Announcements

- Form your project group soon! Those who don’t have a group could stay after at either lecture or tutorial to meet others who don’t have a group.
- There is a tutorial this week.
- Graduate students should contact Nancy and Davor regarding the topic for their research presentation soon. Please send email to cs445@student.cs.uwaterloo.ca with a brief description of a topic of interest to you. If you don’t have an idea for a topic, please send us email to set up a meeting to discuss possible topics.
- Details on the research project and possible topics are available at the link “Graduate Students” on the course web page.
- Topics should be chosen and approved by the instructors by Fri Jan 13th.

Review

A requirements specification is a description of the proposed behaviour of a computer-based system (CBS).
- What is assumed of the environment in which the CBS operates?
- Given these assumptions, what shall the CBS do?
- What are the boundaries of this CBS?

SRS = Software Requirements Specification

The requirements specification can often form a contract between the customer and supplier.
Today’s Agenda

- Requirements Elicitation (and Analysis)
  - Why is requirements elicitation hard?
  - Who are the stakeholders?
  - Tasks of the requirements analyst
  - Requirements elicitation techniques

Required Readings:
- Lauesen, Ch. 8 (course pack)
- D. Berry, *The Importance of Ignorance in Requirements Engineering* (course pack)

Quote

From Goguen & Linde:

*Some Computing Scientists might think that requirements elicitation is where silence stops and chaos begins.*

Requirements Elicitation

Definition

“to elicit”

means

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

The purpose of elicitation is to get information about:
- the domain in which the system operates
- information about present work and present problems
- the requirements of the system (from which the system is developed)
Purpose of Elicitation

More specifically, the purpose of elicitation is to clarify

- functions
- attributes
- constraints
- assumptions about the environment
- preferences
- expectations

for the system to be built.

Preferences

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

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

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

Expectations

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.

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.

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

Reasons to limit expectations:
Under ideal circumstances, expectation limitation would be redundant, but
- people are not perfect,
- people are not logical,
- people perceive things differently,
- designers are people too.

Elicitation Techniques

Why is Req Elicitation Hard?

- Requirements are hard to articulate
- Requires problem solving
- Human element
- Lack of user motivation; resistance to change
- Lack of imagination
- Demands may evolve
- May be fixed on one solution
- Hard to judge the quality of the result, or to judge when done
- Involves compromises and setting priorities (negotiation)
**Elicitation Techniques**

[Lauesen, 8.2]

- Stakeholder analysis
- Interviews
- Observation
- Task Demo
- Document studies
- Questionnaires
- Brainstorm
- Focus groups
- Domain workshop
- Design workshop
- Mockups & Prototyping
- Pilot experiments
- Similar companies
- Ask suppliers
- Negotiation
- Risk analysis
- Cost/benefit
- Norms

**Who are the Stakeholders?**

Stakeholder = person “needed to ensure the success of a project” (Lauesen p. 35)

First you have to figure who are the stakeholders for a project. Here are some possibilities:

- Client
- Customer
- Users
- Domain Experts
- Software Engineer
- Inspectors
- Market Researchers
- Lawyers
- Experts on Adjacent Systems
- Value-adders

**Stakeholders: Client**

Client = person paying for the software to be developed

- Ultimate stakeholder – the client has the last say in what the product does.
- For 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.
- For software for the mass market – the client may be the marketing department.
Stakeholders: Customer

Customer = person who buys software after it is developed; possibly a manager
- May be the same as the user; other times, 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?
- Must be an active participant in the project (or have an active representative if there are many customers).

Stakeholders: Users

Users (of both the current and future systems)
- Experts on the existing system – tell you which features to keep and which need improvements, or
- Experts on competitors’ products – give suggestions about how to build a superior product.
- May have special needs or requirements to be satisfied, e.g., regarding usability and training.
- May want to consult special-interest groups: users with disabilities, users who have computer phobias, expert users, novice users, etc.

Stakeholders: Users (con’d)

Selection of users:
- balance in seniority
- must have authority
- must be accountable
- must be knowledgeable
- must be motivated
May have to train users by reviewing existing system.
[Scharer, 1981]

Stakeholders: Domain Experts

Domain Experts = experts who know the work
- Familiar with the problem that the software must solve.
  Examples:
  - financial experts for financial packages,
  - aeronautical engineers for aircraft navigation systems,
  - meteorologists for software that models the weather, etc.
- Know about kind of environment the product will be exposed to
Stakeholder: Software Engineer

Software Engineer = technology expert

- Ensure that the project is technically and economically feasible
- Accurately estimate the cost and development time of the product
- Educate the customer about innovative hardware or software technologies, and recommend new functionality that takes advantage of these technologies

Stakeholders

Inspectors = experts on government and safety regulations

- Familiar with government and safety regulations relevant to project
- Examples: safety inspectors, auditors, technical inspectors, government inspectors

Market Researchers

- May assume the role of client, if the software is being developed for the mass market and there is no identifiable customer.
- Experts who have conducted surveys to determine future trends and potential customers’ needs.

Stakeholders

Lawyers

- Familiar with legal requirements
- Standards that are relevant to the product.

Experts on Adjacent Systems

- Know about the interface for the adjacent system, and any special demands for interfacing with the product
- 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.

Value-adders

- People who build on your product

Tasks of Requirements Analyst

1. Examine project viability
2. Understand problem from each stakeholder’s point of view (may involve decomposition)
3. Extract the essence of the stakeholders’ requirements
4. Invent better ways to do the user’s work
5. Negotiate a consistent set of requirements with agreement from all stakeholders; set relative priorities
6. Record results in an SRS
1. Examine Project Viability

- Does it make good business sense to begin doing the project?

Determining viability requires examining the product's:

1. Purpose
2. Business advantage
3. Scope
4. Feasibility
   - (a) Required resources
   - (b) Costs vs. benefits
5. Constraints
6. Risks

1. Examine Project Viability

1. Purpose: What does the product do?
   - highest-level customer requirement (business goal)
   - all other requirements must contribute in some way to the purpose

2. 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?

1. Examine Project Viability

3. 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.

1. Examine Project Viability

4. Feasibility
   - technical feasibility, and
   - economic feasibility

Does the organization have the resources and expertise to build and operate the product?

One of the reasons for stating measurable requirements early on is to be able to answer questions about feasibility.
1. Examine Project Viability

4(a) 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, halt the project!

4(b) Costs vs. Benefits

- How much will it cost to develop and operate the product? (from part a)
- How much will the product help our work?
- Is the advantage greater than the cost of constructing the product?

If there is an advantage, you must be able to measure it in order to demonstrate that the product achieves the advantage.

Vague statements of advantage are open to misinterpretation between what the customer thinks he or she is going to get and what the software developer provides.

If the cost is going to be greater than the benefit, halt the project now!

5. Constraints

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

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

6. Risks

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

Such risks include:
  - absence of clear purpose
  - little or no agreement on requirements
  - unmeasurable requirements
  - rapidly changing requirements, etc.
1. Examine Project Viability

Summary

... the idea is to do just enough investigation to form a rational, justifiable opinion of the overall purpose and feasibility of the potential new system, and decide if it is worthwhile to invest in deeper exploration.

[Larman, p. 35]

Norms

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 \(X\).

Mockups & Prototypes

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.

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

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.

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.
**Job of Requirements Analyst**

1. Examine project viability
2. **Understand problem from each stakeholder’s point of view** (may involve decomposition)
3. Extract the essence of the stakeholders’ requirements
4. Invent better ways to do the user’s work
5. Negotiate a consistent set of requirements with agreement from all stakeholders; set relative priorities
6. Record results in an SRS

**3. Extract Essence of Problem**

- interpreting the stakeholders’ descriptions of requirements, and
- building models

Holes in the model reveal unknown or ambiguous behaviour that need to be resolved by asking the user, which helps to focus your efforts.

The model becomes part of your SRS!

**Example**

Which of these are requirements?

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.

**Tasks of Requirements Analyst**

1. Examine project viability
2. Understand problem from each stakeholder’s point of view (may involve decomposition)
3. Extract the essence of the stakeholders’ requirements
4. **Invent better ways to do the user’s work**
5. Negotiate a consistent set of requirements with agreement from all stakeholders; set relative priorities
6. Record results in an SRS
4. Invent a Better Way

- Clients notion may be limited to past experience
- Ask why documented requirements are desired
- Consider whether the system should give the user more creative control over his or her transactions
- Brainstorm to invent undreamed of requirements.

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” (could be female or male) who lead or attend these meetings, but may not participate beyond this stage.

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
**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:
   - **Traffic cop** — enforces “rules of order”, but doesn’t throw his/her weight around otherwise.
   - **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.

**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.

**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.

**Pruning**

There are several choices of how to do pruning:

- **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.
**Brainstorming**

- Can be carried out over e-mail. But a leader is needed to prevent flaming and race conditions.
- Look out for:
  - Haggling over details
  - Hurt feelings
  - Time limits

**Tasks of Requirements Analyst**

1. Examine project viability
2. Understand problem from each stakeholder’s point of view (may involve decomposition)
3. Extract the essence of the stakeholders’ requirements
4. Invent better ways to do the user’s work
5. **Negotiate a consistent set of requirements with agreement from all stakeholders; set relative priorities**
6. Record results in an SRS

---

**5. Negotiate Consistent Set of Reqs**

- Find the inconsistencies!
- Convince all the stakeholders to understand the essential requirements from the point of view of each other.
- Come to an agreement about a consistent set of requirements that best satisfies everyone.
### Elicitation Techniques

[Lauesen, 8.2]

- Stakeholder analysis
- Interviews
- Observation
- Task Demo
- Document studies
- Questionnaires
- Brainstorm
- Focus groups
- Domain workshop
- Design workshop
- Mockups & Prototyping
- Pilot experiments
- Similar companies
- Ask suppliers
- Negotiation
- Risk analysis
- Cost/benefit
- Norms

### Analyze Current System

To understand the problem, analyze existing system if possible:

- Document studies (review existing documentation)
- Observe current system/apprenticeship
- Questionnaires
- Interviews

Find out:

- What is used, what isn’t, what’s missing.
- What works well, what doesn’t.
- How the system is used, how it was intended to be used, what new ways we want it to be used.

### Questionnaires

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

Questionnaires are useful also when the answers to questions need to be compared or corroborated.

- Closed questions (gather opinions)
- Open questions: ask problem-oriented questions (gather suggestions)

### Interviews

Detailed questions will be system specific. However, Gause & 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?
Interviews

- 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?

Interviews

- 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?
Ignorance is Good!

According to Berry, ignorance of the domain is good! Ignorance (not stupidity) makes it possible to expose unstated assumptions.

The designated ignoramus can ask questions whose answers are "obvious" to the experts, but often expose hidden knowledge that might otherwise not be modelled explicitly.

Berry suggests that one day, requirements engineers will advertise their areas of ignorance (rather than expertise) to get jobs.

Common Interviewing Mistakes

- Not interviewing all of the right people.
- Asking direct questions too early.
- Interviewing one-at-a-time instead of in small groups.
- Assuming that stated needs are exactly correct.
- If in a group, don’t let one person dominate.

Tip: To ask “why?”, ask “when do you do this?” (Lauesen, p. 340)

Naming

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

What if you were asked if you wanted a qwiddlyhop?

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

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

Be careful with backcronym

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.

Getting the right name is like getting the right norm.
Elicitation Techniques

[Lauesen, 8.2]
- Stakeholder analysis
- Interviews
- Observation
- Task Demo
- Document studies
- Questionnaires
- Brainstorm
- Focus groups
- Domain workshop
- Design workshop
- Mockups & Prototyping
- Pilot experiments
- Similar companies
- Ask suppliers
- Negotiation
- Risk analysis
- Cost/benefit
- Norms

Challenges

What can go wrong in elicitation?
- Unknown requirements
  - have you considered all the inputs?
- Undocumented assumptions
- Discussed but undocumented requirements
- Wrongly documented requirements
- Inconsistent requirements
- Ambiguous requirements

Related Techniques

- PIECES
- Social and Organizational Factors
- Ethnographic Analysis
- Joint Application Design

See notes at the end of this lecture for these topics.

Advice

The best way to avoid change requests and cost over-runs is to
- have a complete list of stakeholders,
- have a complete list of requirements from each stakeholder, and
- ensure that the lists are consistent with one another.
Skills

The skills needed for elicitation are:

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

Summary

- Requirements Elicitation (and Analysis)
  - Why is requirements elicitation hard?
  - Who are the stakeholders?
  - Tasks of the requirements analyst
  - Requirements Elicitation Techniques

Next Lecture: Use Cases

Required Readings:

- Arlow and Neustadt, Ch. 4, 5

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.
**PIECES**

- **Main idea:**
  - 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), EFFICIENCY, 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.

**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?

**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”.
Example

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 CSCF 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 (con'd)
- 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.

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

The PIECES Approach

Control (con'd)
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.
The PIECES Approach

Control (con'd)

- 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.

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**
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

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 in a system.

- Rationale:
  - 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
Ethnographic Analysis

- 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

- Sommerville et al. (Sommerville Chapter 5.4) 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 raise 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.

Focused Ethnography

- Sommerville’s approach was to combine ethnography with prototyping. Prototyping allowed focusing on questions of “why”.
  - 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
    - might be OK (and a good idea) to remove
    - users might like (or depend on) legacy features
    - 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.
Joint Application Design

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.
- 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.
  4. Standardized forms for documenting results.

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
  3. Wrap-up – reporting and summarizing what happened

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.
Roles (con’d)

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.

2. **Session**
   → Session leader welcomes participants, presents task to be discussed, established ground rules and context for discussion, 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.