Business Analysis

Table of Contents

BTC-202 Business Analysis manual v5.02.docx last saved October 12, 2020

Intro to Business Analysis

Purpose

Value

Value driven business projects

A project proposal should be estimated on value. Not only just a return on investment but understanding what the priority is for the enterprise. Value for internal projects can be seen as basically two parts but more can be added if justified:

- The number of people in the company that will use the project in their processes
- The visibility of that project in the organizational chart of the business so that if that project fails, the level of management involved indicates the severity ticket level.

Value for external projects is estimated the same way. There are the same two metrics involved:

- The number of people in the market that will find value in using the product/service.
- The amount of money that the market is willing to spend to get that product/service

The two measures are both based on **quantity of value**. The project then can look back to this value of opportunity and see how to organize the project pieces. For instance, business goals, the basic business driven goals that begin the requirements process, are based on what value the business gets. Requirements meetings should never ask the question "What do you want the project to do for you?" because it's not about the person. The better question is "What should the project do to produce the best value for the business?"

Then, the business can see that the customers have needs where they should adapt their product/service to achieve the best solution to a large market segment. The question for the strategists and enterprise analysts would be "What can I do to help our customers/employees solve their problems?"

Value then becomes the reason for scope analysis where resources are applied to gain the value in the right amounts. Available time and money become project constraints against which are scope is limited.

Why do analysis?

✓ **Understand scope**

✓ **Communicate scope**

Business analysis can be summarized into two major parts. The roles are not always clear and generally no one person is always responsible for them. Because of that an employee who is a business analyst may not be a part of a project but business analysis may be taking place in the project none the less.

Understanding and communication of a part of business scope are the two foci of the role of a business analyst. They are managed through the documents that the tasks create but to determine the quality of the tasks is another task.

The best questions you can honestly ask yourself to check on the quality of the tasks are

- Do I really understand the scope enough to talk about the way it impacts the business?
- Can someone read the documents that I wrote enough to be able to talk about the business impacts?

Stakeholders know about business and know little about the creating software. The stakeholder wants to take what they know in the world and do it better or faster with the help of computer programs. This part of the world that they know about is the problem domain.

The stakeholder wants certain conditions to be realized by the use of specialized software like the ability to know the status of orders and where shipments are. Software must be written to manipulate or manage the problem domain and the stakeholder must be able to describe it to the analyst so it can be written down and communicated.

Understanding

No matter how much communication you have during a project, it is still vital to be able to understand what the results should be. Reworked code after hours of discussion is frustrating for everyone and just having more meetings isn't going to solve it. The diagrams and documentation that you prepare should be focused on realizing meaning from the business side and translating that to a usable interface with common language so that when it comes time to use the new system, the users aren't stymied by odd buttons and menus.

Communication

The software engineer doesn't understand the problem domain unless it's logically spelled out and has all the descriptions that will allow the computer to process the details that need thinking about. Because the business person rarely thinks about all the details due to experience and natural ability, it's important to write down what it is that is being done so that communication can happen between the business type and the computer type.

Throughout the elicitation of the requirements and the reviews into the analysis phase of the project, it is important to have clear and concise documentation and diagrams that communicate what the software will be before the large expenditure of time and money happens. If you are going to kill a project, you want to kill it early and fast.

Extension of knowledge

By writing down the requirements, people can grasp the large picture and add to it without losing the previous ones. It's a permanent memory which most people don't have.

Consistency

The written document provides a similar story about the project so that everyone can participate with the same expectations.

Quick overview

When a new member to the project needs to get up to speed and doesn't know the problem domain well, this can be the one place where they spend the bulk of their time to learn what's going on.

Protection of intellectual property

Why allow the two people who really understand what the best way to develop the software to hold you hostage while the project is being managed? These two could walk out and leave the company with a very big need for expertise and experience. Having the two SMEs document what they know allows the knowledge to stay in the company.

Scope is changeable

The value that comes through scope does not always stay put where you first found it. The business is constantly keeping up with their customers through the changeable forces of

- The marketplace
- Price of goods
- Customer wants and needs
- Competition
- New constraints

The two main components of business analysis, data and process, are always in the state of flux. The least changeable is data. Process however, is always needing some maintenance fixes as decisions on how to manage the data improves.

Scope components

✓ **Processes = trigger, steps, inputs, outputs, rules, dependencies, goal**

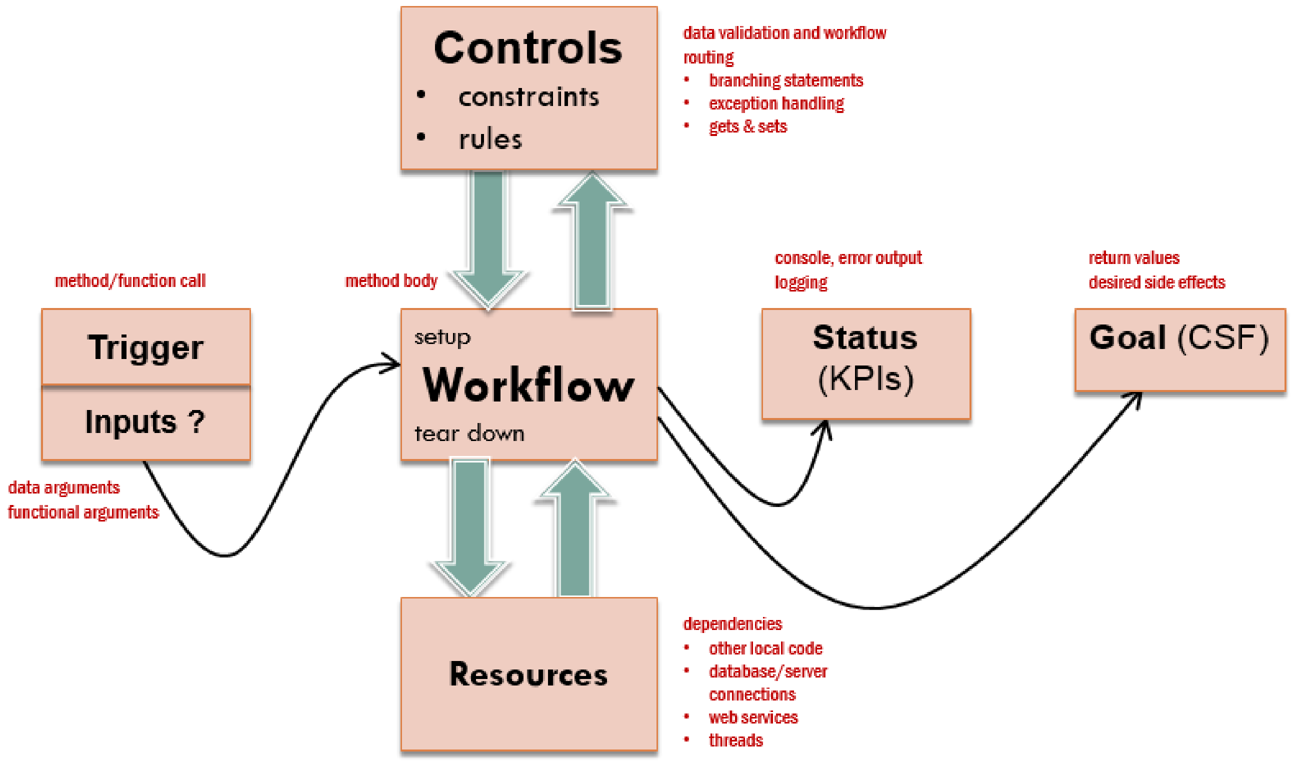
✓ **Data = types, units, rules, relationships**

Breaking down a business function or a project that is going to alter the flow of business requires an understanding of the components that have to be created and detailed. There is no need for computer terminology at this point because the computer is just a tool that automates the business services. It is the computer, in fact, that helped us create a well-defined set of criteria for defining.

Processes are the verbs of the language of business scope. They are the substance of the flow charts and GANTT charts. They give us a grasp on time and effort.

Data are the nouns of the language of business scope. They are the subject of manipulation for making decisions by sorting, searching, and comparing. They give us a grasp on why we choose a process.

A process model



The parts of a process that should be defined in any situation to understand it completely are all the same. A computer will need each one of these parts defined in order to work completely without fault and without a programmer guessing.

At the first there is a trigger that starts the process activity. Sometimes that trigger also has information that needs to be used during the process steps That information are the inputs of the process and help guide the process along the way.

Something similar are the resources that are needed during the process that keep the fire stoked and burning. It could be other data sources or people that are brought in to share the load. It could be other raw materials necessary to be shipped in because they aren't at the factory.

The main core of the process is the process workflow. It starts with a preparatory state where the environment for the process is set up and made ready and ends with a tear down state that cleans up all the work that was done during the steps. If you are just washing dishes, you'll need to prepare the kitchen sink area with soap, water, and cleaning supplies. After the dishes are done, you'll need to wipe down the counters and check supplies before the next time.

The workflow steps themselves can be executed one after another if this is a single person working on the tasks or it could be a concurrent process where many different people are working at the same time on parts of the tasks.

During the execution of those steps there might be controls that are brought to bear for how the tasks are to be done. A car can't just travel down the road without traffic laws to follow. And when you work with data, there are many values that don't really make sense like a negative age or a one digit zip code. For the most part there are two terms that have been used interchangeably:

- **Constraints** are controls that exist generally within project management
- **Rules** are controls that exist for either the workflow or the data of the project

Also during the execution of those steps, a quality check is useful. The Key Process Indicator (KPI) is a metric that is used to collect useful data that allows a manager to make sure the process is continuing on the right path. These status reports can be weekly checks on time and budget spend or knowing that you hit a milestone within 5% of expectations.

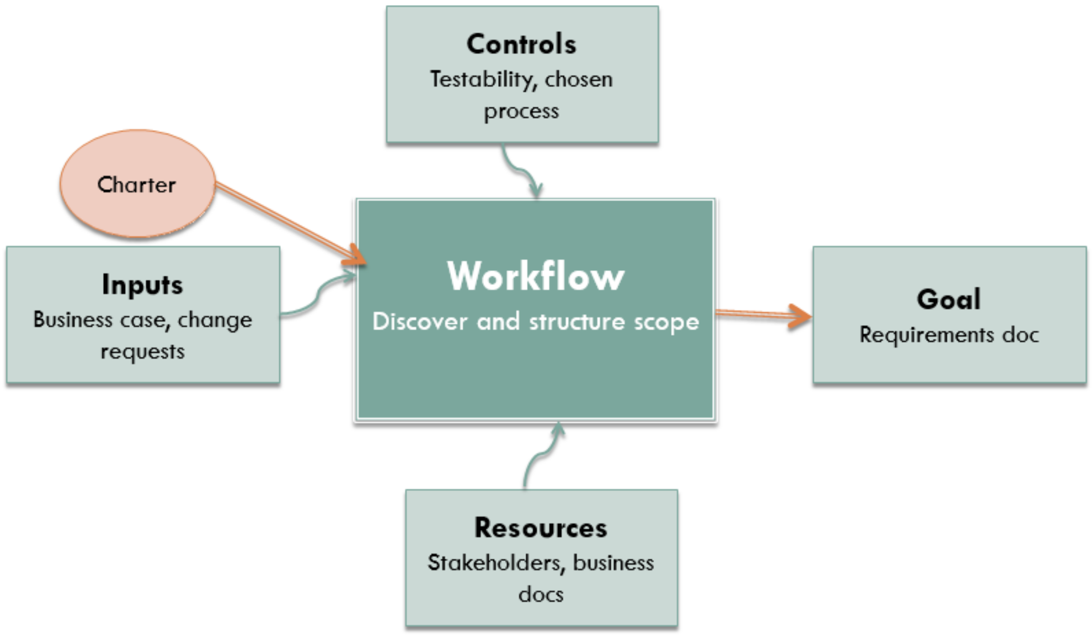
Finally, there is the goal that the process has in mind for producing the value that the business is concerned with. It is the value that is above all, the most important part of the process to understand and create. The goal is defined at a high level usually and helps to keep the project staff on track. It's often referred to as a Critical Success Factor (CSF) and the final check of value that was gained by the business is why it is the critical part.

Maturity levels

In any process, there are stages that people go through from novice to pro. This are called the maturity levels. Most systems identify five different levels.

1. The first level is when no one has a great idea and people just get by. It's heroic individual effort that stands out and the context is that of dealing with problems without solutions.
2. The next level up is when rules exist but have not been standardized. Most businesses are at this stage.
3. The efficient business is concerned with standards. When processes are understood, the maturity level has been attained where the complete business is under a simple level of control.
4. When the business takes control of the understood processes, they have started to optimize the processes and have started to break down the rigid systems of control that is both helpful but frustrating.
5. The final level is where a mastery has been attained and what people need and what relationships work to bring about the best business decision is known.

BA process


Controls
Testability, chosen process
Charter
Inputs
Business case, change requests
Workflow
Discover and structure scope
Goal
Requirements doc
Resources
Stakeholders, business docs

The process for business analysis can be outlined in the process model manner. Initially, on a project, it's the charter or decision to fund the project that creates the trigger for the project analysis to begin in earnest. The business analyst may be brought in before the charter is made to help create a business case for the project when the strategic analysis needs more expert advice.

The business case, or strategic problem definition, is the beginning document that analysts work with and change requests can also be added at the beginning or during the steps. People, such as the stakeholders, and business documents are used to provide information about the project and are considered the resources of the process.

The **workflow** of the BA process are not well defined usually unless there are similar projects which then lends itself to more reusability of the steps. But the purpose of the steps is to provide the scope in a useful structure to the people that take on the design and execution of the project.

Controls are used to provide help in a corporate environment to standardize project management and understand the success of the process. So tests are created to check for quality and for the best set of process steps to use.

Requirements analysis cycle

- Model the current business state (baseline)
  - Select model types
  - Create, update and verify models
- Model the project's scope
  - Select model types
  - Create, update and verify models

    - Obtain agreement from sponsor and project manager
- Model the project's scope in detail
    - Select model types
    - Create, refine and verify models
    - Structure detailed models to hide detail
- Prioritize requirements
    - Agree on priority metrics
    - Apply metrics

User involvement

Keeping the users involved is one major key to having a successful project but isn't always easy to do. Some of the roles that the users can play are:

- design team member for user interface
- subject matter expert
- brainstorming session participant
- acceptance testing group member
- user interface reviewer
- usability testing group member
- steering committee member

Design Components

Analysis skills

Analysis steps

The first part of requirements gathering requires an understanding of the real-world problems and the user's needs. The goal here is to gain a better understanding of the problem being solved before development begins. We seek the root cause which is the problem behind the problem.

The five steps for dealing with the analysis problems are

1. Identify stakeholders and users.
2. Gain agreement on the problem definition.
3. Understand the root causes.
4. Identify the constraints to be imposed on the solution.
5. Define the solution system boundary.

Identify stakeholders and users

You want to find the people who care about the system which will be anyone who could be materially affected by the implementation of the new system and those who are sponsoring the project. Questions that will help you find the people who are directly involved and indirectly involved are:

Questions to ask in stakeholder analysis are:

- Who are the users of the system?
- Who is the customer (buyer) of the system?
- Who else will be affected by the outputs of the system?
- Who will evaluate and bless the system at delivery and deployment?
- Who will maintain the system?
- Are there other internal / external users with needs?

Typical answers for these questions are:

- The sponsor who pays for the system.
- Users from departments who will use the system daily.
- Managers of those departments
- Customers of the system owners.
- Business partners like suppliers, banks, partners, etc.
- Authorities like inspectors, auditors, and government people.
- IT people if the product is being developed in-house
- Anyone else providing resources for the project.

Other types of questions to ask that will lead to other stakeholders about identified stakeholders might be:

- What goals do they have?
- What can they contribute?
- What risks and costs do they know about?
- What solutions and resources can they offer?

The work involved in requirements gathering is to elicit the requirements from the stakeholders. Requirements engineers or analysts are the roles that people use to do that work. Typically programmers do the best work because they have a vested interest in getting good requirements so they can program to them but anyone can do the job including non-programmers like marketers and expert users.

Identifying your stakeholders is necessary to ensure the success of the project. Each one whether they contribute to the resources through people or money must feel that they are getting something back or they may pull their support and even try to compete or stop it.

The kick-off or blast-off meeting is a typical place to round up as many people as possible so that these questions can be answered. Get to all the people that you think are important individually if they can't all show up at the initial meeting. Some may not want to participate due to a bad experience in a previous project but you can gain ground by personally attending to their needs and possible get them to be involved in a later meeting.

Gain agreement on the problem

✓ **Write the problem without a solution.**

When your stakeholders don't agree on what the needs of the whole group should be, there can be power struggles, issues about costs, resource provider issues, and who should own the risk of a certain process. Negotiation boils down to having an arbiter resolve the issue by gaining an understanding of the two conflicting needs and providing a requirement that satisfies both.

If the issue is more emotional and the team is looking for consensus on an issue that is not about getting to a final requirement, then the delay will kill the project and an executive decision must be made after all parties have communicated their opinions.

Consensus building may sound good but will take much more time and resources that you have if you have differing opinions and opinionated differences.

One of the easiest ways to gain agreement on the problem is to write the problem down and see if everyone agrees. This is where you use the standard format of problem writing which is

"this problem… affects these people… with this unhappy result for the business... but our solution would benefit us by…"

- Standard format (best)

> The task of putting a full-sized canoe on top of a tall vehicle such as a van, camper, or tall SUV is difficult for many average adults as well as by older individuals with decreased strength. Unsuccessful attempts include damaging the vehicle, straining muscles, and restriction of the normal operation of the vehicle. Our solution will allow us to easily raise, position and secure a full-size canoe in varying weather conditions, on uneven and possibly slippery terrain.

Problem statements may try to outline system requirements as well. Sometimes, the problem statement is the way to introduce the final product features to the user. Here are some bad examples:

- Disguised as requirements statements

> Design a device that will enable one person to easily raise, position and secure a full-size canoe on top of a tall vehicle such as a van, camper, or tall SUV. The device may be used by adults of average height and strength, or by older individuals with decreased strength. The device will be used outdoors in varying weather conditions, on uneven and possibly slippery terrain. The device should be easy to install and use, inexpensive, lightweight and suitable for installation on a variety of tall vehicles. The device must not cause damage to the vehicle or the canoe, nor restrict normal operation of the vehicle.

- Disguised as requirements statements

> The Piper Aztec PC-23-250 twin-engine aircraft has no constant flow aerial fire retardant delivery system currently. A delivery system should be designed to allow a flow gating to permit selection of flow rates between 100-900 gallons/second. Accuracy of each retardant drop is to be within +/- 25% of the specified volume for each 1/4-load drop. Total minimum tank volume must be 2000 gallons, with maximum tank structure weight not to exceed 1500 lbs. The system must be designed within space and operational constraints of the aircraft.

- Disguised as feature statements

> EasyLock is a portable application that does not require any installation process on the host computer and is always portable. Wherever your Verbatim USB flash drive goes, EasyLock is saved on the device and can be used with any Windows XP, Vista and Windows 7 computer. With the intuitive

A problem well stated is half solved.
Charles Kettering

Drag-and-Drop interface, files can be quickly copied to and from the device for fast, secure and efficient workflow.

Can you work out your own problem statement for this product possible starting with "The security of data on a portable drive…" ?

The formulation of a problem is often more essential than its solution.

Albert Einstein

Exercise – problem definition

Write the problem definition of the following scenarios using the standard format of problem writing from above. Be creative in interpreting the context of the problem.

a) One register with a long line is open at a busy grocery store.

b) Oil stains are appearing in the garage on the concrete floor.

c) I have less money to spend this year than I did last year.

d) Competition and budget cuts are forcing us to eliminate valuable staff people.

e) I lose too much time by driving to work.

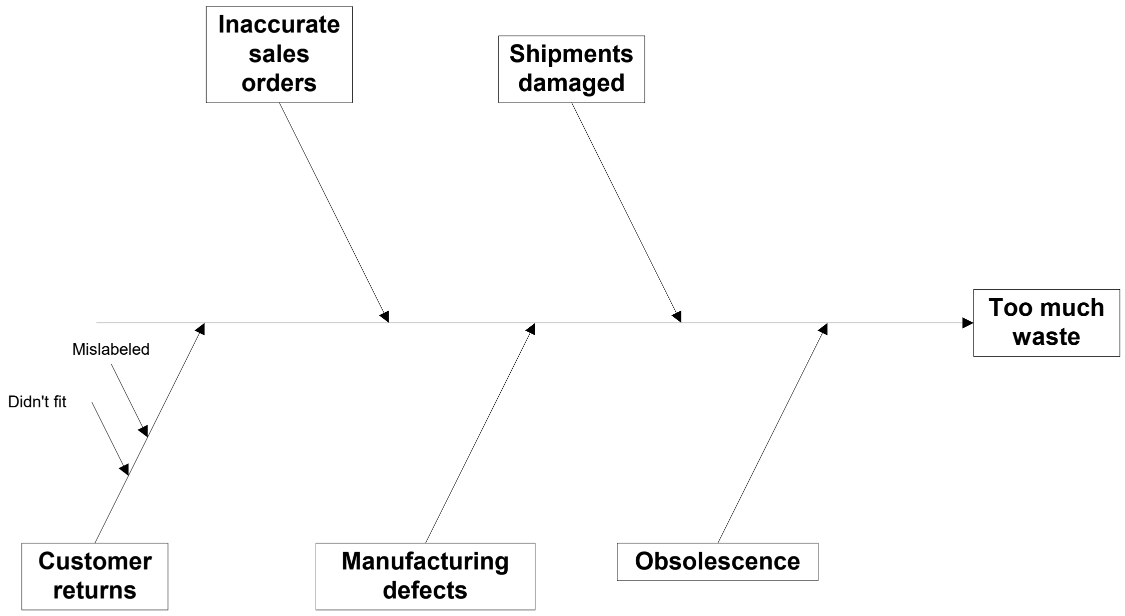
Understand the root causes

✓ **A root cause points to a process that failed or does not exist**

Finding a root cause or causes is one type of problem solving strategy. Others might include divide and conquer, means-end analysis, reduction, research or trial and error. But asking questions about what you know is a good first approach.

The root cause is the problem behind the problem but because it is hidden, you may not know that it is the root cause. It's a question that you are always asking because you never know that it's been answered adequately. So even though you've asked the stakeholders what the problem is, you may want to ask them again, and possibly several more times to get it until your intuition says that you really have it.

Brainstorming or using charts like a fishbone diagram to analyze the situation helps to bring a little process and visual support to the problem. They tend to work well in a smaller group for documenting root cause ideas but take up a large space on a whiteboard.



1 - A fishbone diagram

If you analyze further and find that the majority of waste was caused by the inaccurate sales orders then you have a solid cause for developing a system. The new system's cost compared to the amount of waste should give you a return on investment. Then you can add more detail to the fishbone or start a new one that is focused on the sales orders.

The real root cause should point toward a process that is not working well or does not exist. Untrained facilitators will often observe that answers seem to point towards classical answers such as not enough time, not enough investments, or not enough manpower. Although partially true, they don't point at the bull's-eye of the problem so it may be more helpful to ask "Why did the process fail?"

The types of questions to be asked generally start with trying to find a functional requirement. This is what the system must be able to do. So if the system is going to exhibit a type of behavior, the questions become

- What value will the stakeholder get because the system will perform this task?
- What is the task that the system must do that would cause this behavior to occur?
- What is the action that the system is performing in order for the results that you desire?
- Would the functional requirement remain the same if this task were performed in front of a real person?
- Would the functional requirement remain the same if this task were performed over the phone?

Then you can concentrate on the non-functional types of behaviors which are reflected in these types of questions:

- What are the traits of the system that you must have when the system is performing the tasks?
- What makes the tasks of the system different than just satisfying the basic functional needs?
- Why is it important that you get this type of modification to the system?

Once you figure out what the root causes are, what do you do about it? You probably want to fix all of them. Often it won't work because quality data routinely shows that a number of root causes are simply not worth fixing because the cost of fixing them exceeds the cost of the problem.

The Five Whys

✓ **A good root cause is actionable**

A root cause analysis can start with asking the question "why?" and getting a new answer. Sometimes, asking the question up to five times can lead to a profitable answer. This technique was developed at Toyota in order to dig down to a root cause and named the Five Whys.

In general, you can keep asking until you get back to a point where you can't do anything about that. That point may be reached a lot earlier than five steps back. But you should stop when you get back to the point where the control gets out of your hands even if it's sooner than the five steps back.

Exercise – Root causes

Rewrite these “requirements” to eliminate the solution so that it focuses on the root cause of the need. Typically, there are one or two functional and many non-functional requirements. Sometimes there may be no functional requirement. Also identify the system.

a) The car shall have power windows.
System: car

b) The user will press the submit button when finished with the customer data entry.
System:

c) The food for the meal will consist of four courses.
System:

d) The system will illuminate the screen at night.
System:

e) The database will store customer transactions.
System:

Strategic Analysis

Constraints

- ✓ **Constraints are known before the project starts.**
- ✓ **Constraints on process or data are called business rules or data validation rules.**
- ✓ **A constraint combined with probability of occurrence becomes a risk.**
- ✓ **Assumptions are risks used to move ahead with designs**

Boundaries are the limitations to anything. Constraints are boundaries that restrict the project before the project starts. A constraint may be an internal business decision or an external legal requirement. In a project the constraint of lack of time or budget can affect what scope is able to be accomplished. Many types of business forces can alter the final design of the project.

System and functional constraints are often written as business rules in project documentation which limit or set the capabilities of the system.

When constraints are not a sure thing, they become a probable event that is measured for the value that can be variable in a project and are called risks. Project management then details those risks and how to mitigate or bypass negative risks.

When it is necessary to enter into the stage where a risk must be acted on by planning with the current architecture and resources available and a decision must be made, you start with the assumption that the risk will happen or not happen. This allows you to move ahead with the design stage and finally the development. But the project manager is still aware that the risk is there and that an assumption has been made so that progress can be made.

Constraints have been factored in as risk that the business is willing to take based on the likelihood of a profitable outcome before the project is launched. A change in the market, environment, legal system, or social attitudes can have a limiting effect on the business outcome. The project manager monitors for changes in the environment to risks.

Constraints are limitations on the project before chartering and will affect the scope of what requirements the project manager chooses but do not have to affect any one requirement. They could be budget, time, or other resource driven and are the concern of the project manager rather than the business analyst. It is common to use the word constraint when talking about any kind of limitation but the project documentation section for constraints is usually for project level limitations known before chartering and analysis begins.

For instance these statements can be constraints:

- The project will follow the HIPAA Privacy and Security Rules even though the enterprise is not a covered entity. (**process rule**)
- The staff will authorize workforce members who work with EPHI in locations where it might be accessed. (**data rule** - security access)
- The government will be passing a new law that disallows registration fees to be charged for any volunteer agency. (**process rule** with risk).
- Small medical practices cannot afford an internal HIPAA compliancy service. (**business case** - strategic market opportunity if we are not the practice)

Many types of constraints should be documented at the beginning of the project from stakeholder's sources. Here is a list to start with for questions to ask:

Economic

- What financial or budgetary constraints are applicable?
- Are there costs of goods sold or any product pricing considerations?
- Are there any licensing issues?

Political

- Are there internal or external political issues that affect potential solutions?
- Interdepartmental problems or issues?

Technical

- Are we restricted in our choice of technologies?
- Are we constrained to work within existing platforms or technologies?
- Are we prohibited from any new technologies?
- Are we to use any purchased software packages?
- Are there any concerns with open source software?

System

- Is the solution to be built on our existing systems?
- Must we maintain compatibility with existing solutions?
- What operating systems and environments must be supported?

Environmental

- Are the environmental or regulatory constraints?
- Legal?
- Security requirements?
- What other standards might we be restricted by?

Schedule and resources

- Is the schedule defined?
- Are we restricted to existing resources?
- Can we use outside labor?
- Can we expand resources? Temporary? Permanent?

Constraints & system boundary

Constraints are identified for the solution system through the consultation of analysts and business subject matter experts.

The boundary of the solution system is then determined through the compromise of the needs of the enterprise, the resources available to the project manager, and the constraints identified up to this point.

Exercise – constraint type

Unfortunately, our project did not have a good business case but we were able to get a high-level business requirement. Read it and then determine what kind of constraint each of these should these be considered? They can be classified as either a external constraint hard to get around, or an internal constraint like a business rule that can be a tradition but possibly changed. Discuss.

Business requirement - The Organization shall install moderate sized woody debris along stream banks to provide shelter for salmonid fingerlings.

- Access to creeks is limited to footpaths without motor vehicle access.
- Recurrent funding for the project will be met from state growth stimulus funding.
- Logging or removal of woody debris is allowed on-site after getting property owner's permission and necessary local regulatory permits.
- Records must be kept of salmonid species strains used in production.
- Restoration of the habitat debris must be made in early spring.

Metrics

Requirements management can provide vital information. For example, by associating costs with requirements and using the data to estimate similar work, improvements in cost estimates are possible. The development processes must be stable enough to make the cost data comparable, however.

Some measures of the effectiveness of the requirements process include:

- the status of each requirement
- the cumulative number of changes (including the cumulative number proposed, open, approved, and incorporated into the baseline)

- effort and funds expended

Project management

Project phases

Any kind of project has five significant parts to its cycle. They are

- **Initiation** – strategy, identification of opportunity of value and constraints
- **Analysis** – requirements gathering, analysis, structuring, documentation
- **Design** - combining needs and constraints to create a plan
- **Development** - Building, testing, deployment
- **Operations** - Day to day production of value

Project concepts

Important management concepts are used throughout the project and can include:

- Strategy, business goals and value, constraints
- Uncategorized needs, design wishes, beliefs
- Requirements, entities, design ideas, responsibilities
- Architecture, risk, assumptions, classes
- Unit testing, TDD, SCM
- Integration, system, etc. tests
- Continuous integration
- Service desk support

Project deliverables

The project is typically split into phases in a project management style by the deliverable that results at the end of the phase:

- **Initiation** – Business case, project plan, glossary
- **Analysis** – Meeting notes, needs lists, Use cases, ERDs, rule models, interface models, data dictionary, screens, external text
- **Design** – Same as above but with more detail for dealing with constraints
- **Development** – Code, SCM plan, Test plan, test cases, test data, test results, Deployment plan
- **Operations** – Tickets, bug reports

Pre-charter documents

After the need for a new system or modification to the current system is identified, it is analyzed further to define the requirement during the phase where the project has not been chartered yet (approved and funded). Types of initial documents created are:

- **A feasibility study -** to answer questions in more detail such as: what needs to be done, what is the value of doing it, what business goals are supported, how it can be done, what are the alternatives, and what are the risks.
- **A business case** is created to document the results of the feasibility study. Requirements at this stage are insufficiently detailed to build or modify a system. Most documents become part of this one in business before the project is launched.
- **A project strategy plan** may then be initiated based on the selected alternative from the business case.
- **The requirements traceability matrix** (RTM) is ideally started now. The high level requirements are linked to the business goals. As work products are created throughout development, the RTM will be updated to provide full traceability.
- An **Operations Concept document** may be created to convey uses of the system and to set the system and project boundaries.
- A **top-level architecture** document may also be developed.

Project types

Projects have distinct sets of features that repeat themselves across the same types of projects. If you look only at a restricted type of project it is easier to understand. The typical project type that is assumed is that of a system. It is only concerned with software development.

The business only type of project is concerned only with a change in business process and using current automated solutions. Most business people would call this a process re-engineering

Another common type of project is that of updating the system to add new features or features that weren't able to be included in the first release. This sometimes is called a phase two or maintenance project.

But by far, the most common type of project is that of a combination of one or more of each of the projects.

Service delivery types

Requirements will be used by the customer to make sure that they reflect their needs so they will have to be able to read it and understand it. The type of project helps determine the initial structure of the project management plan and gives a way for business management to categorize and then understand the types of services and methods they are delivered by.

- **In-house development** has both the customer and supplier in the same company, usually a large corporation. Many had no or poor requirements and were swept under the rug after the project failed. Many companies are improving their written requirements because of this.
- **Product development** involves designing a commercial product that is developed outside of the company. A problem with this is that the developers never get to see the real customer and have to depend on the marketing requirements to be accurate.

- **Time and material based development** is a loose way to developing using a software consulting firm where the company pays by the month and the requirements are informal and allowed to develop over time.
- **Commercial off-the-shelf** (COTS) purchase is when you use an existing software package to satisfy the requirements by configuring it however you can. Sometimes they can be extended or become the basis for development projects themselves if they are a software framework like Lotus Notes or an ERP system.
- A **tender process** involves sending out requests for proposals (RFPs) and vendors are invited to submit proposals. The tender documentation has elaborate requirement specifications. The customer then selects the best proposal. These are difficult to write requirements for because a contract is based on a reply to the proposal and bargaining is not permitted usually afterwards.
- **Contract development** is when a supplier develops a system for the customer company based on a mutually agreed upon requirements specification and contract.
- **Sub-contracting** is where a person develops a whole or part of a system to a main contractor who then delivers it to a customer. Both people usually talk technically in contrast to all the other types of processes here.

Requirements management plan

✓ **A plan for what you do as a BA is included in the overall project plan**

Even though it is typically not created, an understanding of the process you follow for what you do as a business analyst could be documented to create a full understanding of how the project is going to be managed. This then becomes a component of the project management plan.

Different types of sections to the requirements management plan or created in other documents that would be useful to consider before embarking on a new project would be:

- the document scope
- people involved with authorities and responsibilities
- process of gathering
- process of structuring
- process of changes to requirements
- quality assurance including traceability and change control
- issues affecting implementation of the plan, such as training on the requirements management tool
- applicable documents, such as policies and standards
- methods and tools that will be used for the requirements management process (or the requirements for selecting a tool if one is not selected)
- appendices usually contain a discussion of quality factors, as well as references, any forms to be used in the process, and additional details not included in the main body of the plan, such as report examples

Change management

Change management is the sum of the processes that allow for the smooth transition of the status quo to a new state that involves the completion of a project. During the transition, it's important to collect information about the current state as well as the state of the system you are moving to. This allows for a way to back out of a change to a workable state. The current state is sometimes known as the baseline.

Change management - Project tasks

✓ **Reduce risk of failure by using standard processes and documenting changes**

The business is interested in producing value for the customer and anything that changes to reduce that value is to be avoided. Therefore, the main goals of the project manager while facilitating the changeover to the new business process are to:

- Minimize impact / disruption
- Optimize risk exposure
- Control changes through standardized methods and procedures
- Document changes

Change management - BA tasks

✓ **Before a major change, create a baseline.**

✓ **Control and trace your changes**

The first task of any project whether it be a new process, a modification to an old process, or even a mothballing of a process no longer needed should be to understand what currently exists. That's what setting a baseline is about.

There are baselines for the current data structure and the current processes that can also include the current business rules in effect. Once the baseline is taken, long term goals can be set and measured for progress from that baseline. Milestones along the way can be ascertained. Metrics to measure what kinds of activity or data that support the achievements are decided. Without a baseline, an objective measurement for managing by data can be made. Without data, management is flying by the seat of their pants.

Many types of techniques are available for business process analysis. UML uses a type of diagram called the activity diagram and is similar to that seen in software systems development but varied slightly.

Another system is the Task On Arrow approach which is more like a data flow diagram. When a company decides to do Business Process Re-engineering (BPR) this business process understanding is critical. For just working with a system that integrates with the business, it would be very helpful. Either way, having a baseline to chart against is a great way to keep track of progress.

Then after the baseline is complete, you can control the changes through a ticket system or another type system and accept them, evaluate the changes to the current or planned system, and then integrate them into the design and final product. While you are managing those changes, you will want to keep track or trace them through the process for help when producing any reports that help you increase the quality of your effort on the next project. Each change should be identified and documented so that you can have a breadcrumb path through the project from initial submission to final outcome.

Scope creep

✓ **Scope creep is probably missed requirements**

Requirements documents will be continually clarified throughout the project as an iterative approach is followed or even in one massive phase. It's because the requirements don't all become clear at once. As you analyze and design, new requirements pop up. It is usually talked about in terms of scope creep which is just completing the requirements gathering process. You will always get new requirements as you step through the analysis and design phases and keep communicating with the customer. The more light you shed on the problem domain, the more requirements you see.

Some scope creep is due to market changes and should have been documented as a risk. If the requirements need amending from a risk now looming, it is the duty of the project manager to adjust the factors of the project to make the project not cause undue stress on the project members. Then instead of calling it scope creep, it becomes a managed risk.

Another type of scope creep is when you get a request to do something that was not foreseen. It becomes a change request and is handled within that process.

Roles

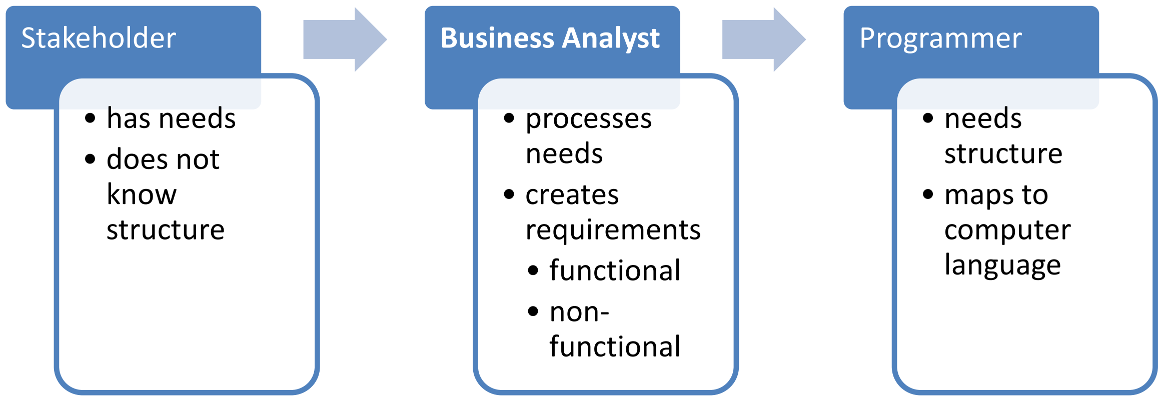
Role definition

Project roles

- **Initiation** – Executives, SMEs
- **Analysis** – Project managers, business analysts, DBAs, stakeholders
- **Design** - UX, GUI designers, architects
- **Development** - Programmers, testers
- **Operations** - Stakeholders

Role interactions

The process by role



Stakeholder role

✓ **The stakeholder provides processes, business rules, and data used for business.**

Stakeholders are those people or organizations that have input that will influence the way that the project will be done. Stakeholders have a process, business rule, or kind of data that requires the relationship between them and the organization and the proposed change will affect the outcome of our solution.

Needs of the stakeholders are chaotic groups of perceived required process steps, data items or ways of doing business by the stakeholder related to the project. They are expressed in less than perfect forms. But, stakeholders are not required to have any structured knowledge of their needs. They have a very good experiential knowledge for what the right business model is that works for them. And it's these experiences that need to be extracted and understood so they can be made reusable and communicated.

Business analyst role

✓ **The BA must understand and communicate scope.**

✓ **The BA decides what gives the most value to the business**

The business analyst must make sense out of stakeholders' wants by organizing them into a document based on the value to the business, and if a system project, documenting the scope of that system under construction, and creating some structure for the programmer to follow. Each level of analysis peels off another layer of the onion until it understands the direction completely. Applying that knowledge to a set of constraints and best practices produces the design specification that refers back to the original needs of the project.

The analyst's goal is to understand and communicate the business scope as seen by the stakeholders and gathers its priorities. The analyst does not finalize the scope.

Project manager role

✓ **The project manager decides project goals and leads members to those goals.**

The project manager is an engineer who manages the three major parts of the project of

- scope
- time
- budget

Often the project is constrained by time and money and even scope can be constrained. With the knowledge of the constraints and the possible scope with priorities, the project manager can make good decisions for what scope should be implemented.

It's because of the need for scope advice from the business analyst that the project manager and the BA work closely together at the beginning of the project.

The ability of the manager to complete a project by obtaining the goals originally proposed is a hard task that usually involves quite a bit of people skills as well as tracking and monitoring of the three major parts of the project.

Blame chain vs. quality chain

Review the types of views that the different roles have when being a positive influence on the project vs. what a negative influence looks like so you recognize when you are contributing to the quality of the overall project.

The blame chain

BUSINESS	Trial & error	Tradition	"The market is crazy."
ANALYST	Document what they tell you.	Give them what they want	"Users don't know what they want."
CODER	Code what you're told.	Make the schedule and PM happy.	"Analysts don't understand what I need."
TESTER	Test until the deadline.	Follow what everybody else did.	"I tried to find as many bugs as I could."

The quality chain

BUSINESS	Customers have real needs.	"What can I do to help them solve their problems?"	
ANALYST	I have to communicate needs clearly so they can be tested.	"How does what the customer does now help them?"	
CODER	I have to make everyone's job easier including mine.	"How can I translate requirements simply to code?"	
TESTER	The customer needs the best app possible.	"How can I guarantee quality?"	"How can I let everyone know how well they did?"

Post Analysis

✓ **Know your programmer**

✓ **Know what makes good code**

It's very helpful to know what goal you are trying to achieve. The main goal that the analyst focuses on is completing the documents. But the main reader of the documents for a software project is going to be the programmer. The first rule of communication is to know who you are communicating to. That means you should understand your programmer and the needs they have.

Software has needs to and they are better addressed at the beginning stages of analysis so they can be respected throughout the design. Without the understanding the path to a better piece of software is much harder to achieve.

The needs of software are mostly about structure and reusability. Structured workflow becomes units of code. Structured data becomes sufficiently detailed names for data. Other types of qualities that provide good code are being able to understand other programmers' code easily and keeping the code simple so that maintenance is easy.

Unfortunately, as we increase the maintainability of code, the reusability of code decreases. Programmers must balance the two for a good software design. Analysts can provide a solid structure to make those choices upon so that the complexity of the job is lessened.

If the analyst provides too much unstructured detail, the programmers must resort to doing scope and data analysis on their own. They write comments about how things should work and create lots of possible solutions without knowing if they are based in reality. They have been trained to do this by schools that don't use real world scenarios. Testers who then test the code have no solid plan to use to gauge the quality of the code. When analysts do their job in structuring the scope of the project, they inherently produce testable scope statements making the testers' job easy.

The list of ways that programmers have tried to solve the analysis problem are many. Without a basis in value to the business, the limits that programmers have will always produce a flawed model of scope. Here's some of the ways that are being used to manage the scope of projects when analysts are not doing their job:

- XP – Extreme Programming
- FDD – Feature Driven Development
- MDD – Model Driven Development
- DDD – Data Driven Development
- TDD – Test Driven Development
- BDD – Behavior Driven Development
- CRC cards - Class, responsibility, collaborators
- AM – Agile Modeling

The most popular of them is TDD. This essentially trains the programmer to write structured requirements in the code and follow the requirements when writing the code from a simple implementation to the one that fulfills the user's needs.

Requirements

Requirement development

Requirements are developed in four phases:

- elicitation
- analysis
- structuring
- verification

During the elicitation phase, stakeholders and business documents are used to gather inputs for the steps. Then, the reviewer identifies the specific system functional and performance groups into which the requirements fall which then may be further categorized into subgroups. Use cases are identified at this stage and a use case model can be developed.

Requirements are then analyzed to ensure they are complete, consistent, testable, and feasible within budget and technical constraints. The requirements are also reviewed to eliminate redundancy. Acceptance criteria for each requirement are developed. A draft system specification is produced.

Structuring is then the process of writing them in a different form such as the detailed use case or a commonly used template.

Verification is the last step but really one that occurs before you go back to the first step and repeat this set of phases in order to gain more and more detail. Models are created and meetings are made in order to get feedback on the structured requirements you've been able to compile up to this point.

The repeating of these phases continue until the understanding of the process and data are complete enough that you could give the likely person in charge of the process a manual and almost any case would be thoroughly understood about how to take care of the value being created at this point. Computers are exceedingly stupid so just imagine that you are talking with a business savvy person but at the same time, they're really dense.

What are requirements?

✓ **A need is not a requirement until it is rewritten.**

Requirements gathering is an art of understanding various techniques and applying them at the right time for the right reason to understand what will make stakeholders' satisfied for the value that they are willing to forego. A project will naturally require the use of multiple types of techniques.

A need is the basic request from the stakeholder for something they want or need that generates the fodder for your requirements analysis. They change, they aren't clear, they hide root causes, they focus on the solution, and they have little structure.

A requirement is a need rephrased in a structured format. If it is no longer wanted or needed, it ceases to be a requirement. For example, 19 inch monitors were specified as a technical requirement. After some experience, it was realized the monitors should be 21 inches. The 19 inch monitors are no longer a requirement. Requirements are put into the structure that best facilitates the next step of the project which, for example, for an OO software project would be structured by entities.

A software requirement is a goal that a computer achieves to automate a task. There may be similar programs or you may have rough sketches of what it should look like but not until the program is written successfully can you write the requirements exactly. Until then, it is just an effect that you want the computer to exert on the problem domain.

A requirements specification is written in close collaboration between the customer and the supplier or representatives and describes what a system should do both functionally and non-functionally. The programmers and other developers use it after that. It acts as a kind of contract to what was agreed on. At the end of development, the requirements are tested to verify that the program features trace back to the requirements documents.

The problem domain description is an important addition to the requirements statements. Besides describing how the problem domain is affected by the change to the new system, it is important to describe how the domain currently is.

If you tell a user that something is not a requirement when it actually is, you will generate arguments and make analysis not only a hard phase to go through but end up redoing your work several times over.

If things go legally wrong, don't depend on requirements specifications to absolve you in court from the customer's demands. Many developers believe that if they satisfy the written requirements, they will win the court case. However, in most countries courts don't work that way. If the customer had reasonable expectations that were not written in the specifications, the court will rule that the supplier must fulfill this expectation. In other words, courts acknowledge reasonable tacit requirements.

Exercise – requirements basics

Think about and discuss these questions amongst yourselves. Choose your answer based on the priority of the issue and the culture that you are in. A class discussion will follow to see what everyone thinks.

a. The customer's system crashes whenever the user forgets to fill in the discount field on the order screen.
   i. Is this an error that the supplier has to fix ?
   ii. Is this a requirements issue?
b. The customer's system has an adequate response time but over time the database becomes fragmented and the response time increases to be incredibly slow. The system has no utility to re-index the database to make it run faster.

    i. Would a court rule that such a feature would be a reasonable expectation that the supplier should have known about?
    ii. Should the customer be expected to know about re-indexing?

c. The customer received a business application that can produce invoices (statement of funds owed) but could not produce credit memos (statement of funds to return e.g. damaged goods on delivery).
    i. Would a court say that this kind of feature is a standard feature in this type of software?
    ii. Should the supplier have known about this?

Business requirements

✓ **Why will this project give the business value?**

Business requirements are created by the sponsor of the project and come to the project manager as part of the business case. They focus on what the capabilities are of the system, service, product, operational process improvement, or infrastructure enhancement. Usually the effects and activities that are required for the future state are identified before any estimate is made. Any

The style that business requirements are written in is a simple sentence that begins with “the capability to…” so that a system name is avoided. From there the structure can follow the order of:

1. do an action – e.g. capture customer info
2. show workflow or transactions – to support new and add-on orders
3. note business segment – for High-End Business
4. note product or services – in GSD services
5. give purpose or objective – in an integrated tool
6. identify a user – to be used by the field sales group.

There usually are only a limited number of business requirements and about ten is a good number for a large project. You will have enough when some scenarios can be developed and the effects of the requirements can be figured out.

Good questions that are answered in these requirements are:

- what is the strategic intent?
- what is being accomplished (not how) ?
- what will be the business outcome?

Test requirements can be created at this stage.

Business requirements provide the scope of the project and roll up to the total business objectives or goals. They also specify the capabilities associated with a new or enhanced product or service, an operational process improvement, or an infrastructure enhancement.

User requirements and needs

The business requirements then allow the elicitation of the needs of the users. They are not in a structured format usable by programmers and therefore are not really requirements at this point. Programmers working from them without any other models will need to restate and reorganize them before coding and sometimes during and after coding which is called refactoring.

Enterprise analysis

Enterprise analysis

Enterprise analysis is the understanding and documentation of the business at a high level so that the business can perpetuate itself and foresee strategic opportunities. Considering the operations of a business as a project the types of enterprise activities would entail:

- Defining, assessment and validation of high level business goals
- Writing documentation to understand how to achieve those goals with input from all significant stakeholders.
- Planning an architecture that will support those goals
- Build out of architecture
- Architecture support and monitoring

Exercise – company mission

Write down your

- company's organizational reasons for existence
- goals and objectives
- accomplishment of those
- and how it needs to change

Students from the same company should work as a group to come up with a consensus.

Interface analysis

Data flow

An interface analysis is just showing the broad picture of how the system should fit into the current infrastructure and knowing what systems will depend on it and what it will depend on. The questions that can be relevant to ask at this point include:

- Who will supply, use, or remove information from the system?
- Who will operate the system?
- Who will perform any system maintenance?
- Where will the system be used?
- Where does the system get its information?
- What other external systems will interact with the system?

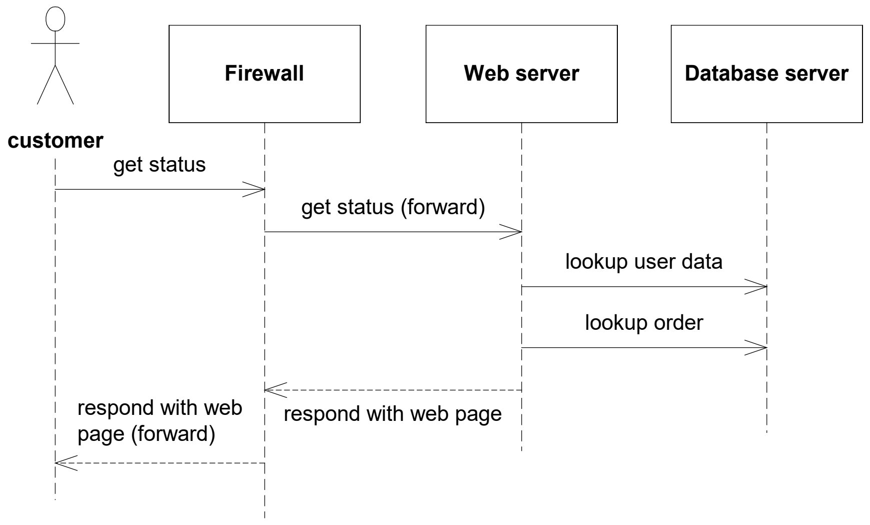
System-level sequence diagram

✓ **Sequence diagrams show systems or roles that participate in our project**

✓ **Sequence diagrams show how often we communicate with other systems or roles**

✓ **Sequence diagrams show events by trigger in time order**

At this point you could create a simplified diagram that just shows the roles of people involved and the systems involved. A Unified Modeling Language (UML) diagram that is useful to show many system interactions is the system-level sequence diagram.



2 - UML system level sequence diagram

Roles of people and systems are not the same as the job that they perform. The job title is a business term but the role name is part of the interface of our system. You could map the job titles to certain roles but it could be possible that the same job title could generate different roles. The role identifies that we are interested in a certain access to some functionality in the system.

Exercise – System level sequence diagram (ATM)

Read through the requirements for the ATM system (see ATM user requirements at back of book). Draw a system level sequence diagram for the ATM system that represents the interaction of the customer when they go to withdraw cash.

Requirements elicitation

The different sources of the requirements demand different techniques of elicitation. Acquisition is also called elicitation in many resources.

In talking to the people that are needed to ensure the success of the project, the first task is to make sure that you don't leave anyone out. Then you have to find out what their interests are by asking basic questions like these:

- What goals do you see for the system?
- What would you like to contribute?
- What risks or costs do you see?
- What kind of solutions and suppliers do you see?

Elicitation process

The traditional approach to getting requirements is to interview the stakeholders, study the existing documents, conduct workshops, and brainstorm. Finally you analyze the results and write the documents. The business goal doesn't need to be written because the business case has already been approved.

The work is reviewed after the first draft, the customer validates it, and it is checked for consistency. Because the customer has little time to participate, in many cases the analysts do most of this work by themselves.

An upgrade or a technical component as a part of a system will lead analysts to gather more design or technical requirements whereas a new system should be more business or analysis level requirements specification.

Barriers

Acquisition can be difficult because our stakeholders may have difficulty in expressing their needs, or they may ask for a solution that does not meet their real needs. They focus on today's problems and not the big picture and express it as a problem instead of a requirement.

Stakeholders may have conflicting demands. They don't have the skills to express the demands as tasks and many don't know why it's important to do the task they do. If you get someone who can express their needs, you may get too many non-essential requests.

Users find it difficult to imagine new ways of doing things or to imagine the consequences of things they ask for. When they, for instance, see the system that has been built for them, they often realized that it does not fulfill their expectations, although it fulfills the written requirements.

Sometimes there are not users because the product is completely new, and nobody has used IT for this purpose before.

Demands often change over time. When users, for instance, see a smart system somewhere, they may realize that they need something similar themselves. External factors may change too, such as operating system releases or new laws.

Overcoming objections

Objections are always going to be the excuses for not planning properly. How well you know the responses can help save a project. Here are a few of the most common objections and responses to get in your head for that budget or project meeting.

a. Manager: "We have done a management survey/brainstorm and **we know what we want**." "I'm a manager and I'm in the business for 20 years. I know what the user needs." "I'm a user, I know what I need."

What managers in the developing company think the website should offer, is certainly interesting information, but these managers are not the future users. What they want on the site is not (necessarily) what users want on the site.

What managers know about future users is what they need to know for their job. Since their job is not designing websites, they probably don't have the kind of knowledge about users that is needed to design interactive systems.

b. IT department: "**We have a team** with great ideas, creative concepts, and powerful technologies. Surely they can design a great system."

Creativity and technology provide solutions. We still need to answer a real problem with this solution or it will not be used.

Do you want to design a product that the creators think is great or a product that users think is great?

c. Know-it-all boss: "User requirements research is **not specific** for our product and our company. We don't want to pay for it. **We’ve got an** internet **expert**. He should tell us what people need on the internet."

A user experience designer can't be expert in all domains. His expertise consists of methods to learn about user needs and skills to translate these into an interactive solution.

A detailed understanding of users' needs is a competitive advantage. Companies developing an interactive system should not rely on public knowledge about users and their market. To obtain and maintain a competitive advantage with interactive systems, you have to know more about users than your competitor.

d. The accountant: "We **don't have the budget** for user requirements analysis."

The cost of "learning something about users you didn't want to know" at the start of a web project is small compared to the cost of learning it after launching the website.

e. Busy internet manager: "It **doesn't fit in our planning**. We want to launch our website in 2 months."

User requirements analysis can be scaled to the size of the project. It's better to do a 5-day analysis than no analysis at all. User requirements analysis should be planned from the beginning of a project.

Time-to-market is less important than being in the market with a product that is used by users. First mover advantage is often misunderstood. **You** don't create a first mover advantage by launching a website. It's **the user** who creates the first mover advantage by using your site. If they don't, no advantage.

f. Technological genius: "What we're developing is **something completely new**. There's nobody to analyze." "We don't care how people currently work. We're going to provide a new way of working with the system."

If your system is so new that nothing is related to it, there's a big chance that nobody will understand nor use the system. If users can't relate to your product, you probably don't have a market.

If you don't know the current way of working, how will you know whether the new way is more effective? Are you sure you will not create new problems with the system? On what are you going to base design decisions for the new system if you don't know the current way of working?

g. Assistant manager: "**Users don't know what they want**." "Users disagree about what they want. They all want something different."

User requirements analysis is not about asking users what system they want. User requirements analysis is about understanding users' current practices and the problems they encounter. Users are not interaction designers. They often don't know the possibilities and constraints of interactive systems. It's the job of an interaction architect to vision a solution that answers real user needs.

People are not all that different. There are patterns in their practices and needs. User requirements analysis is about recognizing these patterns.

h. Optimistic marketer: "You can't *really listen* to a representative sample of the target audience." "The future users are located all over the world. You can't possibly travel all over the world to do your analysis." "**Everybody will use our product**."

Involving a small sample of everybody is better than involving nobody at all. It's better to base your design on a thorough understanding of 10 users than on statistics about 10000 users. Statistical market research shows how often something happens in the target audience. It does not help us understand how or why it happens.

Now that more and more applications are used globally, international cultural differences can become apparent. However, patterns of users' behavior and needs will still appear and its important to take into account possible cultural differences when designing user experiences.

There aren't that many products nor interactive systems that are really used by "everybody". Targeting "everybody" with an interactive system is not easy, nor cheap. User requirements analysis can contribute to defining precise target audiences that are more likely to use your product than others.

i. Practical manager: "User requirements analysis sounds **like academic research**. We're not at the university. We're a company developing a website."

User requirements analysis uses some techniques derived from social science research. However, it is not like scientific research in its goals nor the way it is conducted. User requirements analysis is entirely focused on designing effective interactive systems. It does not pretend to answer research questions beyond that goal. In addition, timing and costs of the analysis are reduced to meet development demands.

j. Confident marketing manager: "**We have done marketing research** so we know what the user needs." Or: "We have done marketing research and didn't learn anything from that, so why do that exercise again?"

Statistical market research alone is not sufficient to design effective interactive systems. It shows how often something happens in the target audience. It does not help us understand how or why it happens.

k. Security manager: "Our project is **top secret**. We can't approach the future users."

Even a top secret product has to be optimized. Perhaps it's possible to find trustworthy users or user representatives.

l. Salesman: "Users **cannot spare time** from their jobs to participate in user requirements analysis."

They spend time now or later struggling with product that are not adapted to their needs.

m. Software engineer: "Instead of analyzing user requirements, we will **make a prototype** and test that with users."

Usability testing can reveal poor usefulness of a product. Only user requirements analysis shows how to fix it.

Adapted from Sim D'Hertefelt's article at
http://www.interactionarchitect.com/articles/article20000609b.htm

Elicitation techniques

✓ **Requirements gathering is not a specific technique that you apply to a project.**

✓ **Software requirements cannot be gathered completely.**

Survey/questionnaire

To study a large group of people and then analyze the results with statistics, questionnaires will give you some good numbers. Another result is gathering opinions and suggestions from a large number of users as if you were conducting an interview with them. Prepare your questions carefully with some interviews to help you determine the best types of questions. Test the questions on a small group so that the misunderstandings that are sure to infiltrate are minimized.

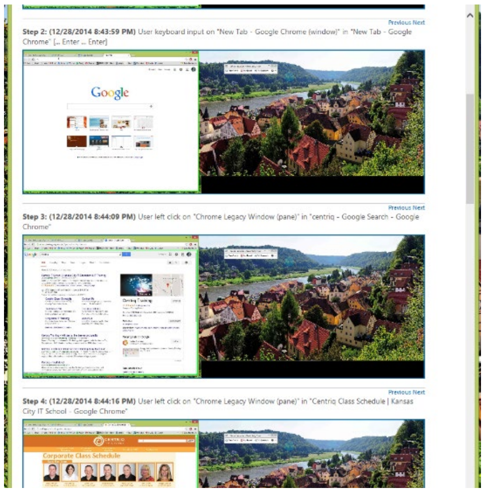
Observation

Because people are not the best judge of what they do and why they do it, observation can give clear insight into what things are happening instead of a filtered version. Observation can be in person, or with permission, through a video camera. Critical issues and tasks often escape recording so it's good for verification. When your computer goes down, it's rarely when someone else is watching, of course unless you are giving a demo.

The task demo is a type of observation where a specific task is requested for the observee to do. This is how most recipe books get organized when there is no previous recipe. You ask someone to make a dish for your recipe and while you watch, you record the process. Even a critical task can be handled this way as you propose the problem and the user handles it. A good use is to set up a mock emergency and watch how the user reacts even though it's not the same level of stress.

Users don't always identify the problems associated with a task. On a clumsy user interface, a user may be so used to doing things with workarounds that they just say that it's what gets the job done and they do it. Common mistakes, delays, frustration levels, and too many keystrokes can point to problems. A usability test is a task demonstration of a previous system to see how well a common performed or critical task is done.

A few great tools when dealing with people who are not local to you and can also be used to troubleshoot problems are the remote capture tools like WebEx or the screen activity recorders like the **Problem Step Recorder** in Windows (see Figure 4).



Step 2: (12/28/2014 8:43:59 PM) User keyboard input on "New Tab - Google Chrome (window)" in "New Tab - Google Chrome" [... Enter ... Enter]

Step 3: (12/28/2014 8:44:09 PM) User left click on "Chrome Legacy Window (pane)" in "centriq - Google Search - Google Chrome"

Step 4: (12/28/2014 8:44:16 PM) User left click on "Chrome Legacy Window (pane)" in "Centriq Class Schedule | Kansas City IT School - Google Chrome"

Figure 3 - Problem Step Recorder

Document analysis

✓ **The #3 best technique for elicitation**

Documents and screen shots of previous systems can give the analyst a look at the system after hours and allow for questions to be developed in other techniques of elicitation. Data can be verified in the database and other info gathered can be cross-checked.

Focus groups

Like brainstorming, focus groups, or sometimes called future workshops, has an idea development time which is concerned with identifying the problems. Then the people try to imagine the ideal way to do business and explain why this is the best way. Goals and requirements are expressed in basics here.

With many people participating, the high priority issues are gathered at the end and then prioritized. Each stakeholder group should have a high priority item.

A focus group meeting typically lasts one to five hours and the more important stakeholders are involved. The steps for holding a type of focus group are:

- **Invite the participants.** Between 6 and 18 people are good as long as all of the stakeholders are represented and the supplier's staff is only about a third of the total.
- **Open the meeting.** The theme is introduced, people get introduced and a sense of well-being is generated.
- **Bad experiences.** A discussion of horror stories from bad products or in doing a type of work are recorded through some process like computer projection, whiteboard or easel board. Stakeholders should dominate here.
- **Imagine the future.** Now the people get to dream without restrictions to see what they really want. Invite wild ideas and then try to get a picture of what drives them to want these things. Record all of these ideas and reasons. It isn't necessary to record the reflection of a problem that is resolved here such as "my computer crashes once a day" and the reflection of "I want my computer not to crash." Tell the members that each problem automatically counts as a wish which is to get rid of the problem.
- **List the issues.** A good session will produce around 40 issues. Categorize and combine the issues with the help or in front of the group. You can even do that while writing down the issues.
- **Prioritize the issues.** Get each group of stakeholders to select their top ten issues. You may even get more issues at this point. The issues will be ideas, problems, and requirements. Don't have them put them in order because people will remember which ones were their number one or two and then find out another group got their number one issue dealt with and they didn't. Keep conflict from happening at this point by just getting a general group identified.
- **Review the lists.** Finally, you can have a round-table discussion on the issues chosen by the other stakeholder groups. Continue to revise and combine issues here if you can.

Professional product developers will hold multiple sessions until the lists seem to look about the same. One or two sessions will be good enough for you.

The analysis of the effort here will be to select the issues to work with, coming up with solutions to difficult problems or impossible demands. All of the stakeholder groups will need to have some essential issue addressed. The final selection of the issues should be based on a blend of the perceived value of the stakeholders and the potential cost of solving the problems.

Individual / Group interview

✓ **The #1 best technique for elicitation**

This is one of the best ways to get the information about the current work environment but doesn't really get the goals and critical issues. You get plenty of opinions but other techniques should be used to verify information here. Interviewing is a very popular elicitation technique both to discuss and to use because it sends good messages to the stakeholders and the information gained is valuable.

The people that you interview should make up a representative sample from all the stakeholder groups. Don't always rely on a nominated interviewee which is usually a middle-level manager who has been shown to not be as good of source of information as the real end users. But remember that the nominee is politically the right person to start with.

Ask broad questions about the day-to-day work and problems. Focus on the critical tasks. When does the user work under stress? When is 100% accuracy important? Observation will not answer these types of questions and an interview is going to give that to you.

Why are these tasks carried out? This may be a very uncomfortable question for people if the believe they should know but don't. You can ask when people do these tasks instead of why and that will defuse some hostility.

The questions are general at first but with some identified critical issues, you can go for the detail on times and procedures. You should have a list of questions prepared before you start an interview. Leave room for answers on the question sheet and follow the lead of the interviewee rather than the order of the questions on the paper. Leave room for issues that you didn't have questions for.

Questions should be phrased in a context-free way to avoid prejudicing the interviewee. That just means that you don't try to steer them to a solution and you should try to ask open-ended questions like:

- Who is the user?
- Who is the customer?
- Are their needs different?
- Where else can a solution to this problem be found?

When interviewing groups, you will get more information from users in the same work area because they inspire each other to remember details. Use your facilitation skills to make sure that no one member of the group is dominating and you get everyone to talk about something.

Interview tips

✓ **Prepare for an interview**

Prepare your questions ahead of an interview and have them close by to a notebook which you will record the answers in. Perhaps you will use a questions sheet that has lots of room for answers on it. Check the questions on the question form so you make sure you are asking the right questions.

✓ **Research**

Research the background of the company / division / department and what is important about them that you wouldn't want to waste time on during the time that you have with the stakeholders if you could find out earlier. You may want to do a little confirmation during an interview.

✓ **Assign a scribe**

If possible, assign one analyst to take notes while another hold the meeting or interview. Keep track of issues and unresolved items during the meeting, get concurrence on notes taken during the meeting before the meeting ends. Reserve time to read notes back. It works better than distributing notes for comment after the meeting. People forget or try to insert agendas when no one else is there to object. Assign/get volunteers to resolve issues. Obtain user prioritization of the requirements. Provide everyone with written notes from the meeting within a couple of days after the meeting.

Have the scribe write the answers on paper. Typing on a laptop takes too much concentration.

✓ **Don't ask why it's there, ask what it helps**

After much work had been done on the database, a user said the system must use the same ID for two different, but related accounts. The DBA clutched her heart. The analyst asked what that did (why). What the user really wanted was to associate one account with the other. The DBA recovered: the true requirement had already been met.

Users may present a requirement as a solution. Find out why by asking "what" that solution provides or does. Asking "why" makes it sound as if their request must be justified to you.

✓ **Don't ignore useless detail**

Users resent being told the information they are providing is not needed. If the information is relevant to the topic, but too detailed for the current discussion, ask if you can return for that information later. Be sure to put in your notes that additional detail is available and from whom so you can come back for it without having to ask who has it.

✓ **Don’t repeat your questions**

Requesting the same information over and over indicates incompetence. Users will complain to their management about this because it keeps them from doing their real work, and it will come back to you. Establish and index a library of documents that have been provided. Put meeting notes and minutes in the library. Ideally this will be an electronic repository so it can be used by the entire team. Do not go to users for information until you have checked the repository. Keep track of hardcopy documents, also.

✓ **Use other techniques**

Remember, it isn't the users' job to tell you what kind of technology they need. If you want to know what kind of technical capabilities they might like, be prepared to give them examples.

Interview form

✓ **Part 1: Establishing Customer or User profile**

Name:

Company:

Industry:

Job title:

What are your key responsibilities?

What outputs do you produce?

For whom?

How is success measured?

Which problems interfere with your success?

What, if any, trends make your job easier or more difficult?

✓ **Part 2 – Assessing the Problem**

For which <application type> problems do you lack good solutions?

What are they? (keep asking “Anything else?”)

For each problem, ask:

- Why does this problem exist?
- How do you solve it now?
- How would you like to solve it?

✓ **Part 3 – Understanding the User Environment**

Who are the users?

What is their educational background?

What is their computer background?

Are users experienced with this type of application?

Which platforms are in use?

What are your plans for future platforms?

Are additional applications in use that are relevant to this application? If so, let's talk about them a bit.

What are your expectations for usability of the product?

What are your expectations for training time?

What kinds of user help do you need, e.g. hard copy, online documentation, etc.?

✓ Part 4 – Recap for Understanding

You have told me (list customer-described problems in your own words):

-
-
-
-
-

Does this adequately represent the problems you are having with your existing solution?

What, if any, other problems are you experiencing?

✓ Part 5 – Analyst's input on customer's problem

(To validate or invalidate assumptions. If it hasn't already been addressed, ask:)

Which, if any, problems are associated with:
(List any needs or additional problems you think should concern the customer or user)

-
-
-
-
-

For each suggested problem ask:

- Is this a real problem?
- What are the reasons for this problem?
- How do you currently solve the problem?
- How would you like to solve the problem?

- How would you rank solving these problems in comparison to others you've mentioned?

✓ **Part 6 – Assessing your solution (if applicable)**

(Summarize the key capabilities of your proposed solution.)

What if you could:

-
-
-

How would you rank the importance of these?

✓ **Part 7 – Prioritizing the opportunity**

Who in your organization needs this application?

How many of these types of users would use the application?

How would you value a successful solution?

✓ **Part 8 – Assessing reliability, performance, and support needs**

What are your expectations for reliability?

What are your expectations for performance?

Will you support the product, or will other support it?

Do you have special needs for support?

What about maintenance and service access?

What are the security requirements?

What are the installation and configuration requirements?

Are there special licensing requirements?

How will the software be distributed?

Are there labeling and packaging requirements?

✓ **Part 9 – Other requirements / constraints**

Are there any legal, regulatory, or environmental requirements or other standards that must be supported?

Can you think of any other requirements we should know about?

✓ **Part 10 – Wrap-up**

Are there any other questions I should be asking you?

If I need to ask follow-up questions, may I give you a call? Would you be willing to participate in a requirements review?

✓ **Part 11 – Analyst's summary**

After the interview, and while the data is still fresh in your mind, summarize the three highest priority needs or problems identified by this user / customer.

1.

2.

3.

Brainstorming

✓ **Quick lists of thoughts for analysts mostly.**

✓ **Don't criticize or discuss the thoughts of yourself or others until the creative part is over.**

Brainstorming takes a good facilitator to make it work and dissolves into a bunch of people complaining if left to the users conducting the meeting. Strong rules should be announced and users held to them during the session while the facilitator maintains a good sense of humor so people feel at ease.

The rules are few and simple:

- In a group of six to seven or less, people are allowed to voice a suggestion at any time. In a larger group or where people are strongly contrasted in desire to talk, you can **use a round-robin** of letting each person say something or saying "pass" until the suggestions die down to a minimum when you can open up the room to anyone speaking.
- **No criticism of ideas at all is allowed**. None. When someone suggests something, people want to have a discussion right then and there. Cut it off. In fact, a facilitator can make a really dumb or humorous suggestion and validate it by writing it down to show that no criticism is permitted.
- **All ideas are written down.** The absence of the idea being written down is criticism that the idea wasn't good enough. All ideas must be written down so that people trust the facilitator and have fun. People start enjoying the session more at the point where the facilitator has to write down good jokes. But because jokes free the mind to think outside of the box, you often come up with a very new way of thinking about a solution.
- **Judgment comes after the storm**. If there is a standard to which the ideas must conform, it's only used after the brainstorming is done to judge the worthiness of the idea. The standard may be black and white so an idea may stay or go. Or it may be a process of prioritization. But many people suggest disconnecting the brainstorming from the judgment of the ideas because ideas that look stupid after a joke is made about them may look brilliant in a few days.

Requirements workshops

The workshop is a general term to mean meeting where people produce a result. A domain workshop is where people meet to analyze the business process and the design workshop is usually where a user interface is agreed upon.

A domain workshop might use UML diagrams like the class diagram and use case diagram, or task on arrow and dataflow diagrams. The diagrams could be used as requirements if the project timeline is fast but would be turned into standardized requirements documents if the project is more lengthy and formal. It is important to always remember that the objective of a software development project is not to produce diagrams or documentation and if the initial diagrams are good enough to provide understanding of the system then you've completed your duty.

The design workshop could turn out to be one of the favorite types of meeting for a project but the results often are a disaster. You get a bunch of enthusiastic people at the start of the project who forget the goals and tasks that are important and design screens based on other bad systems, who then become committed to seeing it implemented and are unsupportive when it isn't.

The better kind of design workshop would take proposed designs without a promise of implementation and ask for improvements and suggestions to see what people's goals are. It's not the final design that you want but a visual tool to gain understanding for what a person thinks is important.

If you are involved with a design workshop that is more traditional, have the team check the user interface against the task descriptions and business goals every now and then. Also have the designs tested for usability.

People sometimes confuse the focus group with the workshop so the terms can be interchangeable.

XP Story telling

Typical software development projects specify requirements in formal documents which are referred to as "the requirements doc." The XP light methodology which is about as informal as possible is different for one primary reason: you rarely know all the requirements at the beginning of a project and they change frequently anyway as the team learns.

Static documents that feel like contracts don't fit into the XP scene very well. XP teams learn as they go. This doesn't mean you shouldn't think before you start to code. The realistic amount of the future that you can plan for is just a few days or so. Release planning will take over the planning after the code starts coming together.

A trivial system to practice with in the classroom always leads to success because either there's not enough detail to cause headaches or people act in the best interest of leaving by five by signing off on something that isn't going to bite them the next week.

The XP approach is radically different. It proposes that a team sit together in a room and talk about what the proposed system needs to do and how it needs to do it. Programmers or customers write these things down on note cards. Each note card says something like:

```
    As a <some user role>, I need the system to <functional
requirement> so that I can <business reason for the
requirement>.
```

The stack of note cards becomes the set of requirements for the system, as they are known right then. The cards don't have exhaustive detail on them. Rather, each card is a promise for a future conversation to flesh out the details. In the meantime, the card simply needs enough meat on it to give the programmers an idea of how much work it will take to build it, and to give the customers enough detail to write an acceptance test.

This form of requirements gathering requires a different skill from the business people who are the source of the requirements. Meeting skills are about saying things and being heard. People in requirements meetings who think while they talk get their requirements written down even though they aren't requirements but thinking about them. This works for them since they have participated and other people who don't think out loud are more doomed to be blamed when a critical requirement isn't known until near the end of the project. These people who are more quiet because they are thinking about what the requirement is and won't say much until they get it need a different style of requirements gathering. And the people who gush requirements at all levels of scope and priority need to have a better framework also. We really need requirements contributors who are good at telling stories. They tell them at the right level of detail, and they take responsibility for being the one to flesh them out when a pair of developers actually tries to code them.

XP practice is criticized as too informal. That could mean that the requirements documents most projects have should contain more formal language, requirements should have reference numbers, and the requirements gathering process should be more focused on arranged meetings with various stakeholders. The practical minded XP cynic is going to say that it doesn't make the code any better and wastes time producing outdated documents that aren't used much after that.

Even though the neat and orderly requirements documents exist, programmers often have to follow up on each requirement when it comes time to code something. They still had to ask the source of the requirements to tell us the story XP says they should tell them to begin with. Why not shorten the process and start with the story? That's what this practice is about. Story telling is a skill of simplicity and radical practicality and some people are better at it than others. It might be worth a try if people are honest and responsible and want something quick and informal.

The Agile / Scrum process

- Create partial set of user stories or use cases and place into the product backlog items (PBIs)
- Roll up the user stories into larger granular items called themes.
- Prioritize the stories by placing cards top to bottom under theme card and from left to right in workflow or usage sequence.
- Work out the details of the requirement continuously during development with stakeholders (progressive refinement)

- Complete requirement just in time and just enough when the teams need to start building the functionality for that requirement
- If a change from the original plan is necessary, use a change control process
- Re-prioritize requirements as necessary

External stakeholders

Checking with other companies may bring many new ideas to light such as any problems you have overlooked that should be addressed in the requirements. Visiting the site of the system will give users a realistic idea of how a new system will work and keep expectations manageable. If you aren't able or willing to share information with a competitor or another company you can at least find out the industry benchmarks for how internal processes are handled at the level that you are programming for.

Another way of getting to the companies that don't want to talk about their systems is to bypass them directly and go to their suppliers that you both use. They will want to encourage you to buy their products, yes, but they also know how others are using what you are trying to build.

Suppliers of software that have already built what you want also provide a rich ground for gaining requirements if you try to figure out what a feature is useful for and why they identify a certain quality of service as being important.

Reviews

Taking time to review the specs before signing of the contract is putting a tremendous amount of pressure at the wrong spot. In practice, the customer and supplier should be carefully reviewing the specifications along the way and the final one is just a last look instead of a significant event.

Reviewers should just report problems or defects and not debate any solutions in meetings. They then get together with the analysts and give the results to them.

Reviews don't work too well. The problems that reviewers find are not likely to cause a problem during development. The real problems that surface after implementation are going to not show themselves easily and the time spent in review may be spent better in putting more detail in the task description where the requirement is needed.

Pilot experiments

This really isn't so much a way to get requirements as it is to test the system. A small group of people at the company try the system out and get a buzz going for a success.

Elicitation / design techniques

Some techniques are useful as both an elicitation technique to do with a stakeholder to get their input and as a model to work from for design for analysts only. The danger in this is that the stakeholder may think they are designing the final product so care must be taken to introduce enough “noise” in the prototypes so they don’t think it is getting close to the final product.

Prototyping

✓ **The #2 best technique for elicitation**

A prototype is a rough or simplified version of the final version. Two types exist: those that are thrown away and the knowledge is used to produce the final version, and those that become the basis for the final version.

A high-level requirement prototype will show that the functionality is feasible and useful. The low-level requirement prototype will not really work but have a great user interface so that people get a sense of how the final program will act.

It allows the customer and developer to validate the data model. It helps the GUI designer support the user tasks. It validates the GUI at a very early stage. And it gives something concrete to the customer satisfying their need for talking about something real, encouraging feedback which often results in new requirements.

A prototype has names like trial or test case when it tests out the functionality of a part of the system.

A virtual window is another type of prototype but unlike a window from the application has no functions or menus. It’s a graphical picture of the data in the system.

The designing of the windows can be done on any type of media in any situation. With one person, you can sketch on paper a screen and make quick revisions. With several people you can try paper or use a whiteboard. Bring a digital camera to the session so you can record the whiteboard version. The versions can be cleaned up and made more formal if the time permits or the project has the time. Index your pages by some system and write that on the screen. Buttons on the screen can have a label on them that refers to an index number of another screen.

Exercise – Prototyping (ATM)

Prototype the ATM idle screen on the next page. Follow the requirements for what information should be on the screen. Use the entire box to represent the entire screen of the ATM.

Page name:

Requirements tools

A simple project doesn't need a tool, but a purchased software project might have up to 1,000 requirements and a Boeing 777 could overwhelm you with 300,000 requirements. That's when you need help with organizing, prioritizing, controlling, giving access to, and provided resources for all those requirements.

Tools that can be used include a word processor, a spreadsheet, or a specialized tool that manages requirements. Most people are more comfortable with the process-empty Microsoft Suite tools because what they don't see doesn't get in the way. But the requirements management tools do add some value to the process and learning the management process first will help you use those tools better. Tracking and management of requirements especially large numbers of them, is made much easier by the requirements shelfware.

Software

Requirements management

Major requirements tools vendors include

- **IBM Rational** - RequisitePro
- **Telelogic** - Doors/Enterprise Requirements Suite (ERS)
- **Borland** Software Corp. - StarTeam and CaliberRM (via its purchase of StarBase)
- **Computer Associates** (CA) International Inc. through a partnership with Integrated Chipware Inc. - RTM Workshop
- Seapine Software - TestTrack RM

Requirements modeling

Graphic models can be put together at the very basic level just using a piece of paper and a pen, and then recording the image with a digital camera so that it can be stored in the requirements repository. Variations on this include using a larger piece of paper like an easel board, or an erasable "piece of paper" like a chalkboard or a whiteboard. Trying to be creative and spontaneous doesn't work well with a computer.

The next level of working with tools is to go with the simple built-in graphics of Microsoft Word. You can do all of the UML diagrams and other graphic aids. Do you really want to? Not unless you are on a shoestring budget but it is convenient and possible.

The next level up is Visio and because it is a Microsoft product you can embed the graphic in Word and then edit it directly in Word although you may find the interface clumsy. But Visio by itself presents the best of both worlds of simplicity and pre-made templates. There are lots of pre-designed symbols arranged by types of diagrams and you can find more on the web.

UML templates in addition to the standard UML Visio template that are sometimes useful can be found at:

- http://www.phruby.com/stencildownload.html
- http://www.holub.com/goodies/uml_visio/

For creating more advanced diagrams that comply with UML or another diagram type, you get more expensive. For instance, IBM Rational Rose is a great tool for organizing all of your UML diagrams and giving you all sorts of variations that the UML standard calls for. But most people use it like it was Visio and never get the full use out of it. In fact, the more you know about what you want, the more the options start getting in your way.

For other types of drawings, you can go with a vector based graphics tool, but now you are talking about a long ramp-up time to get used to a completely different environment. Before you start thinking that Adobe Illustrator or Fireworks is a good choice because you can get it from your co-worker, browse through a book at the bookstore to see what you are in store for. The same goes for the bit-mapped graphics programs like Adobe Photoshop and the knock-offs. Unless you have experience with these, it's best to stick with the more basic types and learn them well.

Selection

Selection of a requirement management tool should be based on your project or enterprise needs and thorough investigation of candidate tools. Get a demo copy and run through the menus looking at what the features are because additional tools may be required to achieve full traceability, particularly to test. Tests are part of the software product and should be maintained along with the software, so tracing to test is important. Try a tool for a month and see if it works for you.

A requirements management tool will not solve problems caused by requirements of poor quality. No matter how great the tools, poor requirements are difficult to manage and can lead to disputes with the customer.

The International Council on Systems Engineering (INCOSE) publishes a comparison of features of many RM tools, which INCOSE updates periodically at

http://www.paper-review.com/tools/rms/read.php.

The site also has a good discussion of the minimum capabilities of RM tools.

In anything at all, perfection has been attained not when there is no longer anything to add, but when there is no longer anything to take away, when a body has been stripped down to its nakedness. - Antoine de Saint-Exupéry, Wind, Sand, and Stars

Technical writing

Writing is a craft

The best technical document is one that you don't notice is the best document that you've ever read. It's easy to see when there are errors but as in writing any other type of document, it is hard to develop that skill that allows people to see exactly what you meant and not slow down because the writing style is hard to understand.

The more you write technical documents and get critiques, the better you will be. And the more you read technical document and critique them, the better you will be. So, start writing and reading other peoples' requirements documents. Share your ability with others in similar positions.

Write for humans

Are you planning to take that 87 page template and add a few paragraphs that no one will read just so you can meet the company's policy for having a requirements document? The bigger the document, the less chance it is that it will be read. So find out what the simplest style you can use first and then increase the substance of the documentation by need rather than by template. Methodologies like RUP are too big for anyone to figure out that they recommend customizing the documents. Tools that support these methodologies only make the problem worse by introducing a feature buffet that doesn't give you a sense of how to simplify your life by using the most important parts. Conforming to the standards doesn't always mean to implement all of the standards.

Here are some questions to keep your documents realistic and readable:

- Is there a way to express this in an easier way?
- Is there too much information here in this section? Should I provide a roadmap, break it up into pieces, or categorize the parts better?
- What are the important parts and the less important parts here? Should I note that somehow?
- Is this information too vague or complex?. Do I need graphics? Do I need a common theme or principle?
- What kind of misinterpretations could my users make?
- What benefits do my readers get out of this section? Who cares about this? Will my project peer think this is a waste of time?
- How would you characterize the style of writing? Formal, friendly, stuffy, pompous, scatter-brained, rambling, simple, direct, impenetrable, etc?
- How can I keep it from being boring so that anyone who picks it up will want to read it? What makes it enjoyable?

Create lists

People like lists to check off because most of the processes in business are goal oriented. It helps to let a person focus on their job and not have to think if you did your job. You can encourage this kind of confidence by asserting your preferences in list form. They stand out from the text.

Form follows content

Don’t let the template dictate the style of writing that you do. All projects do not need the same type of documentation. If a photo of a diagram suffices instead of three pages of text then use it.

Organize details

If you use a template for your thinking about what you have left out, you will not think of all the things that you need. It’s better to design the organization and find the template that you can customize to your needs.

Reinforcement

To you the writer of the document, it may seem silly to put the same information into text and then put it into a diagram. But to the person reading the document who thinks visually, it’s easier to grasp the concept in the diagram and then attach the words to it in the text producing understanding. There are text motivated people also that need help in working with pictures so that they can talk about the PowerPoint slides during the management meetings.

Use relevant text

Don’t use text that you might think would only apply to certain people and others could skip over. This tells the reader of the text that they can skim and pick and choose what they want. This kind of text is called decoy text.

This kind of text which describes the text that comes next causes reader blackouts. It’s the type of redundant info that says “The text that follows is a description of the system following the standards of the enterprise as it is expressed in the enterprise standards documentation.”

More decoy text could be generalities like “This stuff is really important stuff and it should be important to you because it has a high importance.” Or maybe “All text should fit inside the window and be readable.”

A requirement can be re-expressed and have a piling-on foul called on it because you keep adding more and more weight to the definition when it isn’t needed. It is what people do when they talk about an ATM Machine or in requirements statements it might be “the system shall have a user interface that allows a customer to interact with the system in a user-friendly manner.” We do it without much thought so the technical writer has to be on guard for this.

If you start putting irrelevant documents in the requirements documents even though they are project related, it’s not going to be read. Things that have to do with the scope of work rather than the scope of the system tend to frustrate system developers. A requirement that sticks in the throat easily is “the user manual will be the result of the requirements documentation and the design document.” That’s really the project process and shouldn’t be implemented in the final system software so it shouldn’t be in the requirements.

Schedules don’t make sense in the requirements since they change and they are a part of the project timeline. Acceptance criteria are created after the requirements and therefore should not be a part of the requirements documentation. Approval signatures are annoying since it’s not necessary to use them to design the system. They can be kept in a separate but related file.

A traceability matrix is really a project management artifact for reviewers and makes trouble for the reader of the requirements. Feedback forms are fluff in the documentation and should be handled separately.

Completeness counts

If you are going to define a concept or a field or anything else, make sure to completely define it and then reference it throughout the document. If you want to make a hyperlink or use page references, please do but don’t duplicate your definitions in case you decide to update one and the other one becomes confusing to the reader. Also present as complete of concept as possible with references so that you don’t have to piece together the parts.

Use business terminology

The terminology of the document (except the part that is marked as design requirements) should reflect the business terminology of the system as it exists in the business now without a system to handle it. That means that there will be no computerspeak terminology to foul up the understanding and make the document seem more business-like.

How would you rephrase an “airplane reservation data validation function” so that it meant what it was before the system was introduced? Maybe a “reservation check”? Whatever it was, it has to be understood by a mid-level manager who has been working with the domain for many years and has no idea what software is.

Use consistent terminology

It’s good to stay with the term that is most used and decide early on what the term will be. Just because the programmers like the word “process” doesn’t mean that the industry term “technique” should be replaced by that. There is a rule called the mapmakers’ rule that has several parts to it. One part says that you should always name the geographic feature by the way that the natives call it. Don’t go renaming things because people will be confused. Even a large project that brings together several departments may have terminology problems and that should be agreed upon early also.

Assume a friendly reader

As you read through a document, you pick up a feeling that the writer is comfortable with you understanding the material because they have either checked out what audience they are writing to or they believe that they have been able to educate the reader enough to where they won't make outlandish assumptions about the statements made. Legal documents tend not to feel good because the writer assumes that the reader is looking for ways to misunderstand the material at any turn for their advantage.

Some technical minded folks like the challenge of finding flaws in the documents that have to do with friendly omissions. Maybe you said "all warehouse items" but didn't mean the trash in the bin. Maybe you said "potentially harmful software attacks" but didn't want to include Microsoft Windows there. It's a problem with the reader and not the document because the more rigid and legalistic the document becomes, the more the reader picks up on the hostility and tries to respond with a critical eye or gets bored.

Requirements documentation

A report structure

The report structure involves interviews, document studies, workshops and brainstorms. Analysis is done and the document follows the following outline:

1. Introductory parts – includes business goals
2. System limits – what to deliver, the interfaces, context diagrams
3. Data requirements – data models, data dictionary
4. Functional requirements – function lists, feature requirements, process descriptions
5. Quality requirements

A faster approach focuses on describing the user tasks in co-operation with the expert users in such a format as a use case.

If you only have basic information needs, the business goals have already been decided. In a more extended version, you would do interviews and brainstorming to define business goals which would also be a part of identifying business process re-engineering (BPR) tasks or other high-level tasks.

A design-level requirements phase that produces the requirements for the complex interfaces most likely with screen prototypes can be added. A requirement can be written that specifies the use of a prototype screen after validation through usability tests. Other interfaces specifications might include communication formats, protocols, etc.

As the designers dig in and get more detail, they will find that there have been omissions in the medium-level requirements. To allow for tracking and consistency, the designers should update the medium-level requirements. This type of approach lets the consultant doing the system build a bill in two parts also. The first part can be done on a time and material basis and the second part can be done as a fixed price once the first part has been agreed on.

Checks are made to verify that the parts are consistent and complete, and then reviews are made by reviewers and stakeholders.

The weakness of this is that the functional requirements can be hard to specify through user tasks. You can of course specify them some other way.

Types of projects that do not work well with this type of approach are the COTS acquisition where customers want certain features and domain aspects have been resolved, sub-contracting for the same reasons, and for product development except as the first step in a two-step approach.

A typical requirements document

The requirements document could contain the following types of categories and information of contents:

Introduction

Customer and sponsor

Background and reason for getting the system

Supplier type or project type (in-house, tender, etc.)

System goals

Business goals – the success criteria of the system which is why the customer wanted to spend money in the first place is very important info to help guide the requirements gatherers and writers.

Evidence of trace from goals to requirements

Data requirements

Description of data in database or other persistent data. in verbal form and possibly as diagrams.

Description of input / output data formats

Communication states (temporary data)

Initialization of all data and states

Functional requirements

✓ **Traditional**

System boundaries and interfaces

Domain-level events on each interface

Domain-level requirements, e.g. tasks

Product level event and functions on each interface

Design-level requirements, e.g. a prototype of user interface or communication protocol.

Specification of all non-trivial functions, including their input and output.

✓ **Use cases**

Name and index number

Actors

Pre-conditions

Flow of events

Post-conditions

Alternative flows

Handling of special cases

Cases that stress the system to its limits

Special events such as power failure, hardware malfunctions, installation of new components

Use cases or similar artifacts that deal with special business events or alternative flows that deal with things going wrong.

Quality requirements

Performance – efficiency, capacity, and accuracy

Usability – ease of learning, task efficiency, ease of remembering, understandability, and satisfaction.

Maintainability – corrective, preventative, and refactoring

Safety and security – handling of physical disturbances and hacking attempts.

Reliability – known errors and estimated number of errors

Availability – frequency of breakdowns, time available for normal operations, recovery

Fault tolerance – handling of erroneous input and unexpected events

Portability – moving the product to other platforms

Reusability – reusing the components in other products and configuration management plan

Interoperability – co-operating with other products

Ease of installation

any other quality factors you might want to include

Other deliverables

Documentation

Training

Installation

Data conversion

System operation

Support and maintenance

Glossary

This can be a separate document or an appendix to the requirements documentation that includes the definition of domain terms and product terms. One of the better ways of handling confusing terms in a document is by linking to the glossary item with the hyperlink feature in Word. You can also embed a reference to that document if you create one separately.

IEEE Standard 830-1998

There are many different ways or combining the requirements and extra information into a specification. The best known standard is the IEEE Recommended Practice for Software Requirements Specification (IEEE 830). It suggests a standard structure of the introductory parts and several alternative ways of structuring the detailed requirements according to the interfaces, according to user classes, according to application domain objects, etc. It states explicitly that the requirements surrounding the business aspects of building the system or contractual agreements should not be part of the system requirements and should be part of another document.

You will find most likely that the data and the functional requirements will be the bulk of your requirements. Interestingly enough, most programmers get by with quite a bit less specification on these because they have the skills to either find the requirement clarification from a stakeholder or make a good guess about what it should be. The other side that makes you think is isn't necessary to bulk up the document here is that analysts complain often that they spend too much time writing them and wonder if they are right. A good substitute for spending time in a better area would be writing quality requirements well.

This standard outlines the tasks into an introduction of business goals, the scope of the project modeled in a context diagram or a system-level use case diagram, the data requirements with a data model and descriptions, the tasks (use cases) or features (functions), and quality requirements.

The full text of the IEEE Standard 830-1998 is available at:

http://standards.ieee.org/reading/ieee/std/se/830-1998.pdf

if you are a member. Trial memberships are free and allow you to access the document. If you sign in incorrectly several times you will get a screen that gives you instructions on joining as a trial member.

Additional report sections

Other types of factors that can be included in the report can be expressed in different sections that are appropriate to the type of project that you are working on. Here are some categories to work with:

Document information

This is meta-data about the document which can include information such as the table of contents, list of related documents, typographical conventions, software versions that the document applies to, date that the document was last modified, a change log, document preparers, and an index for large documents.

Overview

You might want to give a bird's eye view of the requirements document by telling people how the parts all fit together and where they might find the most relevant sections depending on who they are.

Expectations

These are the results of the software that the customer had in mind when he paid for it. Maybe the expectation is that it will work better than the last piece of discount software they had purchased in 1988 from a vendor in Iceland.

Preferences

The criteria for choosing among different designs that meet the requirements will be a big help to designers who want to give you the best software they can. If your pack of programmers like to add Microsoft specific features and the users hate that, you can help out here by mentioning that preference. It's a place for those vague desires that won't have a place in the rigid requirements.

Invariants

Invariants are conditions that are never to change or at least never violated between events. There are the requirements that state conditions that the system is supposed to maintain even as other operations are taking place such as life support conditions on a spacecraft. And there are requirements that use redundancy to assert conditions that must be true even though other requirements may combine to violate them. This might be the case where you assert that the accounting books must balance as a result of any transaction in a rule like "assets plus liabilities equals equity."

Platform

You may have a particular machine in mind for the system and the software and hardware can easily affect what design is made early in the project.

Global characteristics

This not so standard term is about the properties that the system as a whole is to process as opposed to the separate requirement statements that usually have a nearly on-to-one mapping to segments of program code. This section can also be made into quality requirements as it has to deal with availability, reliability, safety, and security.

Design constraints

These are statements that deliberately violate the separation of the requirements writers and the designers. Maybe the customer has an enterprise-wide style of programming that uses variables in a certain format or case style. They are paying for it so it becomes a requirement for this special section.

Likely changes

Changes that you expect in the future are good to know for the software developers because they can make your life easier or make your life worth moving to a dude ranch in Wyoming.

Interim report types

Some examples of requirements management reports are below. When identifying requirements management information needs for your project, provide an example of suggested reports. This will help you get better feedback on what kinds on information to keep in the requirements repository. Reports are based on the information associated with the requirements.

Backward trace: a traceability report showing the origin of an item. This can be used to determine if unnecessary items are being created (gold-plating) that are not warranted by the requirements or to discover the reason for an item.

Compliance matrix: a report showing the requirements that have been satisfied by a work product or activity.

Forward trace: a traceability report showing items that would be affected by a requirements change, such as the impact on a design, documents, and/or software.

Priority: a report of requirements by priority. This report may be used to determine what requirements must be implemented before others. It is also useful when time to build is inadequate or funding is limited.

Qualification method: a report of requirements by qualification method (inspection, analysis, demonstration, test). This report provides requirements that testers or others must address.

Requirements validation matrix: a report of requirements and the test cases associated with them. The report can be used to track test case development and approval and can be included in the master test plan.

Risk: a report of requirements by risk level. This report may be used to determine what requirements are most critical to test completely or to assist in planning when time to test is inadequate.

Unsatisfied/unallocated requirements: a report of unaddressed or unassigned requirements. This report may accompany a compliance matrix and is a quality check to ensure no requirement has been dropped.

Alternate report structure

User Requirements Outline / Concept of Operation (IEEE STD 1362-1998)

1. Introduction
   1.1. Purpose and background
   1.2. Overview of business and user needs
   1.3. Document overview and conventions
   1.4. References
2. Current system or situation
   2.1. Background, objectives, and scope of the current system or situation
   2.2. Current system or business processes
   2.3. People, organizations, and locations
3. Justification for the new system
   3.1. Rationale for the new system
   3.2. Overview of the system and impacted business processes
   3.3. Affected people, roles, and organizations
   3.4. Priorities and scope of the change
   3.5. Impact on operations, the organization, people, and support
   3.6. Impact on policies, regulations, and business rules
4. New functionality and quality attributes
   4.1. User and user profiles
   4.2. New or changes user capabilities (also see App. D)
   4.3. Impact of the new capabilities on existing user processes and systems
   4.4. Interfaces with other systems (hardware and software)
   4.5. User support environment and user documentation
5. Evaluation of the proposed system
   5.1. Advantages, disadvantages, and limitations
   5.2. Business and organization change management plan
   5.3. Operational issues
   5.4. Alternatives considered

Appendices

A. Glossary of terms
B. Data dictionary
C. Context diagram
D. Use cases and scenarios

Add in system interactions, external interfaces, and other performance capabilities for large complex systems in a system requirements specification (SRS) document. (IEEE STD 1233)

Design tips

Group logically

Information that is related to each other should be placed closer than information unrelated to it. Divisions of information should follow major categorical logical changes. In requirements the types of statements that get grouped together are those that have to do with the clause of the statement. For instance, take a look at the statements:

```
The clerk checks out the guest from the hotel room.
The manager enters a service ticket on a hotel room.
The guest uses an electronic card to enter a hotel room.
```

These are all related through the clause "xxx a hotel room" of the predicate which means that the hotel room is the organizing concept and might be grouped in something called "hotel room activities." This is a data centric grouping. You could also use a grouping of the subject (noun) of the phrase to show the scope of the roles involved where all of the clerks functions were grouped together. That would be the "clerk functions."

Sequence

The best sequence for presenting ordered material is where information has no dependencies on material after it. The logical sequence of the content flows and builds on previous material. So, any term that is introduced is defined there or referred to in the glossary before it is used again.

Emphasis

Giving visual statements importance through a visual means taxes most people's artistic skills. Here are some tips:

- Use a graphic to emphasize content.
- The first item is automatically emphasized.
- Repeating an item makes it more important.
- Placing lots of white space around content emphasizes it.
- Bullets help emphasize
- Long sections of text seem more important than short sections.
- Repeating something in another format helps emphasize (text with a picture, headline with a special bullet, etc.)
- Using contrast sparingly emphasizes (special colored text, colored background, much larger font, bolded text)
- Italics are a weak way to create emphasis. Avoid them.

Design principles

In *The Non Designer's Design Book*, Robin Williams suggests four principles for graphic design.

Contrast

The visual differences between things -- often "the most important visual attraction on a page". Williams' more lengthy definition of this rule is:

Contrast is created when two elements are different. If the two elements are sort of different, but not really, then you don't have contrast, you have conflict. That's the key: the principle of contrast states that if two items are not exactly the same, then make them different. Really different.

Colors, sizes, typefaces, alignment, text direction, and can be use to create contrast in a layout.

Repetition

Repeat graphic elements and colors to provide a sense of unity for both single and multiple page works. "Repetition can be thought of as 'consistency' ... a conscious effort to unify all parts of a design"

The inverse of repetition, difference is also useful strategically; making one page or element wildly different from others can make it stand out.

Alignment

Alignment isn't just picking center, left, right, or justified – it's the art of placing elements in a layout. Williams is crystal clear:

The principle of alignment states that nothing should be placed on the page arbitrarily. Every item should have a visual connection with something else on the page."

As far as text alignments is concerned: pick as few as possible for any one layout. Use differing alignments (a right-aligned image on a page where the text is left-aligned) carefully; they can be disruptive. Avoid "justified" alignment, which rarely looks good unless composed by expensive typesetting software.

Centering everything is a very common design choice. However, as Williams notes, centering often looks arbitrary (and is beat to death). Resist the urge.

Proximity

Put like elements together, and separate unlike elements. This reduces the number of visual units on a page, making the task of design simpler:

The principle of proximity states that you group related items together, move them physically close to each other, so the related items are seen as one cohesive group rather than a bunch of unrelated bits.

Subheads should go with the things which they announce. Dividing elements into categories and classifications is often a part of proximate layout.

Robin does realize these principles can be connected by a "memorable, but very inappropriate," acronym. She describes them in the order "proximity, alignment, repetition, contrast," which may be the best order in which to consider them.

Process writing

Type

business

automated with constraints

blended

Requirement quality criteria

Requirements of good quality:

- **decrease cost and schedule**: no effort is spent during design and implementation trying to figure out what the requirements are
- **increase product quality**: good requirements cause the right product to be delivered and no rescoping to meet schedule or cost constraints
- **decrease maintenance effort**: traceability decreases the effort to identify where changes are required, especially as knowledgeable personnel leave
- **create fewer disputes with the customer/client**: no ambiguity causes less differences in expectations and contractual issues
- are a major cause of project success.

There never seems to be enough time to do it right, but time is found to do it over—if funding is available!

The criteria for writing requirements well are a generally accepted standard but seen in different variants. The IEEE Standard 830-1998 lists consistent, correct, complete, unambiguous, ranked, modifiable, testable, and traceable.

Testable or verifiable

✓ **The #1 trait of a good requirement is that it can be tested.**

You should be able to prove that object of the requirement satisfies the requirement. Most likely a module of code will be the proof of a requirement. Untestable requirements can lead to disputes with the client. Quality requirements most often violate this principle. You would not like a court of law to do your decision making process for whether a requirement was met or not.

Complete

✓ **The #2 trait of a good requirement is that it is part of a group of other requirements that creates a total higher level requirement.**

All necessary requirements are included. But you don't want so many trivial requirements that bloat the documentation and make it hard to read and hard to manage. You may think that user interfaces are trivial requirements but they can be the most common source of problems and shouldn't be ignored. Completeness means there is enough information to proceed to the next stage or phase of work without risking a serious amount of rework. An example of an incomplete business rule:

```
The system shall validate charges over $100.

The system shall accept charges under $100.
```

What about charges of $100? Correcting the first requirement to state: "The system shall validate charges of $100 or more" is one way to address the situation.

Ensuring completeness is hard work, although models can help.

It takes some time to put together a list of tasks that the system should implement in a report. Some of the questions that you can use to investigate whether you have covered all the bases are:

- **Have all the domain events been covered?** These are requests for service from other systems, triggers from users of our system due to business events.
- **Are critical tasks covered?** You want to make sure that the time-consuming, frequent, difficult, or those performed under stress are supported well. Most of the time, checking in a single guest to a hotel will be enough but having a bulk check-in for that busload of tourists can really make a difference.
- **Does the new system cover the old system's tasks?** You want a system that doesn't disappoint because the old features aren't included in the new system and are expected.
- **What kinds of miscellaneous activities should be supported?** Some tasks are almost free-flowing like browsing the web or playing a game. Try to capture as much detail as possible without overloading your use case with variants.
- Did you use a CRUD check? see below.

Measurable or specific

✓ **Without a quantity and a unit, requirements become subjective and often untestable.**

Pick a metric style. Styles are based on the units that you plan to measure by whether time, money, or other criteria. The requirement is written based on that measure and follows the syntax of verifiable quantity and a verifiable result such as

```
Every ### minutes, the system will be able to process abc operations on
### customer data records.
```

Select the target. The variables in the statement above have to be nailed down by talking to the right stakeholders.

```
Every 10 minutes, the system will be able to process data cleaning
operations on 5 million customer data records.
```

Consistent

All parts match in the sense that they don't conflict. It's an internal thing. A common problem is when one requirement is listed twice with different quantities for a metric. Another is when you get a contradiction or a completely unrealistic demand. In reality, the only completely consistent model is the final version so anything before that is just a good guess.

A case of conflicting subjects would be a requirement for blue backgrounds and one for red. An example of a conceptual conflict would be a requirement for accessibility and one requiring red and green on the GUI.

A glossary helps achieve consistent terminology. Even commonly understood terms can create headaches later if they are undefined because they can change meaning over time.

Acronyms common to the users' business can be used in requirements. Define them in the glossary for newcomers to the project.

Use the RM tool to search on key words and phrases to locate consistency problems. Key word and phrase searches may not locate conceptual conflicts. Diagrams can be also be used to locate some inconsistencies.

Correct

The requirement is free of unnecessary design. Each requirement reflects a need that is within scope. Maybe it's just a whim by someone sneaking it in. Models can help ensure requirements are within scope.

Unambiguous

All parties agree on the meaning of the requirement. A glossary is a good accompanying document to keep up that can help. Explaining the purpose of each requirement is a more wordy but excellent way of guarding against misunderstanding.

Ranked

Requirements should be ranked by importance and stability which will gauge the priority and expected changes per requirement. But not everyone's priorities match. Some people's priorities are all number one. Some requirements should be bundled into one priority because without the related requirements, there would not be a priority.

The stability ranking is supposed to identify functions that should be easy to modify but doesn't work in an OO environment and isn't always clear until the end. Think about replacing this with a maintenance requirement.

Modifiable

This means that the requirements and associated information can be changed. Requirements should be easy to change to maintain consistency. This is why you number or index each requirement and they refer to each other instead of repeating the requirement. Of course, no one likes reading "… in relation to DOG129 and CAT133 and without conflicting MOM990" but it keeps it consistent.

And when sent digitally, suppliers can search for the set of requirements which they can fulfill either as a standard or extension or not at all. The method of storing requirements affects modifiability. For example, requirements in a word processor may be more difficult to modify than in a requirements management tool. But, for a very small project, the cost and learning curve for the RM tool may make the word processor a better choice.

Consistency affects modifiability. Templates and glossaries for requirements make global changes possible. Templates should exclude information that will be associated with the requirement.

For example,

```
The Calculation Module shall determine the gross amount of annuity
```

must be reworded if the name "Calculation Module" is changed or if annuity calculating is moved to another module. This change will be in addition to updates to the information or linkages associating the requirement with the Calculation Module in the requirements repository.

Implementing global name changes increases the potential for injecting errors. Design requirements standards to minimize the number of changes needed to maintain requirements.

Traceable

Each product can be associated with its originator and vice versa. You can track it to goals and purposes and to the designs and the parts of the code. This is both backwards and forwards tracing. Compound requirements are difficult to trace and may cause the product to fail testing.

Feasible

The requirement is technically and operationally possible to do within existing constraints. The constraints are typically time and money. For example, we can send a person to Mars, but not within a year and under a thousand dollars. An example of a technically infeasible requirement is time travel.

An infeasible requirement does not always cease to be a requirement. Thinking these requirements cease to be requirements is a contractor/project's perspective. It is true they are not requirements for the project. But, if the client is enlightened and will use the repository for product maintenance, the requirement may be retained for future implementation. Typically, however, the requirements repository (if there is one at all) is scrapped after completion of the contract because it was not specified as a deliverable.

Independent

The requirement does not rely on another requirement to be fully understood.

Requirements that need higher level requirements to be understood are not independent. Parent or higher level requirements rely on their lower level or children to be fully defined. In testing, a parent is not satisfied until all its children are met.

Why retain the parents? These may be source requirements that must be retained. Also, using them to structure the requirements that satisfy them at a lower level improves understandability. For example, "user friendly" can be used to assign, talk about, or locate the group of lower level requirements defining "user friendly" for that particular project.

Necessary

The requirement should be necessary. Here are some unnecessary requirements:

```
The system shall be designed to accommodate the capabilities in the
system requirements specification document.
```

The real object of this requirement is project processes, and it has no place in product requirements. Every requirement, product, and test would have to be traced to this requirement resulting in a requirements repository cluttered with meaningless traceability. The requirement is also vague. What is actually meant by "accommodate"?

A traceability matrix is used to verify that a product includes planned requirements. The correction is to delete this requirement from the requirements document. A requirement to deliver a compliance matrix with each product can be included in the contract or process standard.

```
The system shall be accepted if it passes 25 test cases.
```

Unless the term is defined for the project, "test cases" has no standard meaning, so the requirement is meaningless. In addition, this requirement attempts to set a contractual standard for the client through the backdoor. If the requirements document were attached as part of the statement of work (contract) for the next phase of work, it could limit the client's ability to fully test the product before accepting delivery and would certainly lead to disputes. The correction is to delete the requirement and create a test plan.

Non-redundant

Duplicate requirements increase maintenance. Every time a requirement changes, its duplicate(s) also must be updated.

Duplicates increase the potential for injecting errors. For example, redundant requirements may be overlooked and not updated.

It is OK to associate more than one product with the same requirement in the traceability matrix. If there are duplicate requirements, analyze them to see if they are really the same. If they are different, rewrite them to make their differences clear. If requirements are related, but located in separate areas of the repository or document, they can be cross-referenced.

Terse

A terse requirement has no unnecessary verbiage or information. Which one do you like better?

```
In view of the need to achieve high understandability and in order to
increase the amount of comprehensibility, in a number of cases, the
```

```
removal of multiple words, specifically those that are not central to
the understanding of the requirement, is a necessary and prudent action
owing to the fact that a requirement with fewer words can be understood
in a somewhat more satisfactory manner and in a shorter timeframe than
in the case of a requirement with more words and the same meaning which
would take a lengthier time span to gain better clarity.
```

or

```
Terse requirements are more easily understood than long ones.
```

To improve clarity, remove extraneous words.

Understandable

Understandable requirements are organized in a manner that facilitates overall review between the customer and the developer. The requirements may be detailed, complete, accurate and unambiguous but just hard to understand.

Requirements management tools can facilitate multiple ways of organizing requirements so that reports can be tailored to different audiences. Some ways to organize requirements include:

Organize requirements by their major business entity that they refer to in the clause that should be following the main object-verb part of the requirement. e.g.

`The admin shall report on delinquent accounts` **`of the accounts receivable.`**

`The manager shall check current status of an account` **`in the accounts receivable.`**

Separate non-technical requirements from requirements directed to the actual product. Non-technical requirements, for example, might specify development team processes or how the team must work.

User requirements can be organized by business process or scenario. This allows the subject matter expert to see if there is a hole in the requirements. This is the major thought behind use case detail.

Separate functional from non-functional requirements. Non-functional requirements are those that relate to areas such as performance, reliability, availability, accuracy, and capacity.

Organize requirements by level, which determines their impact on the system.

Understandable requirements are grammatically correct and written in a style that facilitates review by the affected parties.

Standards help ensure all requirements have the same style and increases comprehension. Various organizations, such as IEEE, have created standards for writing requirements that can be leveraged for your project. Remember: while these standards attempt to address many situations, additional details may be needed for your project.

Requirements statements

Components

At the least, you should have the requirement basics which should include:

- ID and date
- Verifiable requirement statement
- Originator
- Priority

Then add enough information to the requirement to be thoroughly understood and managed.

- Cross-references rather than duplication of info. But don’t exaggerate and include a little duplication for clarity’s sake.
  - Business rules
  - Data dictionary
  - Screen designs
- Level
- Tracing to other levels
- Purpose of each requirement, e.g. relating it to some domain activity or business goal.
- Examples of ways to meet requirements. Should be used cautiously. The purpose is to understand what the user has in mind. Avoid analysis which becomes a design discussion.
- Explanation of diagrams, etc. in plain text. Users rarely understand technical diagrams.
- Stability for each requirement
- Index

Prioritization

Prioritization can be done at either the very beginning of the project where the rating is used to determine project scope and schedule or it can be done after the project has been deployed and a bug has been found. But at any stage, a requirement or a loss of the service are both judged by the same combination of impact and urgency.

Whether or not something is a requirement is independent of a project and its timeframe. A requirement may not be implemented during a project. In the instance where users have revised a requirement to require larger monitors, the users can get by with their old monitors, even though they can't work as quickly. Because it must prioritize its spending, the company will not purchase the new monitors during the project, even though the users still want and need them. Consequently, it is not a requirement that must be implemented by the developer the company hired, but is a requirement for the company and may be implemented sometime after the project.

One problem with requirements is that there are always too many of them. So there has to be some way of choosing which ones will be implemented in which versions of the product. The requirements need to be put into an order of desirability, in other words they need to be prioritized. Decisions about prioritization are complex because they involve many different factors and these factors are often in conflict with each other. Also, as there are many stakeholders with different goals, it is difficult to reach agreement about priorities.

The information concerning prioritization is applicable to entire use cases as well as individual requirements.

When to prioritize

How soon should you make choices? The answer has to be as soon as you understand what you have to choose from. The more visible you make your knowledge, the more possibility you have to make and help others make informed choices.

✓ **Early**

You should start by doing a project kick-off to assess your level of knowledge with a high-level context analysis. At that point, because you have some identifiable pieces, you can do a rough prioritization and give each business event response a priority rating. Something like High, Medium, Low will work fine. You can use the results of this prioritization for deciding which parts of the business you will investigate first. Also you can use it, even at this stage, to guide you in version planning.

✓ **Progressively**

When you start to write atomic requirements you should progressively consider whether you can prioritize them. If there are there any requirements that are obviously Low value, then tag them L. Use the idea of Customer Satisfaction and Customer Dissatisfaction to help other people make choices.

Part of the reason for this progressive prioritization is expectation management. We use the term requirements, and people often think that means once they are written down they will definitely be implemented. What we really mean are desires or wishes that we need to understand well enough to determine how and/or whether we can implement them. For example there might be a requirement that is really high priority but, due to a mixture of constraints, we cannot meet the fit criterion 100% . However we do have a solution that will fit 85%.

If we have been progressively talking about prioritization throughout the project, people are able to accept the compromise without feeling as if they have been "had". We prepare people for the idea that we cannot implement all the requirements, but there is no point in saying this just once. To effectively manage expectations we need to continue to mention the need for prioritization and choice.

What to prioritize

Factors that commonly affect prioritization decisions are some combination of:

- **Value** to the **business** (how much will the business benefit?)
- **Value to customers** (how many people want it?)
- **Minimize Cost** of implementation (how much cost to develop?)
- **Time** to implement (how much time to deliver?)
- **Ease** of **technical** implementation (how technologically difficult?)
- **Ease** of **business** implementation (how organizationally difficult?)
- **Obligation** to some external authority (necessity to obey law?)
- **Political obligation** to some authority who has control (how risky is it to lose a stakeholder?)

Not all factors are relevant to every project, and the relative importance of the factors is different for each project. And, within a project, the relative importance is not the same for all of the stakeholders. So, given this combinatorial complexity, you need some kind of agreed prioritization procedure to provide a way of making choices. Part of that procedure is to determine when you will make prioritization decisions.

How to prioritize

Here are some procedures that you can use prioritize your requirements.

✓ **Stakeholders / Value to business (Impact / Urgency)**

? **How many of the stakeholders will be affected?**

? **How much value will this bring to the business?**

? **Bad - how important is this to you?**

Priority is comprised of two components. The first is impact or the number of stakeholders that will be have their needs satisfied through this requirement. The second is urgency or the amount of value that is potential in the completion of this requirement. A combined score determines the full priority of the requirement.

In ITIL, a popular business framework, priority is a combination of impact and urgency. Impact is the portion of the business that will be affected by the inability of a requirement to function. Another way to think about it is at the help desk where after the software project has been implemented and a bug is being called in. The question to ask is “How much of the business has been shut down or affected negatively by the bug in this application?" It is the internal view of the service to the business from the customer.

The other part of the priority is urgency. The question here to ask is "How severe will the business be hurt over time by the outage of this requirement" or "How much value is possible to be gained from this requirement over time that we are missing right now?" It becomes a question of time and benefits because of speed. It is the external view of the service to the customer from the business that we are concerned about here.

The question you don't want to ask is "How important is this requirement to you?" Almost everyone will prioritize their requirement as being the top requirement because from their view, it is. The view that needs to be enforced is the view from the business requirements for the project under analysis. Therefore the questions have to be phrased with an emphasis on the value that is returned to the business.

✓ Priority groups

You can grade your requirement priority however it suits your way of working. A common way of grading requirements is **H**igh, **M**edium of **L**ow. But you can use any other grading that suits you.

For example some people grade requirements by **release** or **version** number: R1, R2, R3 ... The idea is that the R1 requirements are the highest priority and are intended to appear in the first release, the R2 in the second release and so on.

But suppose that, after you have tagged the requirements by release you then discover that you have too many requirements in the release 1 category. At that point you need to prioritize the release 1 requirements into high, medium and low.

The idea of sorting the requirements into three categories is often referred to as **triage** and has been written about by many people in the software engineering field. Triage is a term that comes from the field of medicine. When there has been a disaster that injures a number of people there are usually insufficient resources to treat all the patients. So the doctors need to categorize the patients into three groups:

- Those who will benefit from medical treatment
- Those who will live without treatment
- Those who will not survive

These groups roughly map to the idea of high, medium and low prioritization categories. It is a powerful idea because it makes people realize that there are many more difficult prioritization decisions in the world than prioritizing requirements for a product. And, if you work as a team it is possible to arrive at a decision.

If your progressive prioritization using the above ideas still leaves you with more requirements than will fit into your budget, then you need to go into much more detail.

✓ Cost / benefit

Another area of quantification analysis involves the costs of the project and choices here drive the requirements that have a real effect on what benefits the users get and what satisfaction your customers receive.

There are many ways to do an evaluation this way. Hard factors are tallied up in terms of hard benefits and hard costs with a resulting net value. Doing a year to year future value can show when the system starts being a real benefit but typically after a few years, the benefits start looking really risky. It is traditional to use discounted values in a Net Present Value calculation to show the total.

Soft factors can include line items like maintainability of the new system or how easy it will be to adapt to changes, how well the system communicates to the customer, and how much stress is relieved.

Factors can come from needs and references. Business goals and your judgment are important first resources but you can always turn to academic resources to see if any industry specific types make sense to you.

✓ Satisfaction scale

This concept was introduced by William Pardee as a way of helping people to make choices. Instead of saying "is this high, medium or low priority" he asks two questions: "On a scale from 1-5 how satisfied will you be if we implement this requirement" and "On a scale from 1-5 how dissatisfied will you be if we do not implement this requirement". This helps people to make consider the requirement rather than just making a binary choice. You can use customer satisfaction/dissatisfaction ratings as input to determining the requirement priority grading.

✓ Weighted criteria

You can use a spreadsheet to prioritize the "overflow" requirements. Ideally, especially if you have done a good job on progressive prioritization, these requirements will fit into the Medium or Would Like If Possible category. In other words you have been able to fit all the High priority requirements into your budget. If this is not the case then prioritize the High priority requirements and either drop the Medium and Low or tag them for future releases.

Volere Prioritisation Spreadsheet

Copyright c The Atlantic Systems Guild 2002

Requirement / Product Use Case/Feature	Number	Factor - score out of 10	% Weight applied	Factor - score out of 10	% Weight applied	Factor - score out of 10
		Value to Customer	40	Value to Business	20	Minimise Implementation Cost
Requirement 1	1	2	0.8	7	1.4	3

Requirement 2	2	8	3.2	8	1.6	5
Requirement 3	3	7	2.8	3	0.6	7
Requirement 4	4	6	2.4	8	1.6	3
Requirement 5	5	5	2	5	1	1
Requirement 6	6	9	4	6	1.2	6
Requirement 7	7	4	2	3	0.6	6

On the spreadsheet the number of factors are limited to four. If you are trying to manage more than four prioritization factors it is very difficult, if not impossible to come up with an agreed weighting.

The **%Weight** is the agreed percentage importance of a factor. You arrive at the percentage weight by stakeholder discussion and voting. The total percentage weights for all factors must be 100%

In column 1 you list the requirements that you want to prioritize. These might be atomic requirements or they might be clusters of requirements represented by product use cases, product features or business event responses.

Then for each requirement/factor combination you give a **score** out of 10. The score is assigned based on how much of a positive contribution do you think that this requirement makes to this factor. For example, for requirement 1we have assigned a score of 2 for the first factor because we believe that it does not make a very positive contribution to Value to Customer. On the other hand, the same requirement scores 7 because we believe it does make a positive contribution to our business. The score for minimizing cost of implementation is 3 because we think this requirement will be relatively expensive to implement. And ease of implementation scores 8, to reflect that this requirement will be easy to implement.

For each score, the spreadsheet calculates a weighted score by applying the % weight for that factor. The **priority rating** for the requirement is calculated as the total of the weighted scores for the requirement.

You can use a variety of voting systems to arrive at the weights for the factors and the scores for each requirement. Of course the spreadsheet is merely a vehicle for making it possible for a group of people to review and arrive at a consensus for prioritizing the requirements. By making complex situations more visible you make it possible for people to communicate their interests, to understand other peoples' opinions and to negotiate.

Levels

The grouped level

? **What broad grouping of tasks do you want the system or <role> to do?**

This is a fuzzy level and summarizes or expresses broad ideas about the scenarios that the customer wants to spend money on for this project. It is worded in a vague manner that categorizes the types of activities that would be found under it if it were an item on an application menu bar but could also be a full page menu that had selections to choose from more like a mainframe menu.

You could also call it a menu bar level requirement. Microsoft Word groups functional requirements at the menu bar level concerning common actions, insert actions, page layout actions, etc. They are a summary of the items below it and help limit the detail that you work with functionally.

Groupings can be made up of scenarios, partial scenarios, or even a type of design. They all make it easier to diagram and understand the fundamental nature of the functional processes.

Words used to describe these types of requirements are:

- manage
- handle
- control
- do
- work with
- take care of

A very common grouping of scenarios for managing a data entity is broken down in to the component parts of Create, Read, Update, and Delete (CRUD). Each data operation is a separate workflow but most people think at a higher level of requirements when they say they'd like to take care of or manage a customer's account.

Sometimes on a very large project or a very functional system, there may be a need to segment the grouped requirements into even higher grouped types.

The goal-driven scenario level

? **What sequence of steps can be named that can give value to the business?**

? **Can this sequence be repeated?**

This puts a name on a process from when the user begins using the system to when they stop using the system. This is the highest level type of requirement that can be coded as a test case. A use case is generally thought of as being this level. In XP, this is known as a story. In practice, at the end of development, these can provide verification of what was asked for by implementing it as a set of test scripts. This is the level that is the best to create in your requirements document to be of use to everyone.

A scenario is a group of tasks or workflow that can be performed repetitively and gives value to the business. If another person can come to the system or start the process in business after you have done the scenario, then you have a complete set of tasks.

A business type of scenario is related back to the work package in the work breakdown structure. It is a managed unit of work that can be divided into no more than about 80 hours of effort, is no longer than a single reporting period, and makes sense.

The partial scenario level

? **What are the reusable named processes in the scenarios?**

? **What belongs as a part of a scenario because it doesn't meet the walk-away test?**

This is about functionality does not describe a full business workflow nor does it have a business goal. It is a set of tasks that are logically grouped together but are a part of another goal-driven scenario. These partial sets of features are able to be broken down even further into a workflow set of steps but then are reused in different full scenarios. A common reusable set of requirements is the user logon for wrapping a functional workflow in a security "shell" or session. It is a combination of the entry for a user name and a password, validation and other operations.

Partial scenarios can be lumped into a grouped scenario set or can be broken down into other partial scenarios.

The design "requirement"

? **What is being described with automation or what isn't being used already in mind?**

This is not a business description of a function. It's about how to solve the design of the system. It describes the look and feel of the product where the interfaces are designed. These requirements may help put together the user screens but a prototype is always a good idea so that the customer gets to interact with the product as a real thing and suggest tasks that would be appropriate because they've seen them on other interfaces.

Deciding the level

Executives will typically give requirements at the highest level which means they will think of the benefit to the business and give you a business requirement or think of the general task that will be accomplished in terms of management thinking which means it will be a grouped requirement.

Managers of a process with a good business background think in full processes and will give requirements at the scenario level. People who use the system and programmers generally think at and give requirements at the design level. Design level requirements support visual requirements or are given by very detailed oriented people.

Exercise – Requirement levels

Label the statements with the correct level of granularity.

1. Group of goals
2. Goal
3. Partial goal
4. Task
5. Design recommendation

______________ W1 – The system will count hits (files requested) for each web page.

______________ W2 – The system shall create a report that shows the number of people to visit web pages.

______________ W3 – The system will manage marketing information on the web site.

______________ W4 – The system will report on file tracking information throughout the web site.

______________ W5 – The system will use a hit counter made up of individual digits.

______________ H1 – The system shall check in a guest.

______________ H2 – The system shall validate a password.

______________ H3 – The system shall log in a clerk to the system.

______________ H4 – The system will manage hotel guests' accounts.

______________ H5 – The system will report on occupancy levels.

Traceability

? **Can you define a requirement (at a higher level) that summarizes a group of requirements?**

? **Can you define a requirement (at a lower level) that is a part of the requirement?**

Traceability ensures completeness, that all lower level requirements come from higher level requirements, and that all higher level requirements are allocated to lower level requirements. Traceability is also used in managing change and provides the basis for test planning.

Traceability requires unique identifiers for each requirement and product. Numbers for products are established in a configuration management (CM) plan.

Traceability will be one type of verification that we do throughout all stages. The relationships between requirements and the resulting code must be tracked so that when a change is made the related changes are also made. Without making those changes, the system that functions as a life-critical system becomes dangerous.

Traceability is the structured view of the requirements by level. A grouped level requirement traces down to many different requirements. A goal-driven scenario level requirement will trace down to many different tasks that it takes to complete and can be combined with partial scenarios. And the design choices made for each task trace up to that task requirement. IDs are used to make relationships between the requirements.

A good test using traceability is to determine the higher requirement that a requirement traces up to or come up with several lower level requirements that would support the requirement you are working on. Do the ones you added make sense for the right level of requirement?

It always is an advantage to include business goals in the requirements to trace up from all functional requirements. They improve the supplier’s understanding of the domain and allow you to cross-check requirements for validity. Other items like domain descriptions and design examples of previous or competing products are helpful to improve the supplier’s understanding also.

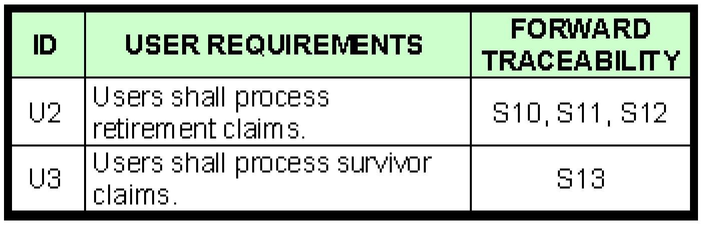
✓ **Use words “roll-up” and “drill-down” to talk about relationships between levels with business language.**

Traceability matrix http://www.jiludwig.com/Configuration_Management.html

A traceability matrix is a report from the requirements database or repository showing traceability e.g. between requirements at different levels and between requirements and other work products such as the code module that is responsible for fulfilling it.

The purpose of the requirements traceability matrix is to help ensure the object of the requirements conforms to the requirements by associating each requirement with the object via the traceability matrix.

The traceability matrix is created by associating requirements with the products that satisfy them. Tests are associated with the requirements on which they are based and the product tested to meet the requirement. In traceability, the relationship of driver to satisfier can be one-to-one, one-to-many, many-to-one, or many-to-many.



ID	USER REQUIREMENTS	FORWARD TRACEABILITY
U2	Users shall process retirement claims.	S10, S11, S12
U3	Users shall process survivor claims.	S13

A traceability matrix is used to verify that all stated and derived requirements are allocated to system components and other deliverables (forward trace or tracing down). The matrix is also used to determine the source of requirements (backward trace or tracing up). Requirements traceability includes tracing to things other than software that satisfy the requirements such as capabilities, design elements, manual operations, tests, etc.

The traceability matrix is also used to ensure all requirements are met and to locate affected system components when there is a requirements change. The ability to locate affected components allows the impact of requirements changes on the system to be determined, facilitating cost, benefit, and schedule determinations.

Traceability strategy

The traceability strategy is devised by graphing relationships between work products. Software is developed in stages, each with work products that can be associated with work products of the previous stage, creating the ability to trace from the requirements through the work products and from the work products back to the requirements. What work products are required depends on the organization's standards and development practices. Ideally, the time to create a requirements traceability matrix (RTM) is when the business requirements are first formed.

Design, code, test, manuals, and other work products are developed in accordance with the requirements. As they are developed, work products in the functional baseline may be updated or modified in accordance with the organization's configuration management process. Traceability is used throughout to verify that goals and requirements are being met.

Traceability types

The idea of traceability is to connect the meaning or dependency that an item has on another item. The relationship can be time-based or through context. Typically, the traceability that we are looking for is the connection from a functional requirement to the code module that satisfies the software need.

Types of elements that can be created to take advantage of the traceability in software or just for you as a way of thinking about them can be issues, assumptions, action items, request for new features, glossary terms, and bibliographic references.

Links for traceability that should be considered to make are:

- requirements and use cases to test plans, test specs, and test results
- user needs to product features
- product features to requirements and use cases
- requirements and use cases to implementation units such as functions, modules, classes, or components.
- implementation units to test plans, test specs, and test results.

Too little information

Modifying phrases

Words and phrases include: as appropriate, if practical, as required, and to the extent necessary/practical. Their meaning is either subject to interpretation or they make the requirement optional, even if it is included in a contract. If the requirement isn't important, assign a low priority to it. Phrases like "at a minimum" only ensure the minimum, while "shall be considered" requires the contractor to think about it.

Avoid "etc.," "and/or," "TBD." TBD can be used during the analysis process to indicate ongoing work, but should not be in the final requirements.

Vague verbs

Information systems receive, store, calculate, report, and transmit data. Use words that express what the system must do. Vague words inject confusion and are typical of very high level requirements or poorly understood ones. Examples of frequently used vague verbs are:

- manage
- track
- handle
- flag

Vague words and phrases make a requirement untestable.

Incorrect: The system shall process ABC data to the extent necessary to store it in an appropriate form for future access.

```
Correct: The system shall produce reports on ABC data.
```

Most adjectives

Adjectives can make meaning open to interpretation.

"All" is one of the most abused adjectives in requirements, but other adjectives create similar problems. "All" does not protect a client or ensure completeness: using "all" means the analytical work is incomplete. The unfinished analytical work is transferred to subsequent phases of the SDLC. How would you plan for, implement, or test "the system must calculate all payments" without knowing what specific payments are involved?

Later in the SDLC, others must complete the analysis outside of the analysis process and hope they're right or annoy clients with additional questions. The client feels cheated. In fact, complete, traceable requirements are what give insight into and control over the contents of the system and support accurate planning and cost estimating for subsequent phases—not vague adjectives like "all."

Some other adjectives that make a requirement untestable are: robust, safe, accurate, timely, effective, efficient, correct, expandable, flexible, interoperable, maintainable, modular, portable, reliable, user friendly, adequate, better, and state-of-the-art. When these appear, usually in a statement of work, proxies or rewriting are needed to define what the client actually wants.

```
Incorrect: The system shall be written in C++.

Correct: The system shall follow the coding standards in the corporate
document XYZ1010.doc.
```

This is a design solution based on whims of a stakeholder. It would be better to understand the root cause for the solution. For example, if this requirement exists to ensure compatibility with another system, then it should address compatibility instead of requiring a specific language. On the other hand, for a statement of work, specificity in a requirement might be needed so contractors can create proposals and bid qualified personnel.

Most adverbs

Adverbs and adverbial phrases can make meaning open to interpretation. Examples are: quickly, safely. Some are impossible to prove: always, never.

```
Incorrect: The system shall produce the ABC report in a timely manner.
```

"Timely" needs to be defined in accordance with the needs of the organization.

```
Incorrect: The system shall require users to select only one of the
following...
```

The placement of adverbs can cause problems. In this requirement, "only" can mean the one action available to users is "select," or it can mean users are limited to one item from the list.

The correction is not adding more words ("one and only one"). The correction is deleting "only," which is an extraneous word.

Time without units

Time-based words can cause confusion or unintended meaning:

- until
- after
- as
- before
- once
- until
- when
- earliest
- latest
- instantaneous
- simultaneous
- while

```
Incorrect: A weekly report will be created at the earliest possible
time.
```

Which is when?

```
Correct: The system shall email a weekly report by COB each Monday to
management (see Stakeholder list / management).
```

Unnamed things

Indefinite pronouns stand in for unnamed people or things, which makes their meaning subject to interpretation. Some of these may find their way into requirements:

• all • another • any • anybody • anything • both • each • either • every	• everybody • everyone • everything • few • many • most • much • neither • no one	• nobody • none • one • several • some • somebody • someone • something • such

Pronouns without reference

> **Another previous requirement:** `The data shall be acquired from the AJH database.`
>
> **Incorrect requirement:** `It shall be displayed on the screen.`

When this occurs, the writer is usually relying on a nearby requirement in the requirements document for the meaning of "it." As requirements are assigned for implementation, they are often reordered and regrouped, and the defining requirement is no longer nearby.

Passive voice

Requirements are written in active voice, which clearly shows X does or provides Y.

> **Incorrect:** `Z shall be calculated.`
>
> **Correct:** `The system shall calculate Z.`

Assumptions and comparisons

The requirement

> `The system shall increase throughput by 15%.`

sounds testable, but isn't. In this real life example, the assumption is "over current system throughput." By comparing to another system, the meaning of the requirement changes when the other system changes. For example, if current system throughput becomes even more reduced, the required result could be a new system with less throughput than that which initiated the requirement. On the other hand, if current system throughput is dramatically improved while waiting for the new system, the requirement could become technically infeasible. Such requirements create disputes later in the development cycle.

> **Incorrect:** `The system shall address the future needs of users.`

This example is sometimes found in requests for proposals. The writer is, probably, thinking ahead to after the contract is awarded. The requirement is meaningless because whenever it is read, it will point to the future.

Too much information

Conjunctions

Review any requirements conjunctions such as "and" or "but" to see if the requirement can be interpreted in more than one way. It could have created a compound requirement or a requirement that should be separated into two or more requirements.

While is it not incorrect to use "and" where multiple items must be included, it facilitates work later, particularly testing, if the multiple items are shown as a list. The list approach avoids misunderstandings when reviewing requirements for traceability.

For example the requirement

```
The system shall calculate retirement annuities and survivor benefits.
```

is a compound requirement. If the first build only includes retirement annuities, it will fail the requirement during testing. But, if the product is marked as having passed testing, the record is that the entire requirement was met. As a result survivor benefits may be excluded from later builds.

Requirements with lists are not necessarily compound:

- The system shall report retirement claim status as one of the following:
  - pending
  - complete
- The interface shall accept the following data:
  - data A
  - data B
  - data C

These requirements are not compound because they express a single need at their level of decomposition. The second list could be written as three requirements, but this would add unneeded complexity and maintenance overhead to the requirements repository. At this level of requirement, there is no need for further decomposition: all three data items must be included for the interface to pass testing.

While is it logically correct to use "and" where multiple things must be included, showing multiple items as a list makes future work easier, particularly in testing. The list approach also avoids misunderstanding when reviewing requirements for traceability.

Definitions - list or glossary item

When you need to list a number of elements that can be expressed in a phrase like “customer record,” you will need to define what they are. One way is to create a separate testable list of requirements items like:

```
R-10 - A user may update the following data in the customer record:

R-10.1 – the e-mail address

R-10.2 – the password

R-10.3 – the credit card information

R-10.4 – the shipping address
```

The downside to this is that the elements will not be understandable if they are sorted out of the original order that they were put it by the author.

The best way is to express the same grouping is in a data dictionary or glossary item:

```
A user may update the customer record (see Data Dictionary).
```

Data Dictionary

customer record – The customer related fields of

- name (required)
- e-mail address (required, validated)
- password (required, 6+ characters, at least one numeric)
- credit card number (required, Luhn algorithm checksum validation)
- shipping address (required) …

The reference can be specific with the parenthetical words, the use of a special font or style like the bold above, or hyperlinked if the document is used often in Word format.

Other systems

Some requirements are not about the system under development at all and are about another system and how they must interact with the system being built. That changes the scope of the project from a single system project to a multiple system project. For instance, if a requirement states that

```
The IT support group will maintain the web servers.
```

and the IT group doesn't even know you are acquiring the new web servers, you have some project coordination to do. Other types of multiple system dependencies occur when the system needs to have an interface to another system for either input or output.

These types of requirements should be categorized under a different project heading and not in the system under development.

Useless information - negative requirements

Everything outside the system is what the system does not do. We need requirements to test that which the system does. Testing would have to continue forever to prove the system does not do something. State what the system does and can be tested to do.

```
The system will never exceed the limits of the network.
```

Functional requirements

✓ **Uses a strong specific verb is used to describe what the system or <role> does**

✓ **A system functional requirement starts with "the <system> shall …"**

✓ **A business functional requirement starts with "the <role> shall …"**

The functional requirement statement contains

- A responsible party/noun
- The action/verb to be done
- A description of the things/direct object which the verb acts on.

All other types of descriptive text will likely be part of a business rule or an explanation of how the business is to implement the requirement.

```
The system will email a completed transaction receipt to the customer.
(rule references for receipt timing, data, types of customers, etc.)
```

There are several detail levels of functional requirements but all of them should express the "what" of the system in a workflow rather than the "how." Of course, it's easy to get confused about this because someone will say a button on a GUI is a what and not realize it's really a how you initiate an action. The action is the what. Remember that all action should be at the analysis level first which means that there is no mention of any kind of automation and therefore this business process description could be applied to any kind of automation or implemented without a computer.

The functional requirements are the business processes sometimes combined with business rules that further define the business problem. Some authors call these business requirements so make sure you understand the context. Also, authors talk about system requirements which are a subtype of functional requirements as defined here.

These processes can be either automated or manual. These requirements specify the features or operations of the system that records, computes, transforms, or transmits data. The user interface is an important part since most of the data is entered and viewed through it but is not a functional part per se.

The primary word is "what" rather than "how" here for discovering operations of the project. The GUI is a documented design of "how" we use visual or audio means to do the operation. The word "why" is used to get back to a root cause and move up the chain to the original business requirements that launched the project.

These functional requirements can be specified in different ways or styles such as:

- a function or feature list
- a function description

- an activity diagram by function
- a user interface prototype
- a work task list (use cases)

A functional requirement must be able to support and roll up to a business requirement. They can created by the business subject matter experts (SMEs). They also should always have a stakeholder and organization associated with them as a reference to when they need more clarification down the line. Most functional requirements can be simply stated but often use multiple rules that should be transferred to a non-functional requirements section.

The types of functional requirements include:

- **System requirements** - extracts the processes which were identified in the general functional requirements to be automated and assigns them to a specific system. The functional requirements can be considered a high-level system requirements document if it is all about the systems because the system requirements details the system requirements in data, interfaces, but should be written in language that the business understands.

Any mention of internal application components means that you have crossed the line into design. Test cases are developed from these and not the design specifications.

The statement of a system requirement should usually be stated beginning with “the system shall…”.

- **User requirements** – changes to a business process.

Example: The data entry clerk shall select the billable party for a multiple party invoice.

Verb style

There are three main ways to word a requirement statement. Requirements are a statement of work to be performed and, even though a requirement is included in the contract, using non-imperative words make implementation of the requirement optional, potentially increasing cost and schedule, reducing quality, and creating contractual disputes. If it isn't that important, associate a low priority rating with the requirement and schedule it for later implementation.

Present tense

The present tense is the easiest to work with.

```
The system displays the account details.

The system toggles the switch between on and off when a password is
entered.

The system sends an e-mail notification to the user when the transaction
is complete.
```

Imperative

The imperative version leaves out a critical part of who is doing the work so that leaves the present tense and the modal verb which isn't really much different.

```
View account details.

Toggle the switch between on and off when a password is entered.

Send an e-mail notification to the user when the transaction is
complete.
```

Modal verbs - must, shall

The modal form is the best, is typically used in corporations, and recommended by standards.

```
The system shall display the account details.

The system shall toggle the switch between on and off when a password is
entered.

The system shall send an e-mail notification to the user when the
transaction is complete.
```

Grouping

You will want to segment the groups of requirements by some method. Some people just use headers in a Word document between the groups of requirements but this doesn't work well if they are sorted in any fashion.

One way is to group them by IDs based on the entity type. Name your ID in a few letters based on the object of the requirement prefixed before the number of the ID. The object is the initial noun in a sentence that takes action. The object could be a product (such as an automated system), manual processes, project activities, or contract. The nature of the object determines the category of the requirement. For example, the responsible party of the requirement

```
SF 28.1      The system will calculate XYZ.
```

is the system, making this a "system functional" requirement. The type of the requirement

```
PM 66.004    The project lead will report progress monthly.
```

is project management. It is a non-technical requirement that is usually found in contracts.

Another style of grouping would be to use a use case ID to start with and then follow with a number of the flow of events that the functional requirement was extracted from and another unique identifier. For example, the requirement

```
UC43.4a The system shall prompt for a location code.
```

Indicates that the use case is 43 and from step 4 and is the first requirement from that step.

Workflow description

An event is something that requests a system to perform some function. Usually an event arrives with some data telling the system what to do. It is an alternate way of saying a function call or a method message.

An event list shows the types of events that can arrive at the system. Since an event causes the system to perform a function, we can make a similar list of the functions that the system can perform.

Domain events arrive to the domain from the surroundings. Sometimes these are called business events or triggers. A phone call prompts the call center employee to look up a record on a past call. The task is what the employee does as a whole such as enter another record on an open ticket or close out a ticket. The task is broken down into smaller parts or refined through the definition of the task at a more fine-grained requirement level that gets to the functions.

Use cases are functional sequences that break down into functions as a part of the use case detail. Again, it is not completely necessary to work at this level when you are starting with requirements but it is possible because many different tasks can be done in parallel.

The creation of a function list from the requirements is a task that is appropriate for the analysis stage but can be done in tandem with the creation of the requirements and can be made a part of the requirements for clarification. A list that itemizes the business events of a hotel system might start out by listing the following:

- R1: The product shall support the following business events / user activities / tasks:
    - R1.1 Guest books a room
    - R1.2 Guest checks in with a reservation
    - R1.3 Guest checks out
    - R1.4 Guest changes room
    - R1.5 Clerk handles service note of room
    - R1.6 System periodically transfers accounting data to accounting system

The types of functions specified by requirements that would end up in a list to support those business tasks might look like this:

- R2 The product shall handle the following events or provide the following functions:
    - R2.1 User finds a free room for guest
    - R2.2 User records data for guest
    - R2.3 User finds data on a guest
    - R2.4 User records reservation for guest
    - R2.5 User prints guest confirmation for accounting
    - R2.6 Clerk records check-in of guest
    - R2.7 Clerk checks out guest

- R2.8 Clerk records service of room
- R2.9 System formats accounting data for accounting system

Notice that there is a particular syntax that is used depending on the level of the requirement. For the business event or user-goal level description you have a noun-verb-direct object. For the more detailed type of requirement there is a clause at the end that describes a little more of what the category that the task is a part of. It's a noun-verb-direct object- descriptive clause. The descriptive clause is used to group like things together because the amount of detail generated is clumsy.

The lists are generally done during the analysis phase but initial efforts can be done when appropriate to help logically structure the requirements and give the analysis phase a head start. The part that is missing from these lists is the sequence that they happen in. A massive list makes you feel good about how much work you have accomplished but doesn't tell you how much is not defined which in the end could cause the users to reject the system. The sequence can be described in type of use case or an activity diagram. This also works better for the customers since they wouldn't know the low-level functions necessary to fulfill the business events or domain events / tasks.

Non-functional requirements

? **What adjective or adverb describes how all of the functional parts should behave?**

? **What details do not affect the functionality of the system or <role>?**

If the requirement doesn't have a subject doing something then it's not functional. Sometimes statements blend a functional requirement with a rule to describe how it should work. Common types of these non-functional requirements are

A non-functional requirement is stated with

- A responsible party/noun
- The action/verb which is not a time-constrained event
- The specific quantity of units to define how to test the action

```
The system will secure credit card information according to PCI Data
Security Standard. (https://www.pcisecuritystandards.org/)
```

Remove jargon, abbreviations, or acronyms such as point and click, plug and play, high definition, Retina, UI, UX, GUI, etc.

Attributes

General operational environment

Performance

Reliability

Robustness

Security

Usability

Development environment

Efficiency

Maintainability

Reusability

Testability

Deployment Environment

Availability

Flexibility

Interoperability

Installability

Portability

Recoverability

Safety

Scalability

Types

- **GUI or design "suggestions"** - users love to help design the application before the functionality is decided. For a good read, look up the term "bike shedding." It's good to record ideas for what it should look like as a way to capture stakeholders' root causes that can't be put in words, but the best design choice is made by a designer or technical staff who is aware of what really works well.
- **Data requirements** – necessities of information at the entity (non-technical) level. This is information about the surroundings, such as customers, or other entities in the business realm. Input and output formats are important to link up with other systems, and if they plug into the system directly to talk to it, then they are called interfaces. The interfaces will have data that tracks the communication state of the system. There is also a system state which is all the data in the system or sometimes just the part that is under discussion.

Data requirements are often included in the functional requirements since functions rely on data. It is usually better to have them separate though.

> `The sales rep shall view` **`customer billing information`** `for new orders. (Billing info includes a new order flag)`

- **Quality or integration requirements (internal SLAs) –** statements of desired performance, availability, reliability, usability, etc. specify how well the system performs the functions in terms of efficiency, response time, up time, accuracy, throughput, usability, and ways maintenance should be performed. Quality requirements are included in use case specifications if no other place exists for them.

A measurement must be in quantity as well as units e.g. 500 sheets per minute, a minimum of 730 lumens of light, operational 24/7 with 99.99% uptime, etc.

> `The service rep will capture a telephone customer order within five minutes.`

- **External interfaces** – systems that require usage due to infrastructure needs.

- **Training and support requirements** – the amount and length of training to make the new functionality available in the workforce as well as customer service or help desk needs.

```
The training department shall offer a class per week on the admin
functionality of the system until demand tapers off to four students a
week.
```

- **Network requirements** – a system need that stems from the network rather than the operations directly.

```
The administrator shall reset server alarms from the network control
center.
```

- **Security requirements** – needs associated with accessing the system and administering that access.

```
The security manager shall administer access to MTPs based on job role.
```

- **Testing requirements** – detailed descriptions of testing beyond the normal test plan such as stress and load testing.

```
The system shall handle 2,000 users per second making a transaction.
```

- **Conversion requirements** – needs stemming from data migration or converting processes.

```
A manager shall view historical customer information once customers are
converted from X billing to Y billing.
```

- **SLA Requirements (external)** – statements of customer agreements and legalities.

```
The system shall query trouble tickets and determine if Mean Time to
Response is less than two hours per business customer contract.
```

- **Reporting requirements** – the needs of the organization or process.

```
An admin shall query business orders with the time used for
provisioning.
```

- **Operational requirements** – the needs of the system e.g. archiving, purging, replicating, etc.

```
The customer service rep shall view archived customer records within 5
seconds.
```

Data analysis

CRUD analysis

CRUD stands for Create, Read, Update, and Delete which are the basic data functions that user interfaces implement. Another useful function is an overview of several types of data and search functions The overview and the others can be remembered, (it's really not a joke but it just works out that way) as O CRUD.

Check to see if the your data needs a create, read, update or delete task. Another way to handle this that people like to do is take the use case up a level to a grouped task and combine all the CRUD user-goal use cases into a single grouped use case that is named like "Manage xxx data." Of course, if the actor should not do all of the functions, you can not use that type of grouped use case. Here are some questions to ask about these checks:

- Can each entity in the data model be created and deleted by an event or function or user task?
- Can each field in the data model be read and updated by an event or function or user task?
- Are all events handled by a function or user task?
- Are the necessary data for each task or message flow available in the data model?
- Are all data on the screens available in the data model and vice versa?
- Are all tasks conveniently supported through few screens?

CRUD - process vs. entity	Entity type									
	Categories	Subcategorie s	Products	Product variants	Product related	Specials	Orders	Order items	Order status	Comment
Planning	crud	crud								
Marketing	r	r	rud	rud	rud	rud	r	r	r	
Selling	r	r	r	r	r	r	cru	cru	cru	
Compensation						r	r	r	ru	
Shipping							r	r	ru	
Operations							r	r	ru	
Purchasing			crud	crud	crud	crud				
Forecasting						r	r	r	r	

Receiving			ru	ru	ru	r				
Ordering			ru	ru	ru	r	r	r	r	
Research	r	r	r	r	r	r	r	r	r	

The data dictionary

If you have a data model which represents the data in graphical form, can you tell me what all the reasons the fields are there right? Of course not. You will have to have some description of why the fields were put in place. For instance, why does the guest need a passport? Is it required? For what kind of guest should it be required? Does it matter if we leave address2 empty and use address3 or should we use address2 first?

The data dictionary and associated models specify the data for the system but the functional requirements specify what data is to be used for and how it is processed and output. You will want to learn a few techniques for notating the requirements, ensure that the customer can validate them, ensure that the developer can verify them, specify a little bit about the surroundings or product functions and what level they are applicable for. Some levels talk about following standards and processes where others talk specifically about product functions. And then there is the level which talks about the parts of the general task.

The data dictionary is structured however the writer would like. It could be alphabetic or by groups of fields possibly associated with an overview diagram. The information contained would be the best guess of name, meaning, other names used in practice, examples, type, and length. Not everything needs to be described especially if it’s trivial. It’s really annoying to read about a field called FirstName when the description of the field is “the first name of a person.”

Here are some questions to ask about the entity being described in a data dictionary:

- Which entity name is used in the product? How does it relate to other names used by people in the domain?
- How does the entity relate to other entities?
- Are there cases where it doesn’t have the usual relation to other entities?
- Are there special things to consider when creating or deleting the entity?
- Give a typical example of the entity. Give also unusual examples of the entity.

Here are some questions to ask about each field being described in a data dictionary:

- Where in the domain do the values come from?
- What are the values used for in the domain?
- Which values are possible for this attribute?
- Can the attribute have special values, e.g. blanks, and when?
- Give typical and special examples.

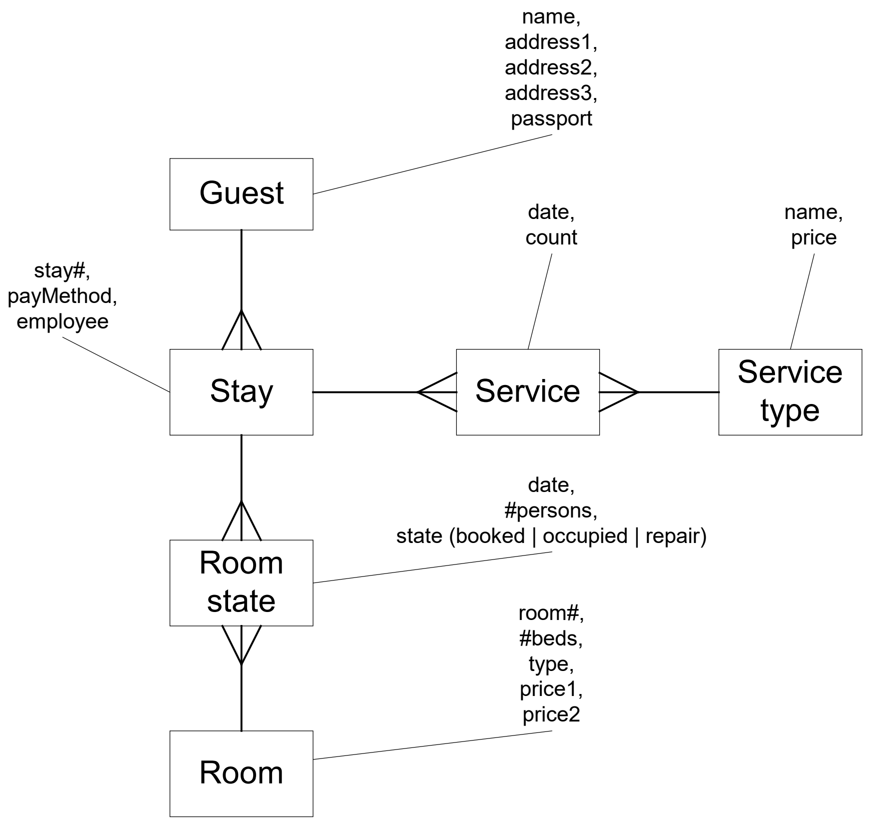
The data dictionary is a source for requirements and can be tied to one requirement by saying that the product shall store the data according to it. This may leave the software analysts with a sore feeling since they have final say about the organization but if you specify in the data dictionary whether it is flexible or not, it would help.

Data dictionaries are good sources for time wasting. since it's easy to keep refining and getting details on data forever. Simple is good. Omit the trivial stuff and work on the stuff that is more difficult. The dictionary will take up as much time as you give it so the deadline is an important factor is how big this can get.

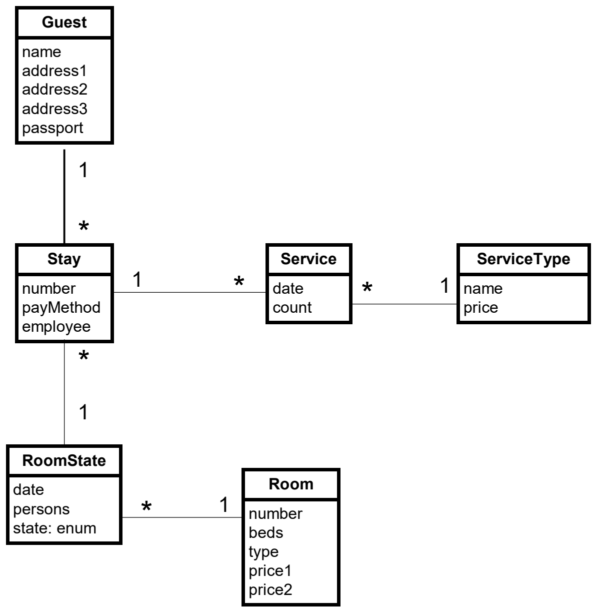
Data expressions help shorten the business formulas and are expressed with field names and regular expressions. They complement dataflow diagrams and are somewhat easier to understand than E / R diagrams for the customer since they look more like an Excel formula.

ERD

The data model is a block diagram that specifies the data to be stored in the system and relationships between the data. An entity-relationship model (E/R model or E/R diagram or ERD) is used to express data for a relational database storage system. In an object oriented project, the E/R diagram would be necessary for the storage but an additional OO diagram expressed in UML called the class diagram would be necessary to express the data when stored in an object model. A further complication is that these two do not match and in complex systems, it is necessary to develop a mapping for object to relational models called an O/R model.



Entity diagram



Just when you thought all of the models have been covered, you find out that XML is not a relational model and can’t express all of what the OO model does. That means that you should work out an XML model with the mappings necessary to go to relational or OO models if the data is complex enough. XML diagrams are not standardized so you can adapt one of the others if you want or just write and include the schema.

It is not the responsibility of the requirements analyst to develop the final logical data model since the analysis phase of the software development cycle is tasked with taking the business domain concepts and creating the best data model to manipulate them. Many people do the data work up front, however, so it is important to know the diagrams and their meaning. Many projects are extensions or have interfaces to data models that already exist. Data modeling is the skill that has its own rules and concepts apart from the requirements analysis.

Data relationship diagrams

When reading a data diagram the lines do not represent one relationship. They represent two relationships if they are connected to two entities. The relationships are the roles that the originating entity plays with the target entity.

Read the line in each direction for determining both the cardinality (number of times it can be related) and the role that it plays. A star or crow’s foot means that the target entity has multiple relationships to the originating entity. For instance you may have multiple books you own.

In reading the line in the other direction you find out that an originating entity can be related to one (required), zero or one (optional), one or more (required), or any other necessary constraint on the target entity. For instance, you may have a book that only has one owner.

Or you may have a book that has only two authors and other books may have different numbers of authors as well. In any case, you need a relationship to multiple authors.

Roles are assigned in the same manner but only one role is written on the line if any to show what the originating entity plays with the target entity. If there is no role written, it either means that it's not exactly known, it's too much detail, or it's just a simple relationship of containment (a thing has another thing).

For instance, a book can have a person entity role so the person's role is "authors" or read in the reverse way the role would be written "has author of." Use the simplest role to make the diagram easiest to read. You'll just have to guess which way the role reads unless you want to use a little solid triangle (UML) to point in the direction of the way to read it.

There can be multiple roles so you may have an "authors" role as well as an "owns" role and even a "borrows" role. Roles are written as verbs to make them easiest to read. Other possible common roles are maintains, uses, tracks, owns, sells, delivers, buys, creates, deletes, or manages.

Data diagram process

The way to diagram either an ERD or an entity diagram (a high-level class diagram in UML) is as follows:

1. Discover all the candidate nouns from the use cases/requirement docs.
2. Organize the simple datatypes (numbers, dates, text, flags) in groups by related concepts. These are now called complex datatypes.
3. Combine the groupings of simple data with any other complex datatypes to become a new complex datatype
4. Show the relationship of a complex datatype to another complex datatype by a line between the entity's simple datatypes.

Complex datatypes generally follow four types

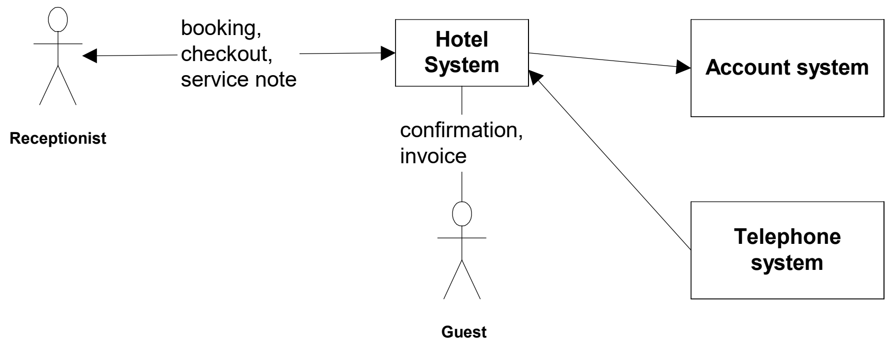
- people, places, things – a real person, airport, or airplane
- roles – a pilot or a customer service rep
- events – things that happen over time like a transaction
- reference data – flight schedule

The lines of the diagram should be read by a business person as one piece of business information that

- needs to be reported on with this info most often
- needs to be tracked with the info
- needs to be known when working with this info

Context diagram

The context diagram shows the product as a black box surrounded by users groups and external systems with which it communicates. Arrows show how data flow between the product and the surroundings. Each context diagram should have a textual explanation accompanying the graphic.



4 - A context diagram showing data interfaces

A similar type of UML diagram is the system-level sequence diagram that shows the relationship of the communication in time. This type of diagram mimics the UML collaboration diagram more to show how much activity there is by type between the systems. It is good to understand the system as a whole at the beginning of the project.

Relationship map

A relationship map shows info and products exchanged between external customers, providers and key functions in organization. It is also called a business interaction model, org context diagram, org relationship map.

Business rules

Business rules are identified in the normal course of requirements gathering and analysis and are often confused with the requirements themselves. A rule of thumb is if something defines a calculation or operating principle of your organization then it is likely a good candidate to be documented as a business rule. You want to separate business rules out of your other requirements artifacts because they may be referred to within those artifacts several times.

"I'm here because my boss said we should use more decisions tables for our project. What types of decision tables do you sell?"

A rule is a type of business constraint and is likely found in your list of requirements where the requirement starts with "if this… and if that." And kind of condition statement is a flag for a business rule. Business rules apply to either technical or business issues and are concerned with either

- process workflow or
- data calculation and validation

Create a list of business rules that, for example, starts with a code of BR### if you have many rules to document or reference them often in your artifacts. However, if you have only a handful of business rules or use cases, you may choose to document them right in the use cases or requirements document. Start out including them in the use cases until it becomes obvious, or painful, to do so. This may be because the sheer number of business rules is dominating the use case or because the same business rule is referenced in two or more use cases.

Qualities

Good business rules:

- describe one, and only one, concept. By ensuring that business rules are cohesive, you make them easier to define and increase the likelihood they will be reused.
- are stated plainly
- exist independent of procedures and workflows (e.g. multiple models).
- are built on facts, and facts should build on concepts as represented by terms (e.g. glossaries).
- are single sourced.

Classifications of business rules can be

- Data – Inference (by lookup)
  - KS state tax rate = 5.3%
- Data - Calculation (formula)

    - KS total tax rate = KS state tax + municipality tax rate
- Workflow – restriction (boundary definition)
    - Password must be six characters (reenter)
- Workflow – extension
    - Earnings in KC, MO requires an additional tax form.

A business rule is made up of many parts but overall there are only pieces of data to check and a result that is the decision to be made. The pieces of data can be called a facet or to a programmer they become variables. The decision to be made is the result of the rule or to a programmer, it's the return value of the rule function/method. If a facet has two possible values of data and another facet has three possible values of data, the number of outcomes to specify is two times three or six. All possible outcomes must be documented. The more facets you have, the more outcomes you have.

Using a table of business rules

A good way to identify business rules is in a table format. The table can be included in the requirements section itself if small enough, in a separate section if large enough, or in the glossary if re-used. Here's an example:

R-1. The system will send e-mail to clients who are described in the following table:

Facet	Facet	Facet	Facet	Result
Client	Amount due	Average purchase	Last purchase	Type of e-mail
Region 1 client	<10	100	<50	Follow-up
Region 1 client	<10	100	>50	Follow-up and discount coupon
Region 1 client	>10	100	<50	Follow-up
Region 1 client	>10	100	>50	Follow-up and small gift coupon
Region 2 client	Etc.			

Business policies

Policies are just groupings of business rules and are large concepts of business operations derived from your business goals. Most managers have a good sense of them but whittling them down into the specific rules is difficult.

Exercise – Rewriting

Analyze the following requirements and rewrite them as functional requirements to eliminate any problems. Do not include any business rules. Some will be business process requirements but most should be expressed as a system requirement.

Also, write down any other systems that need to talk to this quoting system in the next section.

Include any other systems requirements.

Business goal: The ability for Marketing to charge retail customers (without accounts) a higher rate for all shipments.

Original requirement Rewritten	New functional requirement	Rule	New stakeholder
3.1.1 All customers tendered through a retail counter site will be charged counter rates.			
3.8.1 Cash-only customers without accounts will be quoted and charged counter rates.			
3.8.2 Handheld courier devices will support counter rates as needed.			
3.9.1 A corporate credit card customer will not be charged counter rates but AmEx Small Business cardholders will be charged Counter rates.			
3.11.1 Customers without accounts should be quoted the counter rate when using the web quoting system.			
3.11.3 Customers without accounts will be quoted the counter rate when calling customer service.			

3.17.1 A counter rates guide will be available to retail counter sites.			
3.22.1 Customers using a counter rate will not receive a discount.			
3.22.2 Counter rates will be some percentage above list rate.			
3.22.4 They will be set by pricing in a rate table.			
3.22.5 The corporate data warehouse will support the counter rates.			
3.24.2 Invoice adjustments shall support counter rates to support account holders incorrectly billed a counter rate. Counter rate messaging shall be included in training.			
3.24.3 Shipments shall be invoiced to the Payor when an external credit card is declined and the customer is unavailable to provide other payment.			
3.27.1 The ability to flag counter rate transactions in the data warehouse and strategic marketing applications to properly track volume and revenue by retail location is needed.			
3.27.2 Couriers will be trained on how and when to charge Counter rates.			
3.28.3 Counter rate transactions will be monitored.			

Other Stakeholders

System / role	Name

Action items

Who	Issue to follow up on

Glossary

counter rate customer – A customer who pays by one of the following methods: U.S. currency, customer’s check (personal or business account), traveler’s check, money order, credit card, official check, cashier’s check, certified check, company coupon, or company prepaid stamp.

counter rates – rates in a rate table to be used by retail counter sites for cash-only customers or customers without an account.

retail counter sites – businesses such as company service counters, partnering businesses, company mini-centers, company ship sites, drop boxes, and on-call pickups. Other future agreements for new sites will not be affected.

counter rate guide -

Testing / Reviews

Verification

It is important to check that requirements meet the stakeholders' real demands. This is called verification. These kinds of tests are what most people call getting feedback during the process. If you keep checking in with your stakeholders during all phases, then it is most likely that they will be satisfied with the final result.

Demands not reflected in the requirements specification are called tacit requirements. Tacit demands are those demands that the user is not aware of or cannot express.

Most requirements are tacit. If you tried to write down all of the requirements, you'd end up with a huge document that no one would use. The goal is to write down what the developers don't already know and can't guess plus a little bit more just to make sure. So the requirements must be based on the type of programmers you work with.

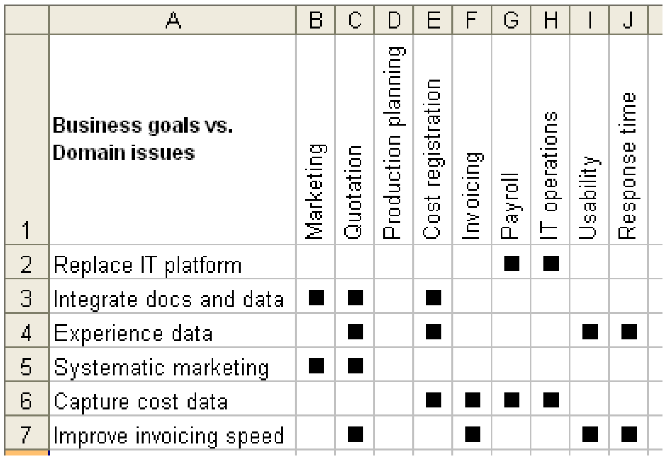
Verification ensures that all the tacit requirements are being met even if they are not documented. As a final set of verification checks, it's good to do a final post-project review and provide feedback for the next project.

Validation

Validation is used to trace the requirements in the documents to the product features and is sometimes called acceptance testing. Verification is also being done at that point.

Real validation includes matching demands to requirements or vice versa to see that you got everything and that everything has a purpose. Tracing product features to requirements is useful to avoid feature creep.

Goal-domain analysis



	A	B	C	D	E	F	G	H	I	J
1	**Business goals vs. Domain issues**	Marketing	Quotation	Production planning	Cost registration	Invoicing	Payroll	IT operations	Usability	Response time
2	Replace IT platform						■	■		
3	Integrate docs and data	■	■		■					
4	Experience data		■		■				■	■
5	Systematic marketing	■	■							
6	Capture cost data				■	■	■	■		
7	Improve invoicing speed		■			■			■	■

5 – Goal-domain tracing

When you match up the business units (domains) against the very high level requirements (business goals) that you have gained, it may be suddenly obvious that something has been forgotten. You may even see a set of business goals that have nothing to do with any of your business units.

Verification

Verification is the process of evaluating a system or component to determine whether the products of a given phase satisfy the condition imposed at the start of that phase. In each verification phase you will get feedback which will tell you whether you are on the right track and if not raise questions for new requirements.

In verification you will want to verify that:

- the features we expressed really meet the needs
- the use cases and requirements we derive from the features truly support the features
- the use cases are implemented in the design
- the design supports the functional and non-functional aspects of the system's behavior
- the code really does conform to the design and the design objectives
- our tests provide full coverage for the requirements and use cases that have been developed.

Verification can take place at every phase of the lifecycle of your project. Milestones can be implemented to set off the verification process but the most important one is going to be the last one when you verify that the customer likes the result.

Use case basics

History

Use cases have become the de facto standard of the OO software development world. But just because they have become the standard doesn't mean that people use them. You can adapt your enterprise system to include them and work to introduce the clarity and detail that they give a structure to by understanding where they fit in.

The use case is a set of related functional requirements initiated by some person or system expressed as a story or script. It was introduced by Ivar Jacobson and others in 1994 and suffered slightly in terminology by being literally translated from Swedish. For instance, the use case description could be called a task story. The actor could be called a task initiator.

UML has added one diagram to support visual modeling of use cases. But the way that the model descriptions in text are written has no standardized style or template. It is up to you to come up with one based on your needs.

Definition

✓ **A process that can be repeated to give value to the business.**

✓ **Anchored by functional requirements and tied to non-functional requirements.**

✓ **If it's a system use case, it's done by the software that is being built.**

The use case is a base set of functional requirements with associated non-functional requirements for a system initiated by someone or a computer system and generates value. That's why there are different use case levels and two types of use cases depending on what kind of system you are talking about. Use cases can be repeated with the same results being generated.

The system use case expresses the current business process so that an application can be built to model it. The business use case expresses a business workflow when an appropriate trigger happens and ends when the value has been created.

The use case name is a label for business functionality and is defined in a use case description to some detail level. It can also be graphically shown as a set of relationships between actors and use case names in a UML use case diagram.

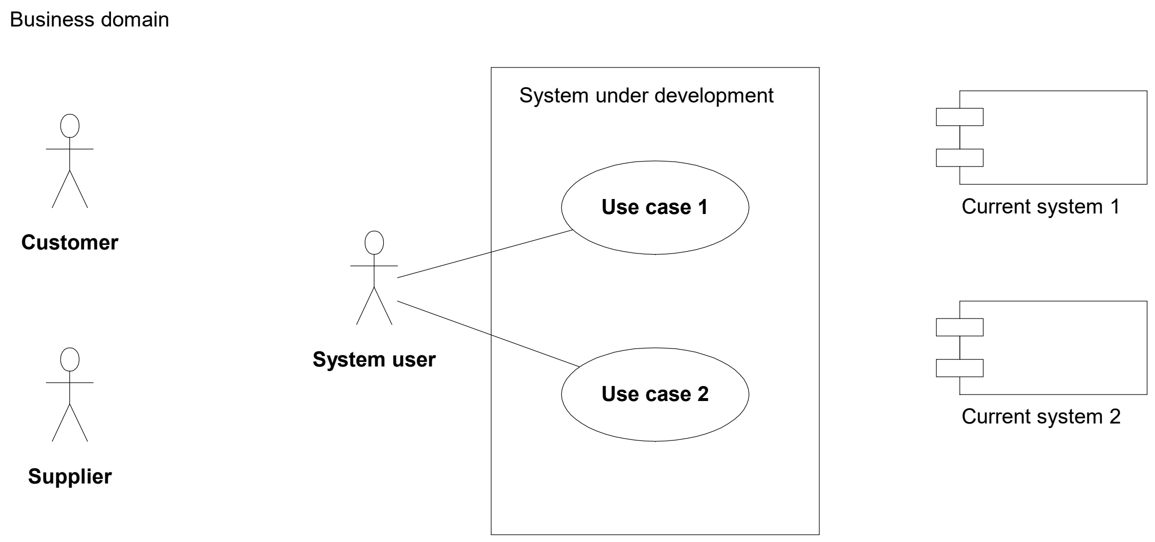
Domains

Another way to categorize the requirements is by domain. If there is a business process that will use the system then it's a business use case. If the system will be doing the task all by itself, it's a system use case. A system use case will replace an entire current business process.

A call center that has to manage 500 calls per hour will write business requirements about managing calls. A system requirement that supports the call center will state that a call center's support application will find a customer's call record in less than 2 seconds or the system will add a follow-up record to an open ticket.

Defining functional requirements in terms of the inputs and outputs of the business activities may be one likely solution but does not work well because the business environment changes often. The business activities to be automated are important to specify clearly and collectively are known as the system domain. This group of business activities is the domain of the system and the software being developed is called either the system or the product.

There is also a domain which can be called the business domain. It is the environment which the company does business in and shows the full workflow of the people, systems, communications, and documents involved. The use case associated with the business domain is sometimes called an organizational use case.



6 - UML diagram showing domains

Use case styles

For a project that gets off to a quick start, you may need to do a minimal set of use cases before the programmers get going or you may need a quick overview to give other people a solid understanding of what the functionality of the system will be. You can then expand that overview to add more and more detail as the need arises. Each level of detail can be expressed in a use case with a different format.

The informal format

The basic level of the use case is a quick description of the use case in a short paragraph. Think of it as a very short story. It's also a good way to provide a high-level summary of more detailed use cases.

The formal format

A more detailed format involves numbering all of the flow of events. If you need more detail, you add pre and post conditions as well as information on quality or SLAs, and alternate paths and extensions. The alternate paths and extensions can also be expressed either informally or formally.

The formal use case process

• find actors	• find use case names
• **diagram** the relationships of the use cases	
• **detail** the course of events for each use case name	
• increase the **detail** of each use case if very important, complex, or high-risk by adding pre-conditions, post-conditions, alternative conditions and flows, and special requirements.	
• **structure** if model is too complex or if needed with «include», «extend», and «generalize».	

Use case names

✓ **Name your use cases with verb-noun syntax.**

✓ **Use the words of the domain**

The use case follows a standard granularity at the scenario level. The grouping and the partial scenario levels can be used but are used later to help structure the use cases for better understanding or communicating a specific point.

Naming a use case uses the syntax of verb-noun and any extra descriptive words following the noun such as:

- view customer data
- delete expired licenses
- track package number
- send an e-mail to newsletter member
- withdraw cash

Candidate use case names (from Peter Coad's *Object Models*) start with verbs that show action such as:

- activate (initialize, initiate, open, start)
- answer (reply, respond)
- assess (appraise, assay, evaluate, value)
- calculate (compute, count, estimate, rate, tally)
- deactivate (close, end, shut down, terminate)
- determine (decide, figure out, observe, resolve)
- find (get, look for, pinpoint)
- measure (bound, gauge, limit)
- monitor (conduct, direct, manage, observe, operate, supervise, watch)
- qualify (characterize, differentiate, discriminate, distinguish, mark)
- select (choose, cull, elect, opt for, pick)

When naming the use case, as well as in any other requirements writing such as in the detail, you want to not rename the business concepts to fit your way of thinking. The documents you produce are for the stakeholders to understand and must use the business language of the stakeholders. Don't go into a foreign land and change the names of the landmarks for your map like the British did when they renamed cities like Mumbai to Bombay.

Also don't go adding detail that isn't a part of the users' requests because you think it is good. Make sure to confirm with the stakeholder about the possibility of a missed requirement and prioritize it correctly. And then keep your technical writing concise. You'll still end up with a long use case name most likely which is normal.

Use case levels

✓ **Aim for the scenario level of granularity**

Determining the level of the requirement and managing the types is what causes the most confusion when it comes to working with requirements. Not everyone thinks by default at the same level. There are some high-level thinkers that see the big picture but are awful at details. And then there are the people that get stuck working out the fine details and get strangled if given enough rope.

Alistair Cockburn (ko' burn), a leading authority on use cases, has names for five different levels of use cases. He attaches symbols to the levels. You can match up these levels with other terms that are used to describe requirements in the following table:

in this manual	Alistair Cockburn's use cases	another requirements system

group	Very high summary	business requirement
group	Summary	goal level
scenario	User goal	domain level
partial scenario	Subfunction	product level
partial scenario	Too low subfunction	design level

The level that fits the best for creating a use case is the scenario level. You can diagram these kinds of use cases the best and provide enough detail in the flow of events. The grouping level use case is good for diagrams but doesn't do well when creating a flow of events. The partial scenario level creates detail suitable for programmers and intricate models but can be useful when completely understanding a complex process.

Actors

✓ **Actors initiate a use case (the trigger of the process).**

✓ **For any two actors, one will have a special use case that the other doesn't do.**

✓ **Actors and security roles both restrict available functionality.**

Use cases introduced the term actor as someone or something that interacted directly with the system under development and triggered the start of the use case scenario. It could be a user in a certain role, the same user in a customer role, or another computer wanting to access information. It could even be the same system telling itself to do something at a scheduled time like run a midnight report.

The actor is the initiator of the use case. The person with that role or system with that permission kicks off the use case and that actor or the system gets some value out of the completion of the use case. The actor maps very cleanly to security roles. Think of the actor as the perception of the initiator from the system's point of view. It is not the same as the person external to the system. A person could act in the manner of several different actors or roles and the system would act appropriately for each type of actor. What matters to the system is that the initiator is able to start and complete use cases.

There are other systems that could be contacted in running the use case but they are not primary actors. Use case diagrams are simpler when you don't show all of the systems that have to be engaged to complete a use case, only the one who started it.

Other people who see use cases as more of an activity diagram instead of a scope diagram use terms like primary, supporting or secondary actors. Their diagrams show systems with interactions between the actors like a flow chart which is not the purpose of the use case diagram. The UML specification does not support the use of arrows to show flow in the use case diagram and use cases have the most value is seeing the general scope of the project.

Exercise – Actor table

Identify the roles or an actor name for each of the people involved in this group of stakeholders who will be using a web-based e-commerce system. Typically there will be about three or four actors that will apply to all of the stakeholders below. Think of the actor as someone who has security rights to do a use case you are providing.

Stakeholder	Actor
customer	
manager	
executive	
visitor	
preferred customer	
returning customer	
customer with a bad credit card	
marketing person	
guest	
unknown user	
administrator	
programmer	
hacker	
search engine	

Use case validation

✓ **Does your use case meet the walk-away test? Or is it a part of another use case?**

✓ **Does the system return to the same state it was in before you started the use case?**

✓ **Can the system functionality be applied to a phone interface?**

✓ **Will the workflow describe only one successful path?**

There are a few principles you can apply to describe a good user-goal level use case.

1. **Closure** – Each use case must end with value to the person / system that initiated it. If you are the person initiating the use case in the system, you should be able to walk up to the system and walk away after the sequence of events was completed. Another person should be able to walk up and repeat the use case. This "**walk-away test**" fails when you try to make something like a user log-on a use case which is a very common mistake if only the user-goal level use cases are being defined. A user log-on is a partial scenario. Closure means that the workflow is capable of being deployed by itself.
2. **Session rule** – A session starts with data in a certain state and at the end of the session, the data is replaced with new data ready for another session. Can you say that the state of the system is back to the way it was before you began? The session rule helps create closure on a workflow.
3. **Don't design –** The use case author should not have any detail about how the task is performed such as pieces of data and specifications of what should be on a screen. They should be referenced though to other sections. The requirements are a reflection of **what** the system should do and not **how** the system should go about doing it. Think in terms of the business language. The question to ask is "could this be applied to another type of user interface?" If you could use a kiosk, a telephone, a business process or a touch screen to get the job done, then you have expressed it correctly. One exception to this rule is if the business has constrained the project to be a web site and you use terms specific to a web site. But don't assume that the project is a web site unless that constraint exists!
4. **One workflow –** The use case should describe one entire goal-oriented workflow without asking about anything that allows for the workflow to take two different successful paths. If there are options for choosing different tasks, then you are probably looking at a grouping of use cases instead of one workflow. Work out the nature of each workflow and break down the grouping into its scenarios.

Exercise –Actors (ATM)

As a class with the instructor as facilitator, use brainstorming to discover the actors in the ATM system. Verify the actors by finding a use case that they do in the next exercise.

Exercise - Use case names (ATM)

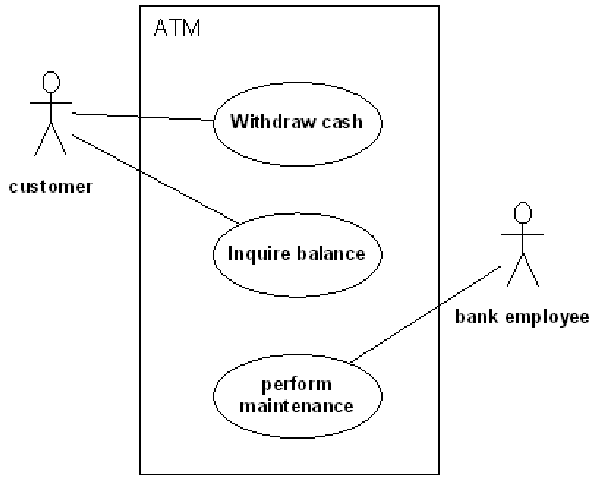
As a class with the instructor as facilitator, use brainstorming to discover the use case names in the ATM system.

Use case diagrams

✓ **Split diagrams into readable sections.**

✓ **Only show actors who initiate use cases on diagrams.**

✓ **Show scope, not activity direction.**



The use case diagram of UML provides the viewer with a quick overview of the use cases being presented. A diagram is simple with the initiator of the use case, or actor, represented by a stick figure. The use case itself is an ellipse with the name of the use case inside. The relationship between the actor and the use case that it can initiate is represented by a solid line without any arrow. Multiple actors can initiate one use case and one actor can initiate multiple use cases.

The entire group of use cases can be enclosed by a rectangle which represents the system being constructed. Other rectangles to show other systems can be represented if multiple systems are involved.

Some people like the idea of supporting actors, or those actors who must be consulted to finish a use case but do not initiate it. These have been represented in diagrams that do not follow UML with arrows to try and show the sequence of events, but confuse the diagram because they look like a primary actor. The diagram is not about the sequence of events, it is about who can do what functional pieces and who initiates it. In this way, it is more a security rights diagram than a flow chart.

Exercise – Use case diagram (ATM)

As a class with the instructor as facilitator, create a use case diagram for the ATM system.

Exercise – Prioritization (ATM)

As a class with the instructor as facilitator, prioritize the use cases of the ATM system using the categories of usage (impact) and value (urgency to fix).

Use case name	Business / Market usage % (3=100-67 2= 66-34, 1=0-33%)	Business / pricing value (3=exec/high, 2=mgmt./med. 1=staff/low)	Result = Market * Value	Total count of top 3 - group

Use case detail

Write the basic flow of events

✓ **Always a “happy path”**

✓ **No if-then-else statements**

✓ **No button click or submit buttons statements**

The basic flow describes the most common scenario in which everything goes according to plan and the actor achieves their goal.

It is also referred to as the normal course of events or “the happy path.”

The first step describes the triggering event for the use case. Each step in the flow of events should be a positive step towards the actor’s goal. And each step should be a simple, declarative statement that describes the actions taken by the actor or the system.

The two basic ways to write the flow is in a dialog where the actor does something and then the system does something in response. It can either be a single column of steps or a two-column script style dialog where the actor and system steps occupy separate columns.

Even though there might be more than one actor that can initiate each basic flow of events, the flow is written with “The actor requests…” instead of listing the actor’s name so that it really doesn’t matter which actor is doing the use case.

In general, don’t use conditional logic in the flow of events. Conditional statements (“if something is true, do this”) are either an extension point which will return the flow of events back to the place where they left or they will take the use case in a different set of steps with a different result. That means that you have a different use case. A small extension point can usually be slipped in to the flow if it’s one step without noting it below as an alternate flow.

Also, business rules are sometimes expressed with if statements and unless they are very, very short rules (age>18), they belong in another section. Combining business rules in one section allows analysts to reuse rules easily.

It is usually a mistake to describe button clicking, menu selecting, tab selecting and other user interface details. This is always better left to experts that know how to design effective user interfaces that support the use cases. Concentrate on the steps necessary to achieving the actor’s goal.

An example:

1. The use case starts when the actor selects Place Order.
2. The system provides the current contents of the shopping cart and prompts the user to enter customer information.

3. The actor enters **customer identification information** (customer id, password, and discount code – or place in Glossary item) and confirms the completion of the entry.
4. The system validates the **customer identification information**. The system calculates the charges, and provides the customer information, payment information, and order details for review. The system prompts for a confirmation.
5. The actor confirms the order.
6. The system validates the method of payment.
7. The system stores the order and updates its status to submitted.

References

✓ **Move out the details that don't functionally matter.**

When writing the flow of events, there will be many types of related information that should be tied back to the flow details. They should all be coded with an id in some way and stored in another section of the requirements document where they can be looked up without clogging up the functional clarity of the flow.

Business rules are extracted and referenced as validity checks during a functional task usually. They appear as a conditional statement that results in a change of data, a change of functional task, or a change to the way something looks.

Screen shots or mock-ups make good references so that the programmers can have a model to start from when designing the forms. Report mock-ups are also excellent to have rough models of.

Data definitions can be extracted when too much detail prevents clear reading of the flow of events. Usually you can name groups of data and then refer to it in a data section where it can be detailed a little more for validity checks, maximum length expected, number of items expected to store, etc.

When messages or error prompts need to occur in a program they can be better managed by extracting them from the flow of control into a section for messages or what programmers call externalized strings. By referring to each message by a code, it allows for updates to be made to the text by the designers or other stakeholders without affecting the use case. It also makes internationalization much easier by creating one document per language using the same codes.

Write the top alternative flows

✓ **Extension points return back where you came from after an optional set of steps.**

✓ **Failure points stop the use case, return you to a different point, or fix the problem and let you continue.**

There are two types of alternative flows and can be broken up into two sections in the document if necessary but usually isn't. There's the interruption of the use case which is when something fails and the programmer will need some assistance to figure out how to handle the error. That requires an alternative flow that is associated with the number of the basic flow that it came from.

Then there are the extension points that identify specific places in the flow of events where the use case may be extended by other use cases. In these cases, the flow continues on after the extension point flow with the basic flow. Again, the extension is identified with the number where the flow diverged from the basic flow.

Typical failure points include:

- Internal System Failure – look for situations where an internal system failure might occur.
- Actor incorrect - look for situations in which the actor performs a step incorrectly.
- Actor inactivity - look for situations that are sensitive to non-response by the actor (primary or supporting).
- Validation failure - anytime you have a validation step, you will need to handle validation failures.
- Performance failure – look for situations where the system must do something within a certain timeframe.
- Included Use Case Failure - anytime you execute a use case from within a use case, the included use case can fail.

Numbering alternative steps

Because it is important to map the alternative steps back to the place in the main flow of events that was where it came from, you need to number both in a way that makes sense. There are several ways to do that.

One way is to just do a backwards reference. The alternative step is named and shows the place where it came from. The steps are just placed in text format underneath.

Zip code failure (5) – Prompt for another zip code and continue at 5.

Another way is to code the alternative flow item with the number that it came from and a letter since there could be more than one exit point from that number. The more format steps are listed under the item and are also numbered so they can be referenced if necessary.

5a. Zip code failure

5a1. The system prompts for another zip code. The user enters a valid zip code.

5a2. The systems continues the use case at step 5.

Capture pre-conditions and post-conditions

✓ **Pre-conditions block the use case from doing the first step.**

✓ **Pre-conditions validate the state of the software before anything happens.**

✓ **Post-conditions state the testable results.**

Pre-conditions are states which must be true before the use case can even begin e.g. package status is in a shipped state. This is about the state of the software and validation checks. It is not about the environment of the software that is not under your control nor is it about the security rights where you might think you need to log in first before you do your use case. It is also not about a validation check that is made during the flow of the use case.

The log-in is a part of the use case flow of events and not usually a pre-condition. Only when the system has no use cases for an unauthenticated user could you put a security validation as a pre-condition. A pre-condition will be more like a value such as in the Cool Down House use case. The value that needs to be checked is the temperature that the software senses and can't be below 78 degrees before the use case is valid to begin. In general, task level use cases will use pre-conditions more than workflow/scenario level use cases.

Post-conditions are the results or states of objects attained after the use case is completely processed. e.g. Toll free customer added. These are usually important points that you want to double-check since they are critical to the system. Systems that work with health or the life support will assert their post-conditions more than others. The activity which is to be checked in the post-condition will be expressed in the flow of events.

Post-conditions can also be used as test results and can be a basic set or an exhaustive set depending on the necessity of the testing.

Document any special requirements

✓ **Put things like SLAs and location or time needs in a special category.**

Because the use case is sometimes the only document that captures requirements, the non-functional requirements must be given a place to be stated. A special section of the use case named additional requirements, or notes, or any other heading you would like. It is used to capture things like response time, timing, availability, accuracy, recovery time, memory usage, frequency of occurrence, prioritization, speed, location needs, or throughput requirements.

Exercise – Use case detail (ATM)

As a class with the instructor as facilitator, summarize and write the detail for the Withdraw Cash use case of the ATM system.

Use case structuring

The grouping use case

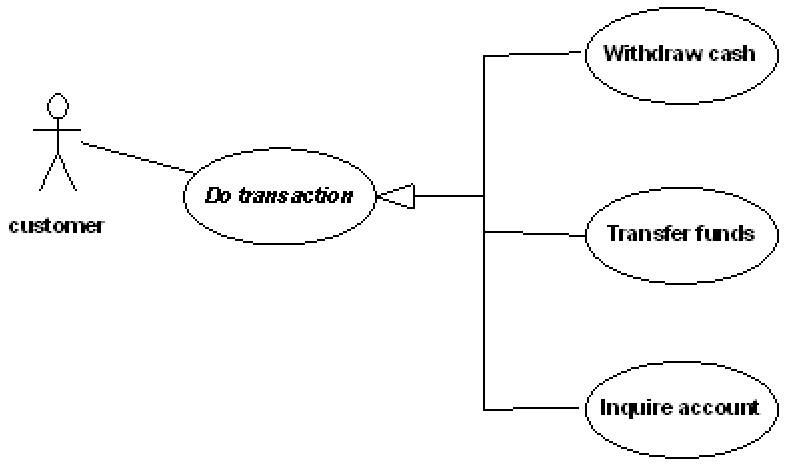
✓ **Use groupings when it cleans up a use case diagram and makes it easier to understand.**

✓ **Often used to show options for reporting use cases.**

The amount of detail need for each use case varies with the use case's complexity and risk. The more complexity or risk, the more detail.

Writing paragraph style descriptions is a valid quick style of capturing the functionality of the system. An index made up of these summaries makes for a quick guide to the system when accompanied by the details that you describe for more formality or understanding of the system. If the system is small and only requires a brief description to gain agreement on, then these are all that is necessary for use cases.

The grouping use case has an advantage in diagramming by being able to represent groups of use cases. But it can be very difficult to write any more than a very informal description of one and should be left to the more specialized use cases to specify the details.



A <<generalize>> stereotyped relationship shows use cases pointing to another abstract (grouping) use case using its basic sequence of events in some way so that they can specialize the use case. The arrow and line are different than the line that was used for the other two special relationships that used the dependency arrow. The same line and arrow is used in the class diagram to express inheritance but that word is reserved for objects and not use cases.

This is a very hard type of use case to use and it never works well when it comes to the formal text version. Informal text versions are the only method that makes sense. The diagram might be useful for talking about how several use cases are all similar in order to clarify the meaning of them. In the example, the use case Do Transaction has been specialized into three separate use cases that take similar courses of action. They all require a logon, access the bank records, and can print a receipt.

Exercise - grouping

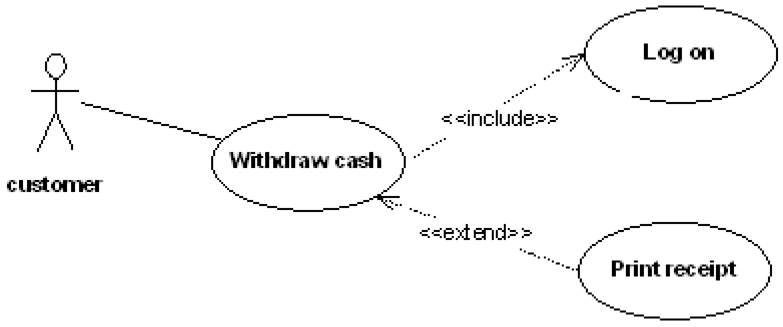
A. Manage Contractor Management System (CMS)
    a. Contractor Maintenance
    b. Company Maintenance
    c. Job management
        i. Estimate jobs
        ii. Schedule jobs
            1. Price jobs
                a. Convert estimates into job orders
                b. Apply discounts
                c. Record pricing details
            2. Schedule contractor
        iii. Dispatch jobs
        iv. Close out jobs
    d. Pricing
    e. Accounting
    f. Querying
    g. Reporting

The partial scenario use case

✓ **Use partial scenarios when users expect it.**

✓ **Use partial scenarios when options are important to see.**

Partial scenarios use cases are often a part of the user-goal use case and can be shown in relationship to them.

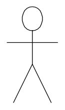
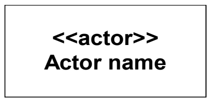
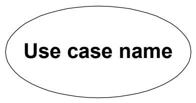
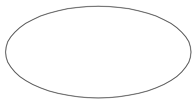
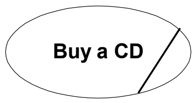


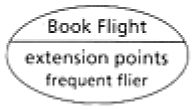
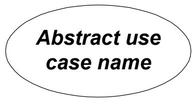
An <<include>> stereotyped dependency which points to what it is dependent on, means that it is required to be done in order for the use case to complete. In the example, the Log on use case is a required part of the Withdraw cash use case.

An <<extends>> stereotyped dependency which points to the use case that will initiate it, means that it is optional to be done for the primary use case to complete. In the example, the Print receipt use case will be an option that will happen if the customer wants it when they go through the Withdraw cash use case.

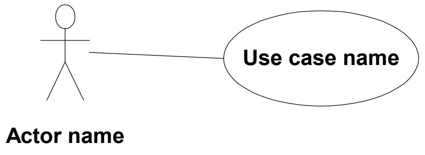
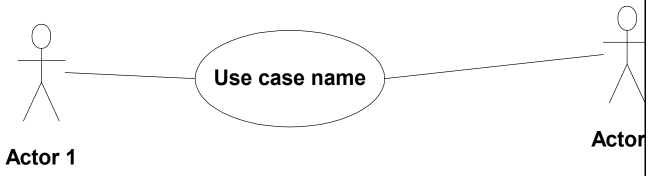
UML - Use case diagram symbol reference

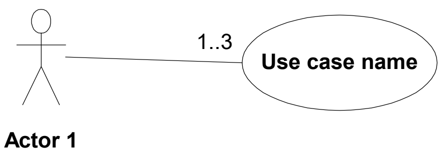
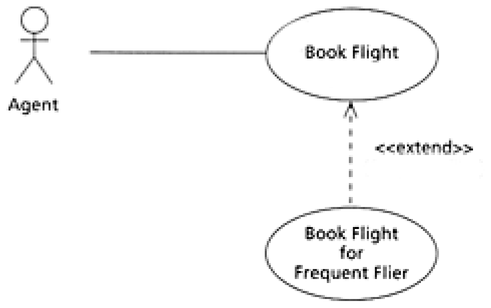
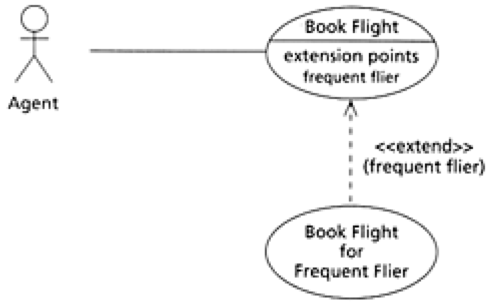
Icons

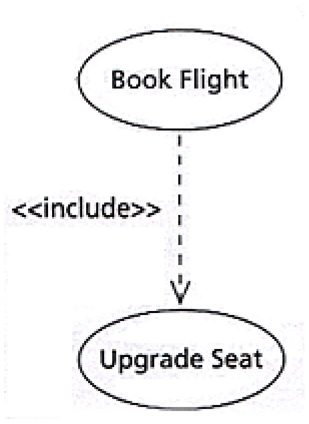
 or  or any other icon	Actor	An actor defines a coherent set of roles that users of an entity can play when interacting with the entity. An actor may be considered to play a separate role with regard to each use case with which it communicates.
 or 	Use case	A use case is shown as an ellipse containing the name of the use case. An optional stereotype keyword may be placed above the name and a list of properties included below the name. As a classifier, a use case may also have compartments displaying attributes and operations. The behavior of a use case can be described in several different ways, depending on what is convenient. Plain text is used often, but state machines, operations, and methods are examples of other ways of describing the behavior of the use case. Sequence diagrams can be used for describing the interaction between use cases and their actors.
	Use case-business	A use case that represents a business process and not a system function is shown with a line drawn through it on the right side.

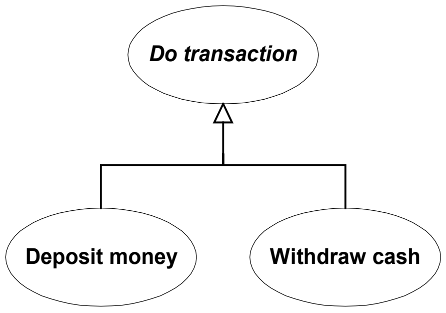
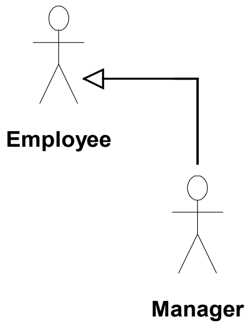
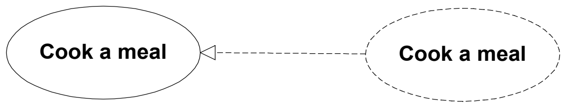
	Use case	Extension points may be listed in a compartment of the use case with the heading **extension points**. The description of the locations of the extension point is given in a suitable form, usually as ordinary text, but can also be given in other forms, like the name of a state in a state machine, or a pre-condition or a post-condition. The ellipse may contain or suppress compartments presenting the attributes, the operations, and the extension points of the use case.
	Abstrac t use case	An abstract use case is one that does not have use case realizations. This means that it is not meant to be used as a workflow but as a template for other workflows. This would be a way to note a grouped level use case.

Relationships

	Association	The participation of an actor in a use case; that is, instances of the actor and instances of the use case communicate with each other. This is the only relationship between actors and use cases.
	Association	These actors participate in the use case but UML notation does not describe which one initiates the use case. Non-standard styles

		dictate that actors on left initiate and actors on the right of the use case participate but do not initiate. Most people presume that any actor initiates the use case.
	Association	The ellipse may have end adornments such as multiplicity.
 Agent Book Flight <<extend>> Book Flight for Frequent Flier or  Agent Book Flight extension points frequent flier <<extend>> (frequent flier) Book Flight for Frequent Flier	Dependency	Extend – An extend relationship from use case A (Book Flight for Frequent Flier) to use case B (Book Flight) indicates that an instance of use case B may be augmented (subject to specific conditions specified in the extension) by the behavior specified by A. The behavior is inserted at the location defined by the extension point in B, which is referenced by

		the extend relationship. The condition of the relationship is optionally presented close to the key-word.
	Dependenc y	An include relationship from use case Book Flight to use case Upgrade Seat indicates that an instance of the use case Book Flight will also contain the behavior as specified by Upgrade Seat. The behavior is included at the location which defined in Book Flight. (I don't agree!) An include relationship between use cases is shown by a dashed arrow with an open arrow-head from the base use case to the included use case. The arrow is labeled with the keyword «include».

 and 	Generalizat ion	A generalization from use case C to use case D indicates that C is a specialization of D. If an actor can do the parent use case, then it can also do the child. Also, a generalization from an actor A to an actor B indicates that an instance of A can communicate with the same kinds of use-case instances as an instance of B. A generalization between use cases or actors is shown by a generalization arrow, a solid line with a closed, hollow arrow head pointing at the parent.
	Realization	A use case realization is the relationship of the description of behavior in the use case and the models that

		represent a specific path through that use case. If you have a need to show a link between the use case and various sequence and state diagrams, this will provide the visual means to show it. Another term that could be applied here is a test script that added data from the test case (use case).

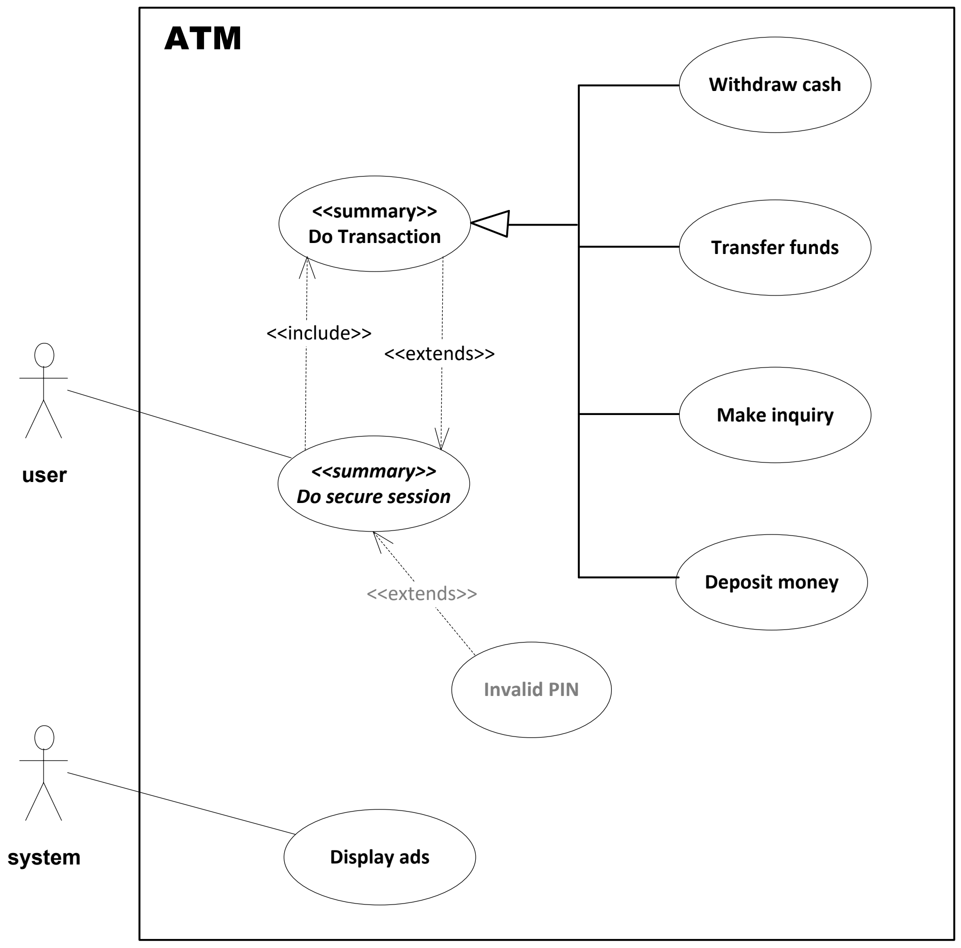
Solutions

Exercise - Requirement levels

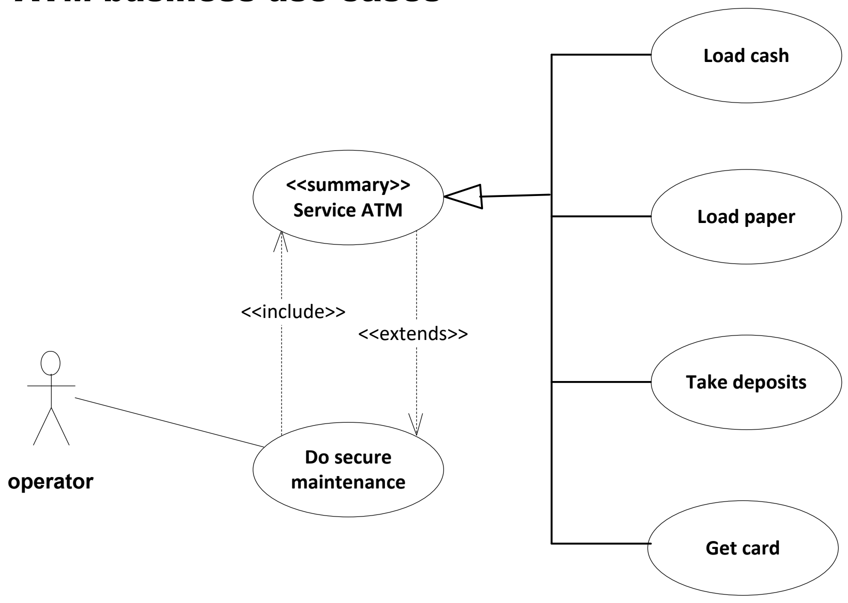
Structure these requirements into the levels of granularity.

Requirement	Level – group, scenario, partial scenario/task, design
W1 – The system will count hits on each web page. (for a report)	partial scenario
W2 – The admin shall create a report that shows the number of people to visit web pages.	scenario
W3 – The managers will manage marketing information on the web site.	group
W4 – The system users will report on user tracking information throughout the web site.	scenario
W5 – The system will use a hit counter made up of individual digits.	design recommendation
H1 – The clerk shall check in a guest.	scenario
H2 – The clerk shall enter a valid password.	partial scenario
H3 – The clerk shall log in to the system. (to check in a guest)	partial scenario
H4 – The admin will manage hotel guests' accounts.	group
H5 – The admin will report on occupancy levels.	scenario

ATM use case diagram solution



ATM business use cases



ATM informal use case solutions

Solutions are written in a paragraph form. Your class exercise solution was written in the more detailed outline form.

S1 – Do Transaction (group)

This use case groups UG2 - Withdraw cash, UG3 - Deposit Money, UG4 - Transfer Funds, and UG5 – Make Inquiry.

A transaction is started with a UG1 – Do Secure Session when the customer chooses a transaction type from a menu of options. The customer will be asked to furnish appropriate details (e.g. account(s) involved, amount). The transaction will then be sent to the bank, along with information from the customer's card and the PIN the customer entered.

If the bank approves the transaction, any steps needed to complete the transaction (e.g. dispensing cash or accepting an envelope) will be performed, and then a receipt will be printed. Then the customer will be asked whether he/she wishes to do another transaction.

If the bank reports that the customer's PIN is invalid, the Invalid PIN extension will be performed and then an attempt will be made to continue the transaction. If the customer's card is retained due to too many invalid PINs, the transaction will be aborted, and the customer will not be offered the option of doing another.

If a transaction is cancelled by the customer, or fails for any reason other than repeated entries of an invalid PIN, a screen will be displayed informing the customer of the reason for the failure of the transaction, and then the customer will be offered the opportunity to do another.

The customer may cancel a transaction by pressing the Cancel key as described for each individual type of transaction below.

All messages to the bank and responses back are recorded in the ATM's log.

S2b – Service ATM (grouped)

This use case groups UG6b – Load cash, UG7b – Load paper, UG8b – Take deposits, and UG9b – Get Card.

UG1 – Do secure session

A session is the part of the transaction that is started when a customer inserts an ATM card into the card reader slot of the machine. The ATM pulls the card into the machine and reads it. (If the reader cannot read the card due to improper insertion or a damaged stripe, the card is ejected, an error screen is displayed, and the session is aborted.) The customer is asked to enter his/her PIN, and is then allowed to perform one or more transactions, choosing from a menu of possible types of transaction in each case.

After each transaction, the customer is asked whether he/she would like to perform another. When the customer is through performing transactions, the card is ejected from the machine and the session ends. If a transaction is aborted due to too many invalid PIN entries, the session is also aborted, with the card being retained in the machine.

The customer may abort the session by pressing the Cancel key when entering a PIN or choosing a transaction type.

UG2 – Withdraw cash

A withdrawal transaction asks the customer to choose a type of account to withdraw from (e.g. checking) from a menu of possible accounts, and to choose a dollar amount from a menu of possible amounts. The system verifies that it has sufficient money on hand to satisfy the request before sending the transaction to the bank. (If not, the customer is informed and asked to enter a different amount.) If the transaction is approved by the bank, the appropriate amount of cash is dispensed by the machine before it issues a receipt. (The dispensing of cash is also recorded in the ATM's log.)

A withdrawal transaction can be cancelled by the customer pressing the Cancel key any time prior to choosing the dollar amount.

UG3 - Deposit Money

A deposit transaction asks the customer to choose a type of account to deposit to (e.g. checking) from a menu of possible accounts, and to type in a dollar amount on the keyboard. The transaction is initially sent to the bank to verify that the ATM can accept a deposit from this customer to this account. If the transaction is approved, the machine accepts an envelope from the customer containing cash and/or checks before it issues a receipt. Once the envelope has been received, a second message is sent to the bank, to confirm that the bank can credit the customer's account - contingent on manual verification of the deposit envelope contents by an operator later. (The receipt of an envelope is also recorded in the ATM's log.)

A deposit transaction can be cancelled by the customer pressing the Cancel key any time prior to inserting the envelope containing the deposit. The transaction is automatically cancelled if the customer fails to insert the envelope containing the deposit within a reasonable period of time after being asked to do so.

UG4 - Transfer Funds

A transfer transaction asks the customer to choose a type of account to transfer from (e.g. checking) from a menu of possible accounts, to choose a different account to transfer to, and to type in a dollar amount on the keyboard. No further action is required once the transaction is approved by the bank before printing the receipt.

A transfer transaction can be cancelled by the customer pressing the Cancel key any time prior to entering a dollar amount.

UG5 – Make Inquiry

An inquiry transaction asks the customer to choose a type of account to inquire about from a menu of possible accounts. No further action is required once the transaction is approved by the bank before printing the receipt.

An inquiry transaction can be cancelled by the customer pressing the Cancel key any time prior to choosing the account to inquire about.

UG10b – Do secure maintenance

The operator is able to do system maintenance while the system is operating which requires the use of a key and special bank card. Once entry is gained, the operator may do any of the maintenance operations of S2b – Service ATM. When the maintenance is complete, the operator removes their card and locks the machine with their key.

UG11 – Display ads

The system starts a timer once any secure session use case is complete or on startup. When the timer reaches 30 seconds, this use case starts. A welcome screen is displayed.

E1 - Invalid PIN (extension)

An invalid PIN extension is started from within a transaction when the bank reports that the customer's transaction is disapproved due to an invalid PIN. The customer is required to re-enter the PIN and the original request is sent to the bank again. If the bank now approves the transaction, or disapproves it for some other reason, the original use case is continued;

otherwise the process of re-entering the PIN is repeated. Once the PIN is successfully re-entered, it is used for both the current transaction and all subsequent transactions in the session. If the customer fails three times to enter the correct PIN, the card is permanently retained, a screen is displayed informing the customer of this and suggesting he/she contact the bank, and the entire customer session is aborted.

If the customer presses Cancel instead of re-entering a PIN, the original transaction is cancelled.

Resources

Writing Effective Use Cases, Alistair Cockburn; Addison-Wesley Professional, 2000 – by far the best practical guide to developing requirements around the central theme of a use case.

http://alistair.cockburn.us/ - Cockburn's web site

Use case template

Customize this form for your use. Eliminate anything that causes useless work. Add items that communicate better or help you understand the requirements.

Name		Number	
Description			
Author		Edited Date	
Tester		Tested Date	
Actors			
Preconditions			
Course of Events			
Postconditions			

Alternatives / Exceptions	
Notes	

ATM user requirements

The software to be designed will control a simulated automated teller machine (ATM) having a magnetic stripe reader for reading an ATM card, a customer console (keyboard and display) for interaction with the customer, a slot for depositing envelopes, a dispenser for cash (in multiples of $20), a printer for printing customer receipts, and a key-operated switch to allow an operator to start or stop the machine. The ATM communicates with the bank's computer over an appropriate communication link (ATMNet) which uses an internal protocol.

The ATM handles one customer at a time. At the time the customer first uses the machine, there is some rotating advertising and other information on the screen. A customer inserts an ATM card and enters a personal identification number (PIN). The data on the card and the PIN are sent to the bank for validation as part of each transaction. The customer will then be able to perform one or more transactions. The card will be retained in the machine until the customer indicates that there are no further transactions, at which point it will be returned - except as noted below. After the card is returned, the screen returns to the idle state where a greeting is displayed along with bank information, date and time, and customizable messages.

The ATM must be able to provide the following services to the customer:

1. A customer must be able to make a cash withdrawal from any suitable account linked to the card, in multiples of $20.00. Approval must be obtained from the bank before cash is dispensed.
2. A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.
3. A customer must be able to make a transfer of money between any two accounts linked to the card.
4. A customer must be able to make a balance inquiry of any account linked to the card.

A customer must be able to abort a transaction in progress by pressing the Cancel key instead of responding to a request from the machine.

The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. Ordinarily, a transaction will be considered complete by the bank once it has been approved. In the case of a deposit, a second message will be sent to the bank indicating that the customer has deposited the envelope. (If the customer fails to deposit the envelope within the timeout period, or presses cancel instead, no second message will be sent to the bank and the deposit will not be credited to the customer.)

If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back.

If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account ("to" account for transfers).

The ATM will have a key-operated switch that will allow an operator to start and stop the servicing of customers. After turning the switch to the "on" position, the operator will be required to verify and enter the total cash on hand. The machine can only be turned off when it is not servicing a customer. When the switch is moved to the "off" position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc.

The ATM will also maintain an internal log of transactions to facilitate resolving ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the ATM is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected), for the dispensing of cash, and for the receiving of an envelope. Log entries may contain card numbers and dollar amounts, but for security will *never* contain a PIN.