A Systems Analysis Framework

About this document: this document reflects the major ideas of systems analysis, and is not intended to develop these ideas fully. It is to stimulate discussion about systems that do not work as planned and for planning new systems that will work.

1. Foundations: the systems analysis environment

Systems analysis grew as a field from biologist Ludwig von Bertalanffy's foundational text, *General systems theory: foundations, development, applications* and developed in the 1950s from the concepts in ontology, philosophy of science and applied to many fields: from technical issues (engineering, technology) to human processes (management, psychology) and units larger than individuals, such as whole societies and work processes. As a result systems analysis is the highly structured, formal study of any system, that is a set of regularized actions or behaviors that when taken together form a whole for the effective and efficient achievement of a goal or set of goals.

The development of systems analysis as a field and its application, especially in management and in technology, helps one to decompose a system, determine how, why, and where it is not functioning, and then, through an analytic technique called process decomposition, determine how to create inter-related functions, or modules, and reconstruct the system to be effective again.

Any set of behaviors (human or computational) can be cast as a system. Most of the time, people think of systems as being applied to a technical or management concern within an organization. The personnel, mission, needs, finances, technology, and specific goals of that organization form the parameters of a project: from the entire organization down to a small task. Generally, the work of systems analysis is not considered unless the project has significant financial, human, data impacts or materially affects an organization's ability to achieve its mission.

More than anything else, systems analysis is a creative activity. The analyst must frame the whole such that it can be analyzed. The analysis must work with all levels of an organization, data, work processes, gather information through a host of sources, and somehow shape these into a cogent whole. The rest of this document describes in broad strokes how to frame any system, the questions to ask of the personnel and equipment, and how to harness these to create the documents necessary for implementing the new plan.

Managing the case

There are various models of how to modularize any system to guide the analysis through the sequence of steps necessary to ensure all, or most, aspects have been considered. Here is the most commonly used "waterfall model":

1. Project identification and selection

    Select projects to be performed [incremental commitment]

    2. Project initiation and planning

        3. Analysis

            4. Design

5. Implementation
6. Maintenance

Each part of the waterfall model stands as an independent activity and leads to the next activity. Note, however, that there is typically a feedback loop between each activity; when about 90% of the one activity is completed the analyst progresses to the next activity and here sees some of the consequences of decisions made in the earlier activity. The analyst adjusts the previous activity and continues forward. No analysis is ever perfect: the analyst must learn to balance pros-and-cons and to live, and to convince others to live, comfortably at times with uncertainty and compromise.

The analysis *decomposes* an activity, creates a semi-closed system or *module*, determines the impacts of the *coupling* of the modules, and the overall *cohesion* or differentiation of work of one module from another of the project.

Decomposition: breaking a system into smaller, manageable, and understandable subsystems; to facilitate the focusing of attention on the one area, without affecting other parts; permit different modules to be built independently of other modules by other people.

Modularity: the division of systems into chunks of relatively uniform size.

Coupling is the amount of dependency of the modules; if one subsystem fails, the other subsystems should continue functioning.

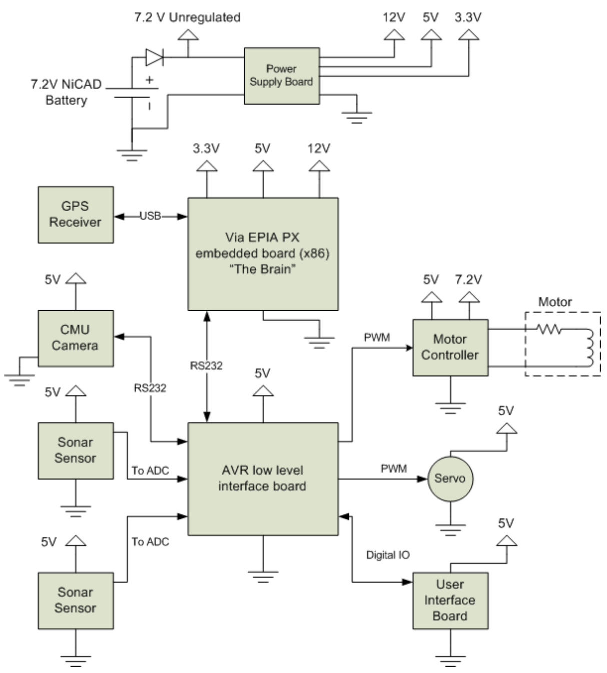
Cohesion is the extent to which a subsystem is successful at performing its function, independent of others. Ultimately all modules are joined into the new system.

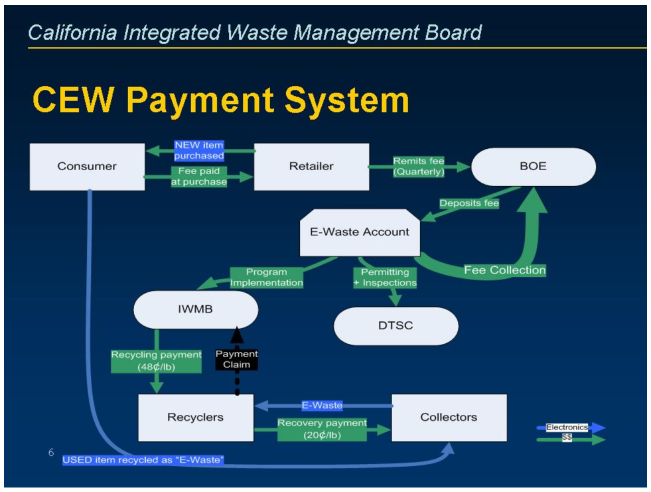
Once a project has been approved, the analyst initiates* the project, plans how she or he will attack the problem*, and will create **logical** descriptions of the system (the virtual system, as if it were really built) and the **physical** system description, the material depiction of the system in order to build it.

Organizational knowledge: The analysis must be aware of the organization from the abstract level (e.g., the mission statement), to the data and technical level, the work function level, and, especially, the human level, communicating successfully with different levels of work. It is useful to frame any organization by roles: *system owner*, *system manager*, *system designer*, *system builder*, and *system user*. Each of these groups has their own needs, expectations, and method of evaluation. Not all of these are reasonable or applicable so it is important for the analyst (a) to extract facts, (b) tie those facts to the project, (c) using certain forms, have system managers or owners who are authorized to act for the organization to *sign approval forms*, (d) and explain through oral presentations, text documents, and graphic displays the module or entire project in a way the target group will understand. Part of the formality of systems analysis is the creation of increasingly detailed content diagrams, flowcharts, data flows and work flows, which are later converted to highly iconic, simple diagram called User System Diagrams (USD).

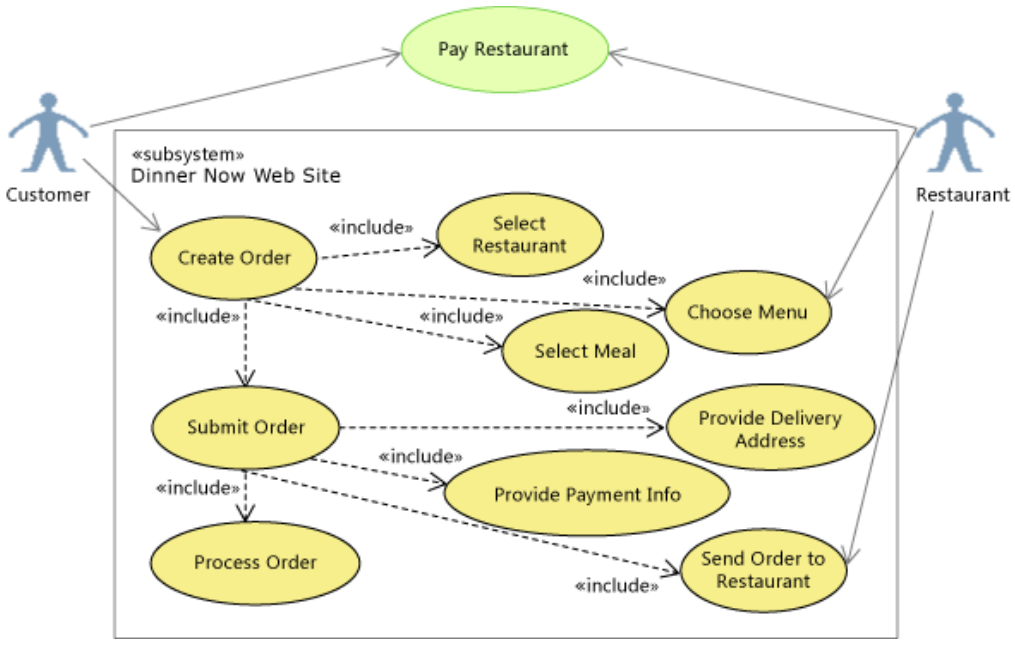
[Sample USD. Notice how the graphic representation of the idea differs by audience.]

USD from IEEE





Sources: (l) Source: http://ieee.ucsd.edu/projects/robomagellan/?y=2009&t=1; (r) http://www.ecyclingresource.org/UserDocuments/CA%20Payment%20System%20Diagram.jpg



http://msdn.microsoft.com/en-us/library/ff183189%28v=vs.100%29.aspx

The above example emphasizes the UML approach. Check out the website.

2. Skills

The analyst needs to have *technical* and *management* skills.

- *Technical skills:*
  - Basic computer technology; Windows, Macintosh, Unix
  - Networking: client/server architecture (the Internet is one example) and local area networking
  - Data modeling: *entity-relationship modeling* for structuring data for relational database management systems (e.g., SQL); *XML* and domain-specific applications of XML (e.g., administrative, descriptive, and rights management metadata); professional groups' and XML (e.g., VRA Core, AAT, ULAN, DC); and *full-text information retrieval* (of which Google is the most famous example)
  - ISO and ANSI standards (e.g., ISO639-2 for language codes; Unicode UTF-8, UTF-16)
  - Familiarity with dynamic data generation for the Internet (especially PHP, Java, ASP, JSP, JavaScript, Ajax)*[1]
- *Management skills*:
  - Resource management: human, budget, project, evaluation, securing resources from outside vendors, contracts
  - Project management: estimating and management of a project, time-sensitive milestones, reporting and documentation
  - Risk management: assessing the risk (disgruntled users, disgruntled staff or management, loss of income, loss of data, etc).
  - Interpersonal skills:
    - communicating with all user groups
    - interviewing, listening, questionnaires and other data-gathering instruments
    - oral presentations to members in the work group or team, to other units, to management, to end-users
  - Legal concerns

[1] The analyst may be just the analyst; usually the person of the analyst does both analysis and systems development ("programmer/analyst"), in which case s/he must know everything in the list.

3. Projects and Deliverables

Many organizations have an in-house technical staff. The system owners, usually executive management, commit financial and human resources only to investigate the feasibility of a larger project. Here the system owner should generate the first **deliverable**, a **System Service Request**, see Appendix for an example. This document and all subsequent ones, along with all work materials, should be stored in a **project workbook**. The project workbook becomes the official record of the project. It should contain the following:

1. Project overview
2. Initiation plan and SSR
3. Project scope and risks
4. Management procedures
5. Data descriptions
6. Process descriptions
7. Team correspondence
8. Statement of work
9. Project schedule

Planning the project:

The analyst should prepare the *project scope, alternatives, and feasibility* statement and include

- what problem or opportunity does the project address
- what are the quantifiable results to be achieved
- what needs to be done
- how will success be measured
- how will we know when we are finished?

Preliminary schedule of work

With a tentative understanding of the project, divide the project into manageable tasks: some tasks may be performed in parallel with others; some must wait until another is completed. This is where **Gantt** charts are used to prepare *preliminary schedules* of work.

Communications plan

Determine the sole communications access point between the project team and the owners/managers/users. Controlling discussions about the project reduces false expectations and requires managers/owners to sign off at certain stages.

Statement of work

If the feasibility plan is accepted or management has decided a project should go forward with a feasibility plan, the analyst in response to the SRS must develop a statement of work. This is very much the same as hiring a painter: the painter says what he will paint, perhaps the brand of paint, the start date, the last day, clean-up, and so on. The statement of work should be viewed as a contract: do not permit the managers/owners to adjust the project after they have signed off. This is called "creeping scope" - the project grows too big or grows contrary to your analysis - and is a guarantee of failure. This leads to the next deliverable, the **Baseline Project Plan**.

Executing the project

Once the project has been approved, the analyst may keep track of when each part is completed or the percent completed at certain times, called milestones. In short

- Monitor the project progress against the Baseline Project Plan
- Manage changes to the plan through supplemental SSRs
- Maintain the project workbook
- Communicate the project status (officially with presentations or memoranda)

Completing the project

When the project draws to a close, people who participated in the project should be recognized (a reception, official party, swag), and verify through discussions with managers and owners that the project is successful. The project workbook should record this and be finalized. The project workbook should hold all documents, approvals, content diagrams, flow charts, data flow diagrams, computer code (on CD), technical and end-user documentation.

4. Tools

There are a variety of software tools to help manage projects. One class of them is called CASE (computer-aided software engineering) tools. The purpose of using such software is to improve the analysis, increase the speed of development, control versions, enforce standards, communicate project documentation to others, promote the reuse of existing software modules and documentation, and simply program maintenance. There are many project management tools; most database management systems include built-in tools (Oracle's CASE, Java doc).

Later you will see that the analyst includes requirements determination (gather data about what is wrong, what should be the case, etc.), and requirements structuring (process decomposition, context diagrams, flowcharting, etc.). To complete flowcharting and as part of the bridge between logical and physical design, the analyst needs the skill of logic modeling, called **Structured English** or **pseudocoding**. To complete these tasks, one should know modeling with Structured English, decision tables or decision trees, **entity relationship modeling** (ER modeling, for relational databases) and **XML schema**.

5. Project in brief

In this section, the major activities and deliverables are presented in outline.

Activity / Specific task	*Deliverable*
1. Project identification and selection	

Identify potential projects	
Classify and rank projects	
Select projects to be performed	
2. Project initiation and planning	
Project initiation	System Service Request
	Build-or-Buy analysis (future value of money;
	Net present value, Return on investment, or Break even analysis
	System costs document
	Technical feasibility assessment
Project planning	Baseline Project Plan
	Statement of Work
	Project Scope
	Walkthrough Review Form & Action List
3. Analysis	
Requirements Determination	
Requirements Structuring	
Process modeling	Context Data Flow Diagram
	Process modeling (DFDs)
	Logical flowcharting
	ER diagrams with attributes
	UML of activities and object models
Alternative Generation & Selection	
4. Design	Logical data model (relational) and physical file and database design [Files and databases]
	Forms and Reports
	GUI Dialogues and Interfaces
	System and Program Structure
	Distributed Systems
5. Implementation	Database and file definitions (DBMS specific code)
	Coding
	Code
	Program documentation
	Testing
	Test scenarios (test plans) and test data
	Results of program & system testing
	Installation
	User guides
	User training
	Installation & conversion plan
	Software & hardware installation schedule
	Data conversion plan
	Site and facility remodeling plan

Documentation
Training
Support

6. Maintenance

Once the documents in #2 (Project initiation and planning) are completed, the analyst and the appropriate contact person (the client) should perform a **walkthrough**. This is a meeting between stakeholders and should review the BPP and collect any last minute opinions. The authorized parties should sign this document.

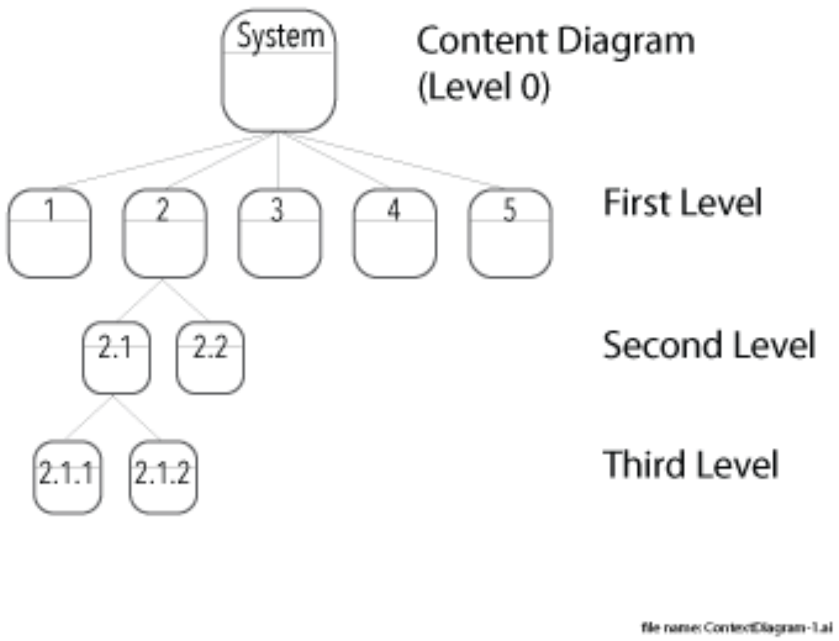
Analysis The first step in analysis is gathering data. Typically people sense there's a problem - something isn't working well or people want something that might be useful to help do the job - and that discontent is evidenced in email, documents, and by having discussions with staff. The traditional approach is to interview people, conduct surveys, observe people on their jobs, study business documents. [In this document, we do not go through all the steps of interviews and analysis techniques.]

A good technique for determining real needs (not the ones the end-users *think* they need) is to bring together owners, managers, users, and the analysis and design staff to a *Joint Application Design* (JAD) session. During a JAD session, the coordinator (usually the analysis, but an independent coordinator may be politically better) guides this group in describing the actual work flow, or in expressing their idea of the best interface, or how they would prefer to accomplish some task. The result of this session should be a plan of action that the coordinator then goes through with the group to confirm. For example, if the task is something such as how to input records, have the inputters describe how they do the job now - sketch it out - and the managers are available to answer any specific rules questions. Then cooperatively sketch out the ideal approach.

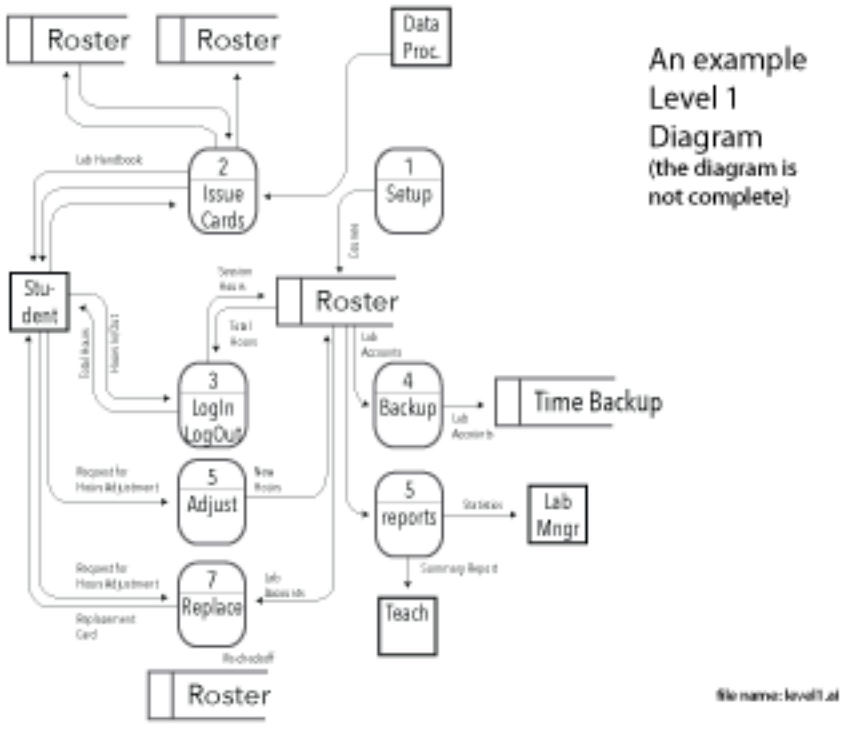
Alternatively or at a subsequent JAD session, create a prototype of the workflow or an interface design and have the participants use the system as if they were completing the task. Talking aloud during the prototype session reveals how people feel: look for that "Oh, boy!" or "hmmm" or whatever oral and non-communicative behaviors to understand user attitudes.

RAD: *rapid application development*. Usually after the first statement of work, it is useful for interface designers to provide to end-users something - a prototype input screen - that lets them get a feeling for the product. With very little investment in time, the end-users feel like their needs are primary and future interactions with them will be improved.

Context diagrams

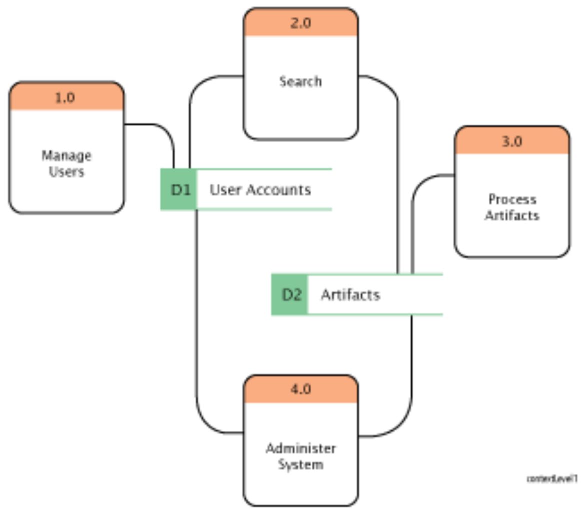


Recall that *anything* can be modeled as a system, including and here especially, work flows. The analyst now divides any system into two functions: the (human) work processes and the data processes. To modularize any large system, the analyst should break down the *concept* of the system (e.g., a library digitalization project) into gross level functions (such as "cataloguing", "filming", and whatever else is appropriate) and then show how each function communicates to the other. This high level abstraction is called the **context level diagram**. Here is an example:



Each major module (e.g., "create records") is assigned a number (e.g., 1.0 create records).

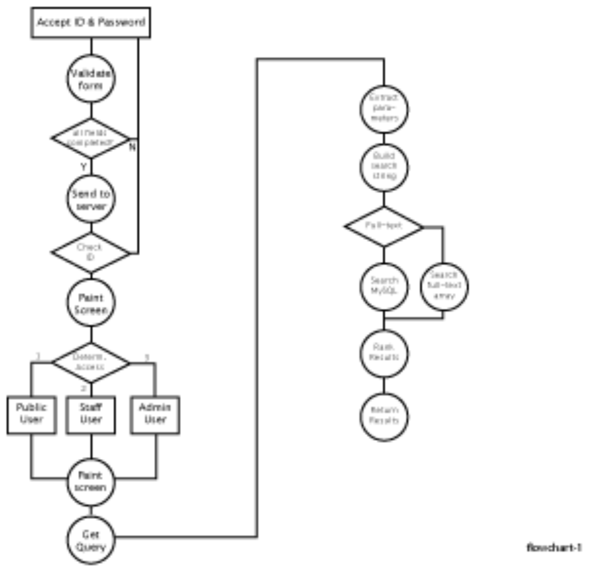
Then the process is decomposed even further in a **level-n diagram**. If "1.0 create records" is the major process, each of the gross level steps that constitute that process are articulated, e.g., "1.1 select item for cataloguing", "1.2 prepare cataloguing sheet", and so on. This is the **Level-1 diagram**; each processes is decomposed (Level-2, Level-3, etc.) and so on. This is called **work-flow analysis**.



Once the work-flow analysis is completed, have the people who perform that function work through the diagram.

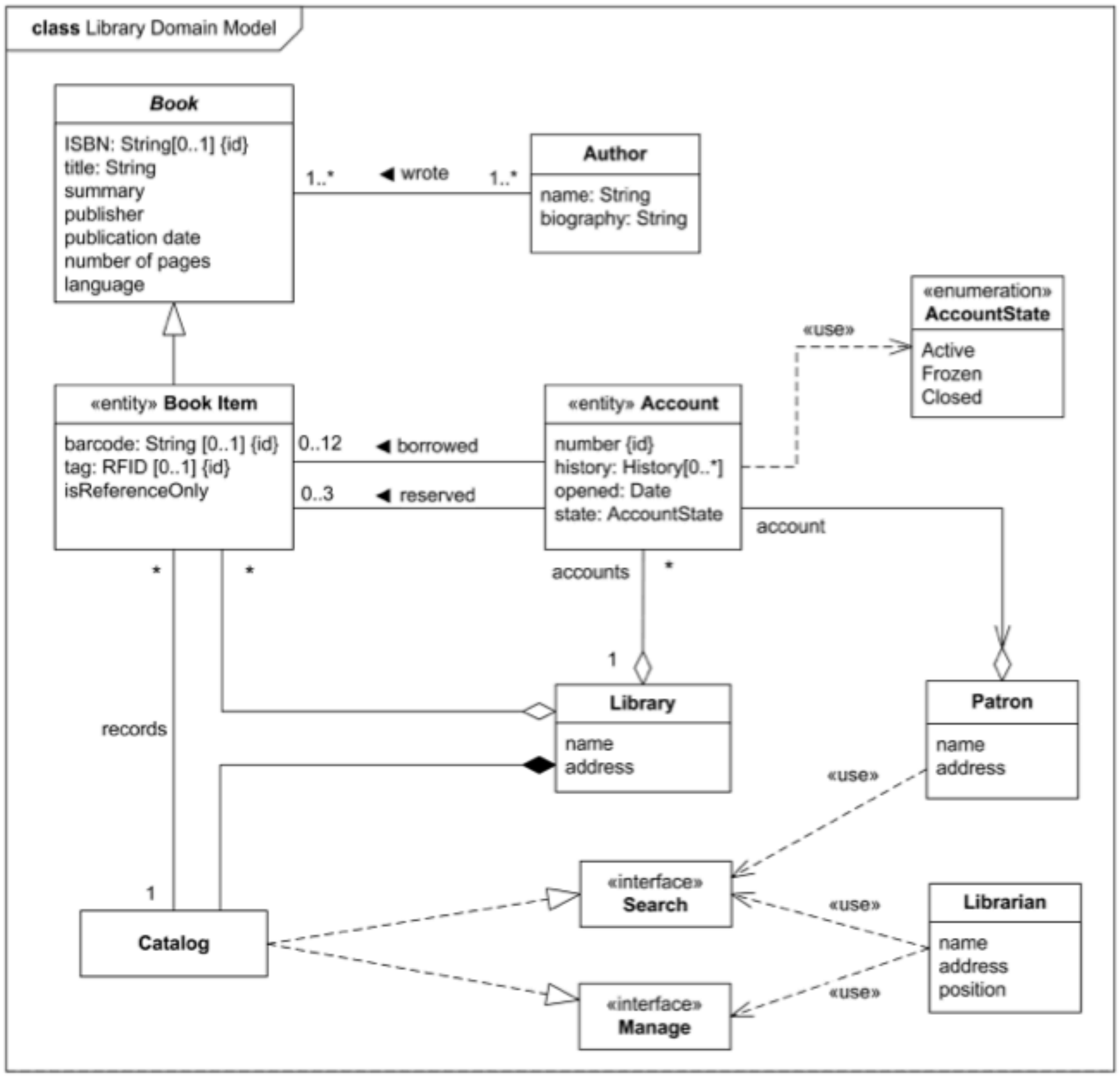
The next step is to do a **data flow diagram**, demonstrating how the *data themselves are transformed*. In the level-n diagrams there are multiple processes. The analyst should review these processes *from the data perspective*: from one process to the next the analyst should state how the data are transformed. For example, one process may "collect data". Another process (e.g., "calculate salary") might calculate some new value from the "collect data" process.

These diagrams use four symbols: "process", "data store", "source/sink", and "data flow." The process is just that: some activity, represented by a verb. The "data source" could be a relational database or file or another process. The "source/sink" refers to where data leaves a process (the sink) or where data enters a process (the source). Finally the "data flow" is an arrow pointing to indicate the direction that data flows from one process to another. Note: do not use "process" as a place to store data.



After decomposing a work process (usually no more than 2 levels), the analyst considers the *decision-making involved* in a given sub-process. This decision phase results in **flowcharts**. Although flowcharts look similar to data flow diagrams, they are not the same! Note, too, that DFDs and flowcharting are iterative processes: they are the heart of analysis and can be time consuming, frustrating, but ultimately the very purpose of analysis: removing the logical inconsistencies that cause the older system to fail.

Universal Modeling Language is used, too, especially for Activity analysis and class modeling. Here's an example of a library system, expressed in UML.



Class Diagram Example - Library Domain Model

Graphic representations of the work, data, and process decompositions help to identify logical inconsistencies and to identify processes we might have missed. Working out the final model requires many iterations!

Implementation plan includes product and work-site specific details. They guide the client in integrating new software and work processes. The best model is a phased integration where parts of the new workflow and computer systems are integrated and tested. Usually includes training.

Maintenance plan must include a model of evaluating the effectiveness of the new system, technical maintenance (updates to software, etc.), and follow-up training (usually 6 mo after).