysis” 2nd ed., by Peter Coad and Ed Yourdon, Prentice-Hall, 1991.

[Demarco 82] “Software Systems Development”, Tom Demarco, Prentice-Hall, New York, 1982.

[Jacobson92] “Object-Oriented Software Engineering: A Use Case Driven Approach”, Ivar Jacobson et al, Addison-Wesley, 1992.

[Kendall87] “Introduction to Systems Analysis and Design” by Penny Kendall, Wm. C. Brown Publishers, 1987.

[Kendall92] “Systems Analysis and Design” 2nd ed., by Ken Kendall and Julie Kendall, Prentice-Hall, 1992.

[Shlaer 92] “Object Lifecycles: Modeling the World in States”, Sally Shlaer and Stephen Mellor, Prentice-Hall, 1992.