Requirements Elicitation

Notes by mainly Jo Anne Atlee,
with modifications by Daniel Berry
dberry@uwaterloo.ca

Fall 2004

Outline

- Most Important Aspect of RE
- Who are the stakeholders?
- Main Task — Examine Project Viability

- Job of Requirements Analyst
  - Understand problem from each stakeholder's point of view.
    - Review documentation
    - Observe current system
    - Questionaires and Interviews
    - Apprenticeship
  - Extract the essense of the stakeholders' requirements
    - Interpreting stakeholders' descriptions of requirements
    - Building models
  - Invent better ways to do the user's work
    - Ask why documented requirements are desired
    - Consider giving the user more creative control over his or her
    - Brainstorm to invent undreamed of requirements
  - Negotiate a consistent set of requirements
    - Key Aspects of Requirements Negotiation
  - Record results in an SRS

- What Can Go Wrong in Elicitation and the SRS?
- Other Techniques
  - PIECES
  - Social and Organizational Factors
  - Ethnographic Analysis
  - Joint Application Design
  - Names and Norms
  - Gause & Weinberg Ideas

Most Important Aspect of RE

What is the most important aspect of the requirements process?

The two leading candidates:

- the Software Requirements Specification (SRS) document
- the process of negotiating requirements that are agreed to by all stakeholders.

Elicitation and Brainstorming

Daniel M. Berry

Definition

“to elicit”

means

“to bring out, to evoke, to call forth”

In this case, information pertaining to requirements

Purpose of Elicitation -1

The purposes of elicitation is to get information about:

- **the domain model from which the requirements are written**
- **the requirements from which system is developed**

Purpose of Elicitation -2

You must get information out of clients' minds without damaging the clients or their minds!

Many times this information does not come out easily.

- **The clients do not know it themselves.**
- **The clients do not want to let it out (subconsciously).**

Purpose of Elicitation -3

Elicitation is a human activity involving interaction between human beings:

- **clients**
- **users**
- **systems analysts**
- **systems developers**

Purpose of Elicitation -4

If you cannot do the human interaction right, you ain't gonna be able to elicit, no matter what technology and methods you use.

Technology and methods might help, but they can also get in the way.

Skills -1

The skills needed for elicitation are:

- **identifying contexts**
- **spotting ambiguities**
- **interviewing**
- **brainstorming**
- **facilitating**
- **getting people to open up**
- **spotting equivocation**
- **inculcating guilt**

Only the first two are not human interaction!

Who are the stakeholders?

- Client — person paying for the software to be developed
- Customer — person who buys software after it is developed
- Users (of both the current and future systems)
- Domain Experts — experts who know the work
- Software Engineer — technology expert
- Inspectors — experts on government and safety regulations
- Market Reseachers
- Lawyers
- Experts on Adjacent Systems

Client — person paying for the software to be developed

This is the ultimate stakeholder. By paying for development, the client has the last say in what the product does, how it does it, and how elaborate or sparse it is. In some sense, by being willing to pay for the development, the client demonstrates just how interested he or she is in the product.

If you are developing in-house software, the client is probably the manager of the product's users — since his or her employees will be the primary beneficiaries, it is reasonable for him or her to pay for the project

If you are developing software for the mass market, then the client may be your marketing department.

Customer — person who buys software after it is developed

You have to understand the customer's needs well enough to build a product that he or she will find useful and buy. Sometimes the customer and the user are the same; othertimes, the customer is an office manager who buys software for his or her staff.

For what requirements will he or she pay? Which are trivial or are excessive?

The customer should always be represented by a stakeholder who is active on the project; if there are many customers, there needs to be a customers' representative.

Users (of both the current and future systems)

These are the experts on the existing system, that tell you which features to keep and which need improvements. Alternatively, they could be experts on competitors' products, and can give suggestions about how to build a superior product. They may have special needs or requirements to be satisfied, e.g., regarding useability and training. You may want to consult special-interest groups: users with disabilities, users who have computer phobias, expert users, etc.

Domain Experts — experts who know the work

The experts you need are familiar with the problem that the software must solve. Examples include financial experts for financial packages, aeronautical engineers for aircraft navigation systems, meterologists for software that models the weather, etc. These people can contribute to the requirements of the product and know about kind of environment the product will be exposed to.

Software Engineer — technology expert

You need someone who can ensure that the project is technically and economically feasible. You need someone who can accurately estimate the cost and development time of the product. You need someone who can also educate the customer about innovative hardware or software technologies, and who can recommend new functionality that takes advantage of these technologies.

Inspectors — experts on government and safety regulations

You need someone who is familiar with government and safety regulations that might affect the project's requirements. These include safety inspectors, auditors, technical inspectors, government inspectors.

Market Reseachers

A market researcher may assume the role of client, if the software is being developed for the mass market and there is no identifiable customer. Market researchers are experts who have conducted surveys to determine future trends and potential customers' needs.

Lawyers

You need someone familiar with legal requirements. Also consider consulting experts on standards that are relevant to the product.

Experts on Adjacent Systems

These experts know about the interface for the adjacent system, and thus will know if there are any special demands for interfacing with the product. They may also have an interest in the product's functionality, e.g., if the product can help the adjacent system do its job better, by providing information it can use.

Main Task — Examine Project Viability

One of the first tasks is to learn enough about the project to decide whether or not it makes good business sense to begin doing the project. For some reason, it is very difficult to cancel a project once it is underway. The more resources that a project has consumed, the harder it is for it to be cancelled. Most managers would rather stick with a dead-end project, than cancel it — even if it is more expensive in the long run to stick with it. To cancel a project is to admit error, which many managers are loath to do.

Ed Yourdon has written all about Death March projects.

Determining viability requires examining the product's:

- purpose,
- business advantage,
- costs vs. benefits,
- feasibility,
- scope,
- required resources,
- requirements constraints, and
- risks.

Purpose

What does product do?

The product purpose is the highest-level customer requirement. It is the business need. All other requirements must contribute in some way to the purpose.

Business Advantage

Why build the product?

The purpose of the product should be not only to solve the problem, but also to provide a business advantage. How will the product help the work?

Costs vs. Benefits

How **much** will the product help our work?

How much will it cost to develop and operate the product?

Naturally, if there is an advantage, you must be able to measure it in order to demonstrate that product achieves the advantage. Vague statements of advantage are open to mis-interpretation and misunderstandings between what the customer thinks he or she is going to get and what the software developer provides.

Is the advantage greater than the cost of constructing the product? The product may offer some business advantage that benefit the client. However, if the cost is going to be greater than the benefit, we might as well halt the project now.

Making this decision requires some way of estimating

- the cost of the project, and
- the benefit of the project,

in what is called a cost-benefit analysis.

Feasibility

Feasibility analysis is concerned with

- technical feasibility and
- economic feasibility

One of the reasons for stating measureable requirements early on is to be able to answer questions about feasibility.

Does the organization have the the skills needed to build and operate the product? Whether or not the project is technically feasible, it is necessary also to know if the organization has the resources and experise to construct the product.

Scope

Is there agreement on the product's scope, i.e., the product's purpose and the system's boundaries.

How much of the work will be done by the system-to-be-developed?

How much of the work will be done by adjacent systems?

We need this information to be able to obtain cost and time estimates.

Required Resources

What are the required resources, i.e., money, time, and personnel?

How do they compare with available money, time, and personnel?

If the latter are smaller than the former, we should not even start the project.

Requirement Constraints

Are there constraints that will restrict the system's requirements or how these requirements are elicited?

These constraints include

- solution constraints:
  - mandated designs,
  - mandated adjacent systems, and
  - mandated COTS (commercial off-the-shelf) components,
- time constraints, and
- budget constraints.

Solution Constraints

- mandated designs

  Designing the solution before knowing all of the requirements is problematic and even deplorable, but there may be some overriding reason — marketing, cultural, managerial, political, expectations, financial — for accepting only one design solution, e.g., client-server arch, must run on PC or Palm Pilot, etc.

- mandated adjacent systems

  Interfaces to adjacent systems are constraints on the product.

- mandated COTS components

  There may or may not be good reasons to insist on using COTS. Now is the time to come to consensus about whether using COTS is wise or even necessary.

Time Constraints

Deadlines are sometimes imposed to meet a window of opportunity, to coincide with coordinated releases of related products, or to beat the release of a completing product. Deadlines can have an effect on whether any fancy features make their way into the product. A deadline is not the same as an estimate of the required time. They are arrived at independently and are based on different reasons. They may even conflict.

Budget Constraints

The money available can affect whether certain features are feasible. Also, it decides how much many a client is willing to spend and can give an indication of how badly he or she wants the product.

Risks

Are there any high-probability or high-impact risks that would make the project infeasible?

Such risks include absense of clear purpose, little or no agreement on requirements, unmeasureable requirements, rapidly changing requirements, etc.

Project Viability

To summarize:

The idea is to gather enough information to make an objective decision as to whether or not to proceed with the project, i.e., whether or not to accept a project from a client or your marketing department.

Job of Requirements Analyst

1. Understand the problem from each stakeholder's point of view.
2. Extract the essense of the stakeholders' requirements.
3. Invent better ways to do the user's work.
4. Negotiate a consistent set of requirements.
5. Record the results in an SRS.

First some detail and then some more detail...

Understand problem from each stakeholder's point of view

Learn about the problem. As you work with each stakeholder, try to understand his or her requirements and the rationale behind them.

Extract the essense of the stakeholders' requirements

In particular, try to see beyond each stakeholder's description of the requirements, which may be expressed in terms of solutions, to the essense of the problem.

Invent better ways to do the user's work

The idea here is that once you have an understanding of what work the users are trying to accomplish, you may be in a position to suggest requirements that would help them, either because you are aware of technology that would help them or because you identify patterns in their work of which they are not aware.

Negotiate a consistent set of requirements

It goes without saying that the stakeholders must agree on a single set of requirments, and that client and users agree that this is the product that they want. The stakeholders are the ones who will decide whether the final product is acceptable or not.

Record results in an SRS

After all of the above is done, it is necesary to record the agreed to requirements in a specification document, the SRS.

Now it is time to examine each of these in still greater depth.

Popping back up to the tasks of a requirements analyst...

- **Understand the problem from each stakeholder's point of view**
- Extract the essense of the stakeholders' requirements.
- Invent better ways to do the user's work.
- Negotiate a consistent set of requirements.
- Record the results in an SRS.

Understand Problem

- Why Analyze Existing System?
- Steps in Analysis:
  - Review Documentation
  - Observe Current System
  - Questionaires and Interviews
- Apprenticeship

Why Analyze Existing System?

- Postulating requirements from first principles? *It's often NOT the best approach. Often there are many systems out there that are reasonably similar to what you are trying to build.*
- These "brainstorming"-type techniques are especially good for new, unproven technologies ...
- ... but if you are building a "new & improved" version of an older system, you must sit down and carefully analyze the old one:
  - $\rightarrow$ *What is used, what isn't, what's missing.*
  - $\rightarrow$ *What works well, what doesn't.*
  - $\rightarrow$ *How the system is used, how it was intended to be used, what new ways we want it to be used.*

Why Analyze Existing System?

- Risks if you don't:
  - → *Users may become disillusioned with new system if it is too different or doesn't do what they want. Nostalgia for old system.*
  - → *Not appropriately take into account real usage patterns, human issues, common activities, relative importance of tasks/features*
  - → *Miss obvious possible improvements (features that are missing or don't currently work well).*
  - → *Find out which "legacy" features can/can't be left out.*

Steps in Analysis of Existing Systems

1. **Review available documentation** — user docs, development docs, requirements docs, internal memos, change histories, *etc.*

   Of course, often these are out of date, poorly written, wrong, etc.*, but it's a good starting point.*

2. **Observation** — Go out into the field, and observe the "IT specialists in the mist".

   Ideally, you are silent observer, at least initially. Otherwise, you risk getting a non-standard view of what is usually done. Later on, you can start asking questions OR interview people OR use questionnaires.

3. **Questionnaires** — Based on what you know now, circulate some questionnaires.

   Most useful when large number of users and you want answers to specific questions. “How often do you use feature XXX?” “What are the three features you would most like to see?”

4. **Interviews** — At the end, when you have a better idea of what you will be doing and have some good questions that requires detailed answers.

   You won’t be wasting other people’s time, or your own. This is very labour intensive!

Review documentation

Review all available documentation. If there exists an automated system, review its documented specifications and user manuals. If the existing system is a manual system, review any documented procedures that the workers must follow.

The goal is to gain knowledge of the system before imposing upon other people's time, before bothering the stakeholders.

Observe current system

Documentation rarely describes a system completely, and it often is not up to date. The current operation of the system may differ significantly from what is described.

Besides, no matter how bad a reputation the existing system has for doing the work, the system is not worthless. It contains a lot of useful functionality that should be included in any future system. The objectives of observing the current system is to identify what aspects to keep and to understand the system you are about to change.

EXAMPLE: Here is a non-computer example of the importance of understanding benefits of the existing system:

Hot-air hand dryers in washrooms.

Great idea! Eliminate paper waste, save trees, cleaner washrooms.

It was not long after installing them that people discovered that certain functions served by paper towels were not served by the new system:

- drying one's face
- clean up spills
- dry one's glasses
- blow one's nose

Nowadays, what do you normally find in washrooms with hot-air dryers?

Paper towel dispensers!

Perhaps if someone had spent one day in a washroom observing how paper towels were being used, he or she would have discovered the secondary functions that paper towels provide that hot-air dryer do not.

When new computer systems are implemented, remnants of the old system often linger on, because the designers of new system overlooked a function provided by the old system.

Questionaires and Interviews

Questionaires are useful when information has to be gathered from a large number of people, particularly users.

Questionaires are useful also when the answers to questions need to be compared or corroborated.
There are a couple of points about questionnaires and interviews I want to stress.

- interview all stakeholders

  A common mistake is to interview only the client and the user and to neglect the other stakeholders, who may have definite views about what the system should do.

- ask problem-oriented questions

  If questions are too detailed and are solution specific, they may miss the user's real requirements.

  For example, consider a camera sales clerk asking questions of novice photographer,

  "Do you want shutter, aperture priority features, or both?"

  vs.

  "Will you be taking action shots, still pictures, or both?"

- interview groups of people together to get synergy.

  Users cannot think of everything they need when asked individually, but will recall more requirements when they hear others' needs. This interaction is called synergy, the effect by which group responses outperform the sum of the individuals' responses.

  For example, suppose I ask one person in the class to recall 10 good jokes.

  "Justin, tell us 10 good jokes."

  Most people, other than expert comedians or total hams, would freeze, possibly not able to tell even one joke.

Suppose instead, I invited the entire class to participate, and I got the ball rolling by telling a couple of jokes myself. My jokes would likely stimulate your memories, and your followup jokes would stimulate your memories further and help each of you to recall many more jokes. Together, we'd have close to 100 jokes in an hour. And most of you will have heard maybe 80% of them before. Most people know lots of jokes, but cannot recall them easily when individually asked to.

Common Interviewing Mistakes

As this is labour and time intensive (and therefore costly), you don't want to diddle about. These are the four most common mistakes:

1. Not interviewing all of the right people.

   Different stakeholders have different points of view. Be careful to talk to everyone appropriate.

2. Asking direct questions too early.

   e.g., *Designing a transportation system:*

   → *How many horsepower do you need? (direct)*
   → *How many people? How far? How fast? (indirect)*

   e.g., *Camera design for novice photographer:*

   → *How important is control over shutter speed and aperture? (direct)*
   → *Will you be taking action shots, still shots, or both? (indirect)*

3. Interviewing one-at-a-time instead of in small groups.

   More people might help get juices flowing as in brainstorming. Also reduces spotlight on individuals, so you may get more interesting answers.

4. Assuming that stated needs are exactly correct.

   Often users don't know exactly what they want and order "what he's eating". Need to narrow what is asked for down to what is needed.

How to get Started Asking Questions

Detailed questions will be system specific. However, Weinberg suggests starting out with *context-free questions* to narrow the scope a bit.

→ Identify customers, goals, and benefits:

- Who is (really) behind the request for the system?
- Who will use the system? Willingly?
- What is the potential economic benefit from a successful solution?
- Is a (pre-existing) solution available from another source?

→ When do you need it by?

- Can you prioritize your needs?
- What are your time/cost/manpower constraints?
- Expected milestones? (with checklists)

→ Try to characterize the problem and its solution

- What would be a “good” solution to the problem?
- What problems is the system trying to address?
- In what environment will the system be used?
- Any special performance issues?
  Other special constraints?
- What is (un)likely to change? Future evolution? What needs to be flexible (vs. quick & dirty) ?

→ Calibration and tracking questions

- Are you the right person to answer these questions?
- Are your answers “official”?
  If not, whose are?
- Are these questions relevant to the problem as you see it?
- Are there too many questions?
  Is this the correct level of detail?
- Is there anyone else I should talk to?
- Is there anything else I should be asking you? Have you told me everything you know about the problem?

→ Unaskable questions (ask indirectly)

- Are you opposed to the system?
- Are you trying to obstruct/delay the system?
- Are you trying to create a more important role for yourself?
- Do you feel threatened by the proposed system?
- Are you trying to protect your job?
  Is your job threatened by the new system?
  Is anyone else's?

Apprenticeship

Apprenticing is a wonderful way to observe the real work. Apprenticing is based on the idea of masters and apprentices. In this case, the RA is the apprentice and the user is the master craftsman. The apprentice sits with the master craftsman to learn the job by observation, asking questions, doing some of the job under the master's supervision.

Almost anybody is good at explaining what he or she is doing while doing it. If the user is doing this job in the normal workplace, he or she can provide a running commentary and provide details that may otherwise be lost. It is probably *only* while working that the user can

- describe the task precisely,
- explain why the task is done this way, and
- list the exceptions that can occur.

Popping back up to the tasks of a requirements analyst...

- Understand the problem from each stakeholder's point of view.
- **Extract the essense of the stakeholders' requirements**
- Invent better ways to do the user's work.
- Negotiate a consistent set of requirements.
- Record the results in an SRS.

Extract Essense of Problem

Extracting the essence of the stakeholders' requirements requires

- interpreting the stakeholders' descriptions of requirements and
- building models

Interpreting stakeholders' descriptions of requirements

The stakeholders' descriptions of *some* of their requirements must be treated as factual. After all, they are the experts on their own needs. The user is the expert on what work he or she needs to do. However, some descriptions are too solution oriented or are based on current technology. The requirements analyst needs to see beyond these details to the essential problem that needs to be solved.

As an example this sort of interpretation, Jo Atlee received an e-mail message from a student from a previous term's CS445 class. He is on co-op, working as a requirements analyst. He had just come from a meeting with his project's client, in which the client listed a number of required "features" of the product. Here is a sample of those features. I don't know what the system is that they are building, but even not knowing this, you can tell that some of these are not requirements.

1. distributed service; distribution hidden from user
2. fault detection
3. objects (records) are uniquely identified
4. security
5. relationships between entities are XXX
6. mandatory adjacent/component system
7. system load shall be low

Which of these are requirements?

Which of these allude to requirements?

Which are design constraints?

Apply the same questions to the attempted requirement statements on the next page.

1. The client daemon must be invisible to the user.
2. The system should provide automatic verification of corrupted links or outdated data.
3. An internal naming convention should insure that records are unique.
4. Communication between the database and server should be encrypted.
5. Relationships may exist between title groups [a type of record in the database].
6. Files should be organizable into groups of file dependencies.
7. The system must interface with an Oracle database.
8. The system must handle 50,000 users concurrently.

Building models

Building models helps you to understand the problem, because you cannot build a model without understanding the subject of the model.

An obvious benefit of building models of the problem is finding out what questions to ask. Holes in the model reveal unknown or ambigous behaviour that need to be resolved by asking the user. As the model develops, it becomes more and more obvious what you don’t know, and sometimes also what the users don’t know. Model building, thus, helps the requirements analyst to focus his or her efforts.

When the model is finished, you have understood the problem, and you have a description of the problem documented in a notation that can be read by both you and the user.

Popping back up to the tasks of a requirements analyst...

- Understand the problem from each stakeholder's point of view.
- Extract the essense of the stakeholders' requirements.
- **Invent better ways to do the user's work**
- Negotiate a consistent set of requirements.
- Record the results in an SRS.

Invent a Better Way

This task is often overlooked during requirements elicitation. We recognize the importance of determining what the client wants and documenting that. If we stop there, then we're likely to build a system that conforms to only the client's limited notion of what is possible. However, where does the client get his or her ideas? He or she might want to automate the processes that are currently done manually. Alternatively, he or she might want a system that is similar to another existing system. However, is automating current processes or duplicating an existing system going to really help the client?

To be really successful, you need to give the client, not what he or she wants, but what he or she never dreamt of having; and when he or she gets it, he or she recognizes it as something he or she wanted all the time (IKIWISI).

- Ask why documented requirements are desired.
- Consider giving the user more creative control over his or her transactions.
- Brainstorm to invent undreamed of requirements.

Ask why documented requirements are desired

It may be that the client requested certain requirements to satisfy some more fundamental goal, and we would be better off concentrating on how to satisfy the more fundamental goal than the stated requirement. Consider a customer that uses an ATM to withdraw cash. Well, *why* does he or she want cash?

Is it to buy something? If so, then why not extend the ATM card to act as a debit card in retail outlets so that he or she doesn't have to go to the ATM in the first place.

Is it to pay her electricity bill on her way to work? If so, the why not offer the opportunity to pay bills at the ATM.

Does he or she just want to see his or her account balance? If so, then why not give her the facility to do this over the phone or on the Internet?

Consider giving the user more creative control over his or her transactions

People would rather do some of the work themselves, if they think they would do a better or faster job. CAD software allows users to design their own furniture, houses. Investors trade stock over the Internet without the advice or intervention from a broker or trader. Shoppers are using self-scanners to scan and pay for groceries, rather than queuing for the checkout.

Brainstorming -1

Brainstorming is already part of our culture, but beware of bad brainstorming.

A bad brainstorming session is a brainblizzard because it freezes your brain, leaves you under mounds of snow, and leaves you cold

We will give rules for brainstorming that help avoid the brainblizzard.

Brainstorming

- When you have no idea, or too many ideas, sit down and thrash it out ... but with some ground rules.
- Most useful early on, when terrain is uncertain, or when you have little experience, or when novelty is important.
- Who participates?
  - → developers, domain experts, end-users, clients, ... just about any stakeholder can take part.
  - → Often, software development companies will have special-purpose "ideas-guys"[a] who lead or attend these meetings, but may not participate beyond this stage.

[a] Could be female or male.

Brainstorming

- Want to hear ideas from everyone, especially unconventional ideas.

  → *keep the tone informal and non-judgemental*

  → *keep the number of participants "reasonable", if too many, consider a "playoff"-type filtering. Invite back most creative to multiple sessions.*

  or it's too hard to be heard (only the loud will prevail).

- Creativity to be encouraged, which means:

  → Choose good, provocative project name.

  → Choose good, provocative problem statement.

  → Get a room w/o distractions, but with good acoustics, whiteboards, coloured pens, provide coffee/donuts/pizza/beer

  → Provide appropriate props/mock-ups (*e.g.*, ComfyCrate)

Brainstorming

First, must designate two (different!) people for special roles:

1. Scribe — Role is to write down all ideas. May also contribute. May ask clarifying questions during first phase, but not critical questions.
2. Moderator/leader — Two schools of thought on this:
   (a) Traffic cop — enforces “rules of order”, but doesn’t throw his/her weight around otherwise.
   (b) Agent provocateur – Assumes more of a leadership role, comes prepared with wild ideas and throws them out as discussion wanes. May also explicitly look for variations and combinations of other suggestions. Not a “philosopher-king”. Also acts as traffic cop.

Brainstorming

Part I — The Storm

- Goal is to generate as many ideas as possible.
- Quantity, not quality, is goal at this stage.
- Look to combine or vary ideas already suggested.
- No criticism or debate is permitted. Don't want to inhibit participants.
- Participants understand nothing they say will be held against them later on.
- Scribe write down all ideas where all can see

  e.g., whiteboard, paper taped to wall

- Wild is good. Feel free to be gloriously wrong.

- → *Participants should NOT self-censor or spend too much time wondering if an idea is practical. Just shout it out.*
- → *Original list does not get circulated outside of the meeting.*

Brainstorming

Part II — The Calm

- Go over the list. Explain ideas more carefully.
- Categorize into "maybe" and "no" by pre-agreed consensus method.

  → *informal consensus,* $50\% + 1$ *vs. "clear majority", Dutch auction, who has vetoes.*

- Be careful about time.

  → *Meetings (esp. if creative or technical in nature) tend to lose focus after 90 to 120 minutes. Take breaks or reconvene later.*

- Review, consolidate, combine, clarify, expand.
- Rank the list by priority somehow; choose a winner.

- Look out for:
  - → Haggling over details
  - → Hurt feelings
  - → Time limits

Pruning -2

There are several choices of how:

- **voting with threshold**
- **voting with campaign speeches**
- **blending ideas**

Voting with threshold

Each person is allowed to vote up to *n* times.

Keep those ideas with more than *m* votes.

Have multiple rounds thereof with smaller *n* and *m*.

Voting with campaign speeches

Each person is allowed to vote up to $j < n$ times.

Keep those ideas with at least one vote.

Have someone who did not vote for an idea defend it for the next round.

Have multiple rounds thereof with smaller *j*.

Blending ideas

Apply acceptance criteria (which tend to be ignored in first step) to ideas.

Rank accepted ideas.

Select top *k* for voting treatment.

Other Brainstorming Ideas

Brainstorming can be carried out over e-mail.

But a leader is needed to prevent flaming and race conditions.

One Final Point!

With lots of good, outrageous, outlandish ideas, the brainstorm is loads of fun!!

Fun motivates people to do well!!!

Popping back up to the tasks of a requirements analyst...

- Understand the problem from each stakeholder's point of view.
- Extract the essense of the stakeholders' requirements.
- Invent better ways to do the user's work.
- **Negotiate a consistent set of requirements**
- Record the results in an SRS.

Negotiate Consistent Set of Requirements

This negotiation is not so easy.

First, you have to detect when stakeholders' requirements are inconsistent.

Then, you have to convince all the stakeholders to understand the essential requirements from the point of view of each other.

Finally, you have to come to an agreement about a consistent set of requirements that best satisfies everyone.

Key Aspects of Requirements Negotiation

- intended requirement — “territory”
  — the real requirements, which lie in the heads of the stakeholders
- documented requirement — “map of territory”
  — model of intentions.
  A good analogy is maps. If the intended requirements are the territory, the documented requirements are a map of the territory. The documented requirements are never as complete as the intended requirements; they are only a model. However, we need to try to document them, to allow all stakeholders to discuss them and to negotiate.

- understood requirement — “interpretation of map”
  — interpretation of documented requirements.
  Ideally, everyone has the same understanding of the requirements. One of the motivations for using modelling languages such as UML and SDL is that they are less ambiguous than natural language and thus models written in these languages are open to fewer interpretations than natural language specifications.

- significance — "fish-eye maps"
  — importance of documented requirement to stakeholder.
  You've probably seen these humorous "fish-eye view maps" of various cities, e.g., the *New Yorker*'s map of New York City, that give exaggerated importance to the city itself and model the rest of the country or world based on its significance to the city. We can imagine that each stakeholder has a fish-eye views of his or her requirements with respect to the project's requirements. Therefore, we need to determine, for each stakeholder, whether a documented requirement is one of the stakeholder's requirements; is the antithesis of the stakeholder's requirements; or is a complementary requirement, in which case, the stakeholder doesn't care whether or not it is satisfied.

- stakeholder's counter-proposals
  — "map-boundary negotiations"
  How a stakeholder responds to a documented requirement, i.e., being open-minded, close-minded, cooperative, etc., can indicate how open the stakeholder is to negotiation and to coming to an agreement that optimizes all of the stakeholders' requirements.

Popping back up to the tasks of a requirements analyst...

- Understand the problem from each stakeholder's point of view.
- Extract the essense of the stakeholders' requirements.
- Invent better ways to do the user's work.
- Negotiate a consistent set of requirements.
- **Record the results in an SRS**

Record the results in an SRS

The last step, recording the results in an SRS, is the subject of most of this course.

What Can Go Wrong in Elicitation and the SRS?

- Unknown requirements

  The hardest part of writing specifications involves anticipating all of the possible circumstances that might occur. Users expect software systems to respond correctly to whatever input is presented. Problems occur because there are situations that nobody considers during development, and thus the software does not handle the situation when it occurs.

  One of the goals of modelling is to reveal circumstances that have not been considered and reveal areas of the problem that need to be explored more with the customer.

- Known but undiscussed requirements (assumptions)

  Assume == “ass” of “u” and “me”
- Discussed but undocumented requirements
- Wrongly documented requirements

  These requirements are sometimes sabotage from users who don’t want the system to succeed, either because they don’t want their routines to change, or because their jobs are threatened by the new system.

- Inconsistent requirements

  Inconsistency may arise in large systems that are organized into parts. Decisions made for one part of the system may not be compatible with those made for another part. Alternatively, requirements from one stakeholder may be incompatible with requirements from another stakeholder.

- Misinterpreted requirements

  With misinterpreted requirements, the requirements document may be completely correct, but the implementor may mistakenly intrepret the requirements differently.

Other Techniques

- PIECES
- Social and Organizational Factors
- Ethnographic Analysis
- Joint Application Design
- Names and Norms
- Gause & Weinberg Ideas

The PIECES Approach

- A more structured approach than simple brainstorming; think of as a vanilla RE process.
- Works best with existing system or well-understood domain, but perhaps inexperienced requirements engineers.

- Main idea:

  $\rightarrow$ Examine system from six specified points of view. Provides a lowest common denominator starting point when you are not sure how to get started.

- Oriented towards office information systems (esp. enhancing/modifying existing systems), but concepts are broadly applicable.

- PIECES == P erformance, i nformation and data, e conomy, c ontrol, e fficiency, and s ervices.

- There is overlap between areas, but that's OK; you're examining different points of view.

The PIECES Approach

Performance

Usually measured as either

- → **throughput** — *number of tasks completed per unit time*
- → **response time** — *avg time between request and completion of task.*

For new system, identify main (and later subordinate) tasks, and query for acceptable avg case and worst case performance. For existing systems, users will likely know where bottlenecks are.

The PIECES Approach

Information and data

Query stakeholders about:

- → **relevance** — *Is the information what you want to see? too much? too little?*
- → **form** — *Is the presentation helpful? Comprehensible?*
- → **timeliness** — *Are you getting the right information at the right time?*

  e.g., *notification if inventory drops below threshold*
- → **accessibility** — *How* easy *is it to get what you want to know? Too many levels of indirection? Unintuitive structuring?*

The PIECES Approach

Economy

→ Many systems have varying usage levels. To handle heavy periods, must have some way of providing minimal acceptable performance. Usually, this means spending money on pieces (*e.g.*, memory, processors, telephone switches) that sit idle during off peak times.

e.g., *Telephone switches have extra circuits to handle peak calling times reasonably, but not enough to guarantee service.*

→ "Economy" means discussing the various trade-offs of costs vs. minimal acceptable performance.

→ Jargonese: Want to reduce "excess capacity" to the point that system provides "desired service level".

For example, telephone switches have just the number of circuits needed to be able handle the load most of the time. In North America, the goal is to be able to provide service to 15% of the population at any one time — at least this used to be the goal. However, the resources needed for this level of service is certainly not enough to serve everyone if everyone decides to make a call at the same time, e.g., when California has an earthquake.

The MFCF undergrad environment supposed to provide enough resources and terminals to handle the student load most of the time. These resources are not enough when multiple assignments are due, e.g., during the last week of a term.

The PIECES Approach

Control

- → Who gets to do what when.
- → Explicitly address which functions may be controlled by which human or hardware or software entities.
- → Particular concerns:
  - exceptional circumstances, non-standard system behaviour
  - system security, auditing

- → Too little control for users ⇒ they have lost the skill to get system to do what they want
- → Too little control for users ⇒ they get lulled into complacency, they do not know the current state of the system, and they, therefore, are not able to handle emergencies
- → Too much control ⇒ too much hand-holding of system required, takes time away from more important tasks; added complexity to using system; easier to perform tasks incorrectly.

Control — degrees of automation, auditing, robustness, security

Control is a general category that covers a lot of topics.

- degree of automation — how much of the work should be automated by the system and how much should be done by humans.

  For example, telephone switching used to be done by human operators, but nowadays, more and more operating tasks are being automated.
- degree of auditing — degree to which the system monitors and audits its own work; it is the ability to see, monitor, and reconstruct system behaviour during or after an event.

  For example, all financial systems audit the users' transactions.

- degree of robustness — degree to which the system is responsible for detecting and correcting faults.

  For example, telephone switches have a lot of code that monitors the states of the software and hardware, and when a weird state is detected during a call, the software will reset that call.
- degree of security — degree to which the system controls access to its information.

The PIECES Approach

Efficiency

→ Measurement of waste.

→ Different from economy:

- Economy — intentional "waste" deemed acceptable.
- Efficiency — unintentional "waste" that no one has noticed or bothered to improve. Usually involves reallocation of hardware or rewriting of software.

e.g., Underused hard drive can be used to cache data from remote sites.

e.g., Naive implementation of system bottleneck is profiled and tuned.

The PIECES Approach

Services

→ Consider the set of services currently provided by the system in the context of the larger problem of how the system in used.

e.g., Look at "secondary users" (clients of users). Look at the bigger picture.

→ Is there a better way to building a system to solve the big picture?

e.g., *Inventory control system where entries are done by company clerk based on requests from secondary users vs. building a Java GUI and letting secondary users enter requests directly. If take latter approach, what other kinds of services would you need too?*

→ Have to interview a wide variety of stakeholders for this to work:

e.g., users, "secondary users", managers

Exercise:

Identify and discuss how the different categories within PIECES apply in the following two situations. If additional information is required to more accurately pinpoint the problem, discuss this also.

1. Employees are gaining unauthorized access to payroll information.
    - I: information accessibility
    - I & C: control accessibility
    - C: degree of automation?
      perhaps access to information shouldn't be automated?
    - C: auditing?
      perhaps more auditing will catch the problem

2. Poorly constructed products are getting past quality control and being shipped to customer.
    - I: is relevant information accessible?
    - I: is information available in time to make decision?
    - C: control over construction increased?
    - C: more auditing of quality?
    - C: can quality problems be detected automatically?
    - E & E: less emphasis on economy and efficiency, if they affect the effectiveness of quality control
    - etc.

Social and Organizational Factors

"No system is an island unto itself".

- All software systems exist and are used within a particular *context*. — *technical AND social*
- Social and organizational factors are not only important, often they dominate the system requirements!
- Determining what these are can be difficult and time-consuming
  - *developers are (usually) outsiders*
  - *people don't always tell the truth*
  - *awareness of one's own "culture" can be hard*
- *No universal way to deal with this problem, just common sense.*

Social and Organizational Factors

These factors tend to cut across all aspects of the system:

e.g., a system that allows senior managers to access information directly without going through middle managers

- *interface must be simple enough for senior managers to be able to use*
- *middle managers may feel threatened or encroached upon, be resistant to new system*
- *lower-level users may concentrate on activities that impress senior managers, which is not necessarily what they* ought *to be doing*
- *users may not like "random spot checks"; may devise ways of hiding what they're doing*

Ethnographic Analysis

Basically, an attempt to discover the social/human factors is a system.

- Rationale:

  $\rightarrow$ studies have shown that work is often richer and more complex than suggested by simple system models derived by interviews alone.

- a specially-trained "social scientist" spends a lot of time observing and analyzing how people actually work

- discovery is by observation and analysis; workers are *not* asked to explain what they do.
- often, this is a very instructive way to discover social- and human-oriented factors of systems.

  e.g., What does a nuclear technician do all day? What does his/her workspace look like?
- *it's less useful in discovering political factors as workers are aware of presence of an outsider.*

"Focused Ethnography"[a]

- Sommerville *et al.* were involved in a project to discover the requirements for an air traffic control system. They spent time watching air traffic controllers in action with an existing system.
- They made some surprising observations:
  - Controllers often put aircraft onto potentially conflicting flight paths with the intention to correct them later.
  - Existing system raises an audible warning when any conflict possible.
  - Controllers turned the buzzer off, because they were annoyed by the constant "spurious" warnings.

  Wrong moral: Controllers don't like audible warnings since they turn them off.

  More accurate observation: Controllers don't like to be treated like idiots.

[a] Sommerville Chapter 5.4

"Focused Ethnography"

- Sommerville's approach was to combine ethnography with prototyping. Prototyping allowed focusing on questions of "why".

  $\rightarrow$ often, the "obvious" answer was not the correct one

- One problem: ethnography concentrates on modelling existing practice

  - sometimes, practices are no longer necessary, but old habits die hard
  - people may not remember why something is done a particular way

    $\rightarrow$ *might be OK (and a good idea) to remove*

    $\rightarrow$ *users might like (or depend on) legacy features*

    $\rightarrow$ *might cause disaster (in unusual circumstances) to remove*

Ethnography: Summary

- Social/human/political factors often extremely important.
- More study (and real-world examples) needed!
- It ain't science, but we still have to deal with these problems somehow.
- This is yet another example of how the human/social side of computer use has received inadequate attention.

[Cue violin music]

JAD — Joint Application Design

- Developed at IBM in the 1970s;
  lots of success stories.
- Think of as "structured brainstorming", IBM-style. *full of structure, defined roles, forms to be filled out, TLAs*
- Two major "steps", three phases each, and six (human) roles to be played!
- Four main tenets of JAD:
  1. Effective use of group dynamics. *facilitated and directed group sessions to get common understanding and universal buy-in*
  2. Use of visual aids. *to enhance understanding,* e.g., *props, prepared diagrams*
  3. Defined process. i.e., *not a random hodgepodge*
  4. Standardized forms for documenting results. *LCD approach*

JAD — Overview

- Two main steps:
  1. **JAD/Plan** — used for elicitation (brainstorming).
  2. **JAD/Design** — used to design software
     (we won't discuss it).
- Three phases of each step:
  1. **Customization** — *Preparing for meeting*
  2. **Session** — *the meeting itself*
  3. **Wrap-up** — *reporting and summarizing what happened*

JAD — Roles

Six roles:

1. **Session leader** — organizer; facilitator; JAD expert; good with people skills; enthusiastic; sets tone of meeting.
2. **Analyst** — scribe++; produces official JAD documents; experienced developer who understands the "big picture"; good philosopher/writer/organizer.
3. **Executive sponsor** — manager who has ultimate responsibility for product being built; provides "strategic insights" into company's high-level goals/practices; later on, makes "executive decisions" as required.

JAD — Roles

4. **User representatives** — selection of knowledgeable end-users and managers; come well-prepared with suggestions and ideas of needs; will brainstorm for new or refined ideas; eventually review completed JAD documents.

5. **Information system representative** — technical expert on ISs; helps users think big, know what's easy/hard/cheap/expensive; mostly there to provide information rather than make decisions.

6. **Specialist** — technical expert on particular narrow topic, *e.g.*, security, application domain (travel agencies), law, UI issues.

JAD/Plan — Stages

1. **Customization**

   → Good preparation is key; JAD session will *not* be just an informal free-flow of ideas.

   → Executive sponsor picks participants. Likely conducts brief orientation of JAD structure for each.

   → Session leader and executive sponsor familiarize themselves with problem/clients/subject area.

   Identify likely points of contention, and clarify what is to be within/outside the scope of the JAD session.

   → Prepare materials for session.
   e.g., *flip charts, Powerpoint slides, markers/whiteboards, order pizza,* etc.

JAD/Plan — Stages

2. **Session**

   → Session leader welcomes participants, presents task to be discussed, established ground rules and context for discussion, *What is on/off topic*, makes initial suggestions.

   → The ball is now rolling. Brainstorm away.

   → At end of session, evaluate suggestions and agree upon recommendations/requirements to be passed to JAD/Design team.

3. **Wrap-up**

   → Analysts write up what has been agreed upon using standardized JAD forms. Annotate recommendations with "rationale".

   → All participants review the documents. Changes are made as needed. Executive sponsor signs off. QED.

Other Elicitation Concepts

Here are some twists of G&W concepts that G&W did not think of, but I thought of when reading G&W.

They concern:

- **norms**
- **mockups & prototypes**
- **existence assumption**
- **right-brain methods**
- **naming**

Norms -1

The general form of the use of a norm to state requirements:

Here is an *X*; build a better *X*

The norm can protect you from colossal blunders by starting with something that is clearly feasible.

But, it can keep you from seeing a new way to solve the problem that *X*, itself, is solving by keeping you immersed in enhancing *X*.

Norms -2

Here's an example of such a norm:

"Build a better pencil-and-paper set" could prevent you from thinking of the computer as an authoring tool.

Norms -3

Another example:

Avocado is a fruit.

Problem: peel avocado.

Thinking of the norm of fruit causes you to try to peel avocado with knife.

Wotta mess!

Norms -4

A better solution is to scoop the avocado meat out of the peel-shell with a spoon just the right size.

Mockup & Prototype -1

Another danger of using a norm is that different people's perception of the norm may be different.

Making a mockup or prototype makes sure that all people use the same norm.

Mockup & Prototype -2

Mockups & prototypes are also used when no norm is possible, when solving an entirely different problem.

Mockups & prototypes are also used to elicit a credible validation response.

You believe a “Yes, this is what I want.” in response to a mockup or prototype more than you do to a DoD-standard written requirements document.

Existence assumption -1

Underlying the whole search for a solution is the assumption that a solution exists.

Generally this assumption is tacitly accepted.

Usually this is just fine.

Existence assumption -2

People have a good sense of when a problem is solvable and when it is not.

But, sometimes it is necessary to examine the existence assumption and verify that it is reasonable.

If the assumption is not reasonable, then maybe the problem has to be changed.

Existence assumption -3

Example:

Situation: I bought rim-mounted woks and used them for years over a gas stove.

Then I moved to an electric stove.

The woks never got hot enough to make the food right; the food came out greasy.

The first attempt to state the problem: Make an electric burner hot enough to heat a rim-mounted wok.

Existence assumption -4

A solution to this problem does not exist, because so hot a burner will damage itself and the rim.

Finally, I realized that the solution was to change the shape of the wok so that the cooking surface is completely on the burner.

So I went out and bought a flat-bottomed wok!

Right-Brain Methods -1

G&W discuss a number of right-brain methods to help overcome communicational ambiguities.

The left brain is the more textual, logical half.

The right brain is the more pictorial, free associating half.

Right-Brain Methods -2

When you do not understand someone, ask him or her to draw a diagram showing his or her meaning.

Or, draw your conception of what he or she is saying and ask if this is what he or she means

This is sort of like what was done using the holodeck on *Star Trek, the Next Generation* episode “Schisms”.

Naming -1

What's in a name? A rose by any other name would smell as sweet!

Ah, but if the rose were not visible or smellable and someone asked you if you wanted a vered would you answer "yes" as quickly as you might if you were asked if you wanted a rose?

"Vered" is Hebrew for "rose".

Naming -2

What if you were asked if you wanted a qwiddlyhop?

G&W show how a bad name can distract a project and how a good name can be an inspiration to all that work with it.

Naming -3

G&W discuss how an inaccurate name can mislead those who perceive it and cause clashes when confronting the real thing.

So, it is worth taking time out to brainstorm for a good name.

Naming -4

Be careful with backcronyms

A backcronym is clever name that is made after the fact, an acronym for a contrived sequence of words.

Those words may not accurately describe the project, and may eventually mislead newcomers, clients, and users.

Naming -5

Getting the right name is like getting the right norm.

“Post-its” suggests better uses than does “half-sticky adhesive”.

“Scoop out meat of avocado” suggests a better solution than “peel off skin of avocado”.

Words

A picture is worth a thousand words.

One word is worth a thousand pictures.

That word is an abstraction of the thousand pictures, and of the million words that are worth the pictures.

Purpose of Elicitation

Basically, the purpose of elicitation is to clarify

- **functions**
- **attributes**
- **constraints**
- **preferences**
- **expectations**

for the system to be built.

Clarifying -1

We now look at what information needs to be elicited.

Basically, it gets down to determining the users' and the client's expectations.

Clarifying -2

To get a precise idea of these expectations, G&W recommend elicitation of

functions,
attributes,
constraints,
preferences,
expectations.

For each, G&W recommend a brainstorm involving all parties with a stake in the requirements.

Functions

Functions are what the product does.

A function should be a phrase beginning with a verb whose subject is the product to be built.

KWIC **accepts ordered list of lines: OK**
refer **accepts imprecise citations: OK**
refer **is fast: NOT OK (attribute)**

Attributes

Attributes are product characteristics desired by the client.

They are adjectives or adverbs.

Two products with nearly identical function are distinguished by their attributes.

Rolls Royce and Volkswagen have mostly the same functions but differ greatly in attributes.

Constraints

A constraint is a mandatory condition placed on an attribute.

In order for the final design to be acceptable, every constraint must be satisfied.

Preferences -1

A preference is a desirable, but optional, condition placed on any attribute.

Any final design that satisfies all constraints is acceptable, but some acceptable designs are preferable to others.

Preferences -2

Preferences allow designers to compare acceptable solutions and choose the better ones.

“After we have met all constraints we’ll take all preferences we can get, so long as we don’t have to pay for them!”

Preferences -3

Preferences come from the questions to which the client answers, “I dunno!”

Since he or she doesn’t have a real answer, it cannot be a requirement, but it *is* a variable.

Some choice must be made somewhere along the line.

Expectations -4

The client, just as anyone else, has expectations.

The difference ’twixt disappointment and delight over a product is how well expectations are matched upon delivery of the product.

Expectations -5

Sometimes, a client's expectations are too high.

Perhaps, the client has developed unreasonable expectations for the product from having seen other products or movies.

Expectations -6

Perhaps, the client has not read the fine print.

Actually, there should be *no* fine print; if there is, you, the requirements engineer, have not done your job.

It is your job to limit the client's expectations to something reasonable.

Expectations -7

Reasons to limit expectations:

If all prior steps were done perfectly, expectation limitation would be redundant, but

- **people are not perfect,**
- **people are not logical,**
- **people perceive things differently,**
- **designers are people too.**