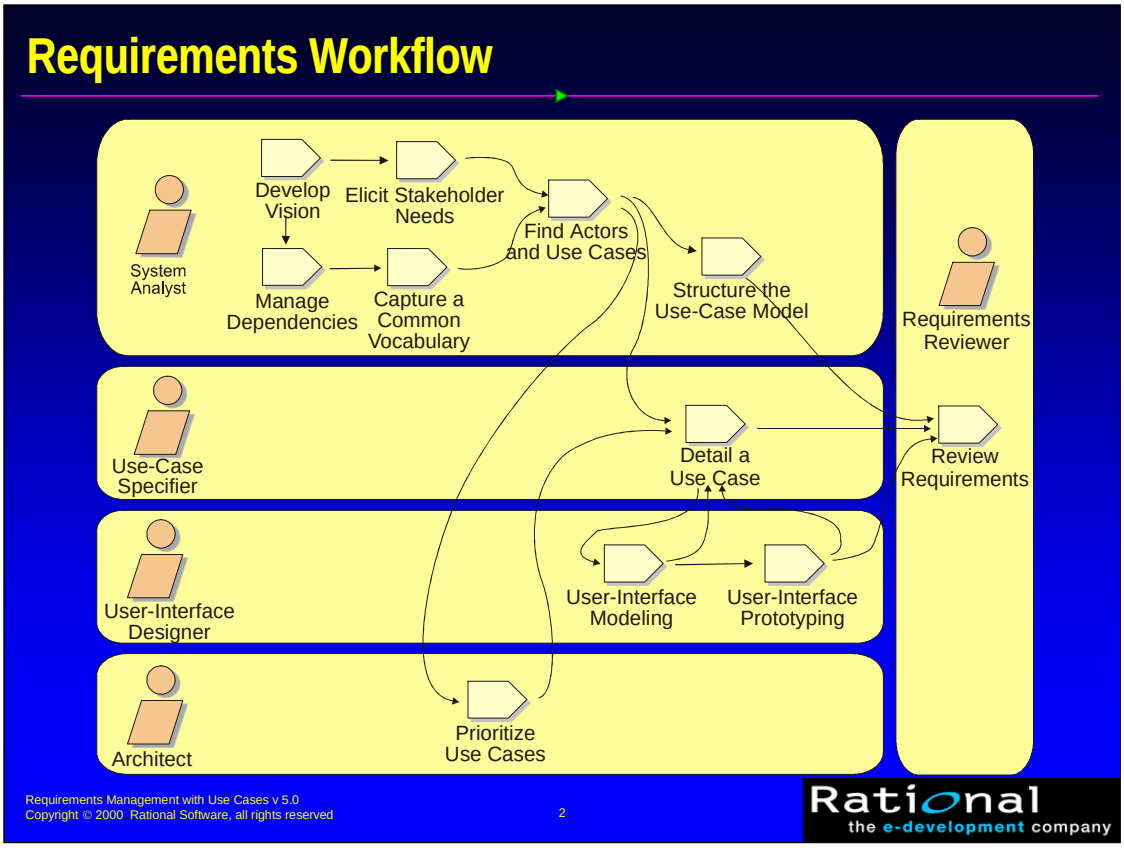
In this module, we describe recommended software development practices and give the reasons for these recommendations.



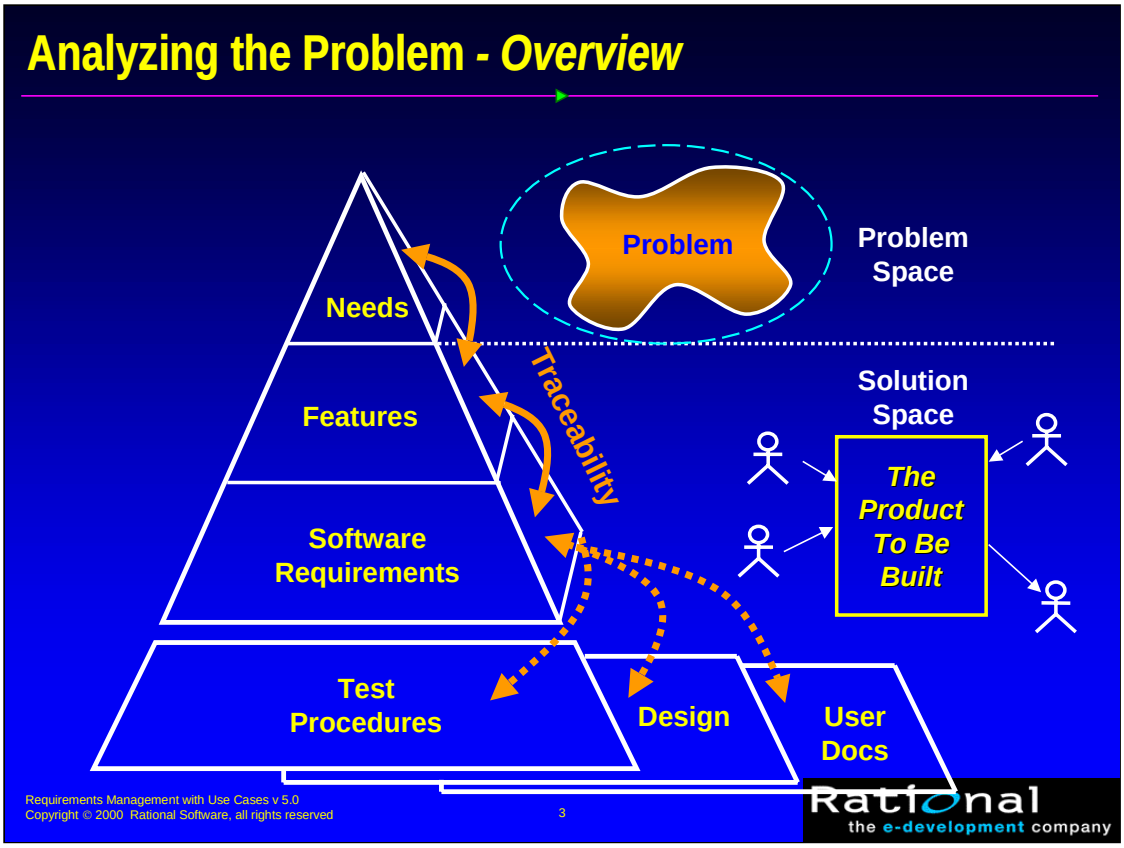
The purposes of the Requirements workflow are:

- ☑ to come to an agreement with the customer and the users on what the system should do
- ☑ to give system developers a better understanding of the requirements on the system
- ☑ to delimit the system
- ☑ to provide a basis for planning the technical contents of iterations
- ☑ to define a user-interface for the system

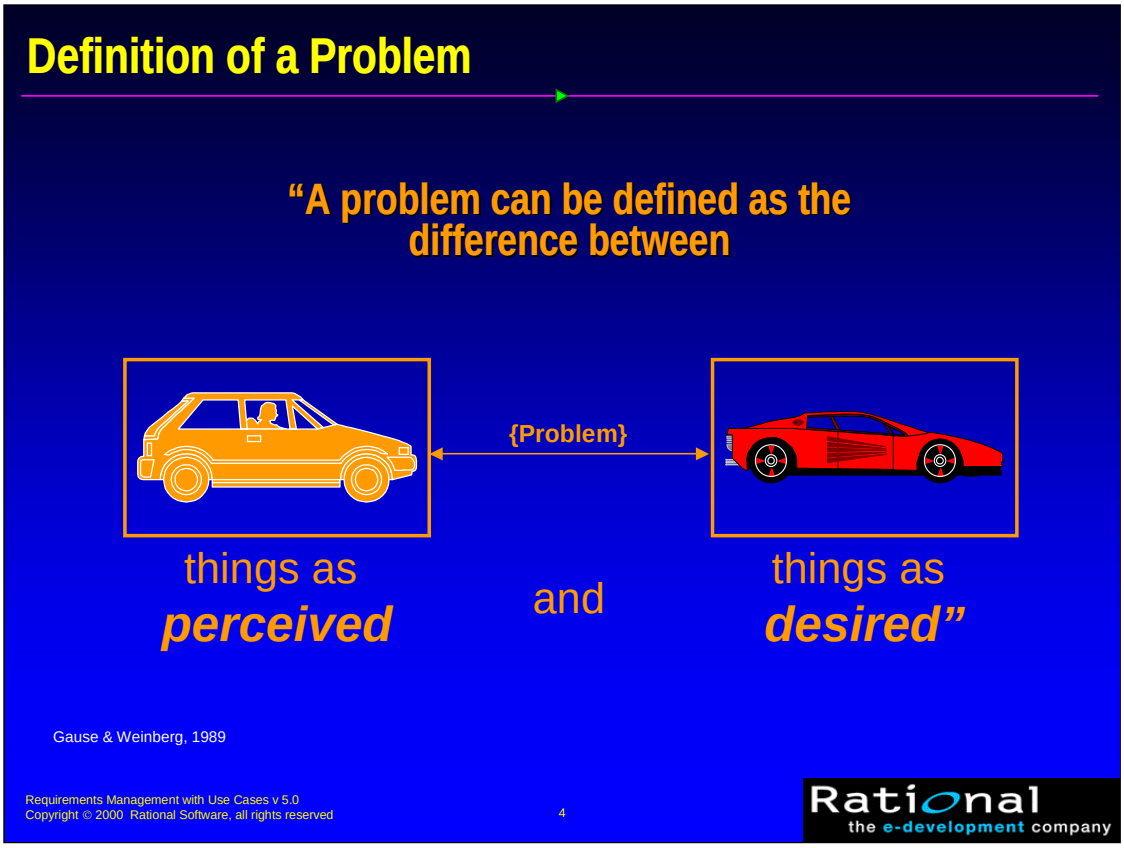
To achieve these goals, a **Vision** document, a **Stakeholder Needs** document, a **use-case model**, and a **Supplementary Specificatio**n document are developed that describes **what** the system will do -- an effort that views customers and potential users as important sources of information (in addition to system requirements).

Complementary to the above mentioned artifacts, the following artifacts are developed:

- ☑ Glossary
- ☑ Use-Case Storyboard
- ☑ Boundary Class
- ☑ User-Interface Prototype



In this module we will discuss some ways to understand the problem we are addressing with the product that will be built.

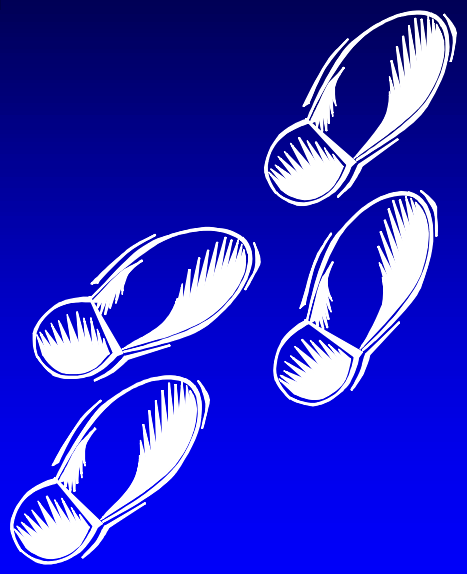


If a difference does not exist between what you perceive you have, and what you desire to have, then there is not a problem. This is especially important when delivering a product to your customer.

One problem we need to address is the gap between our perceptions of what our customer wants and what they actually desire. A solution would attempt to narrow the gap. This might be done by changing the current perception or the desire.

Steps in Problem Analysis

- Step 1 - Gain agreement on the problem being solved.
- Step 2 - Identify the stakeholders.
- Step 3 - Define the system boundaries.
- Step 4 - Identify constraints to be imposed on the system.



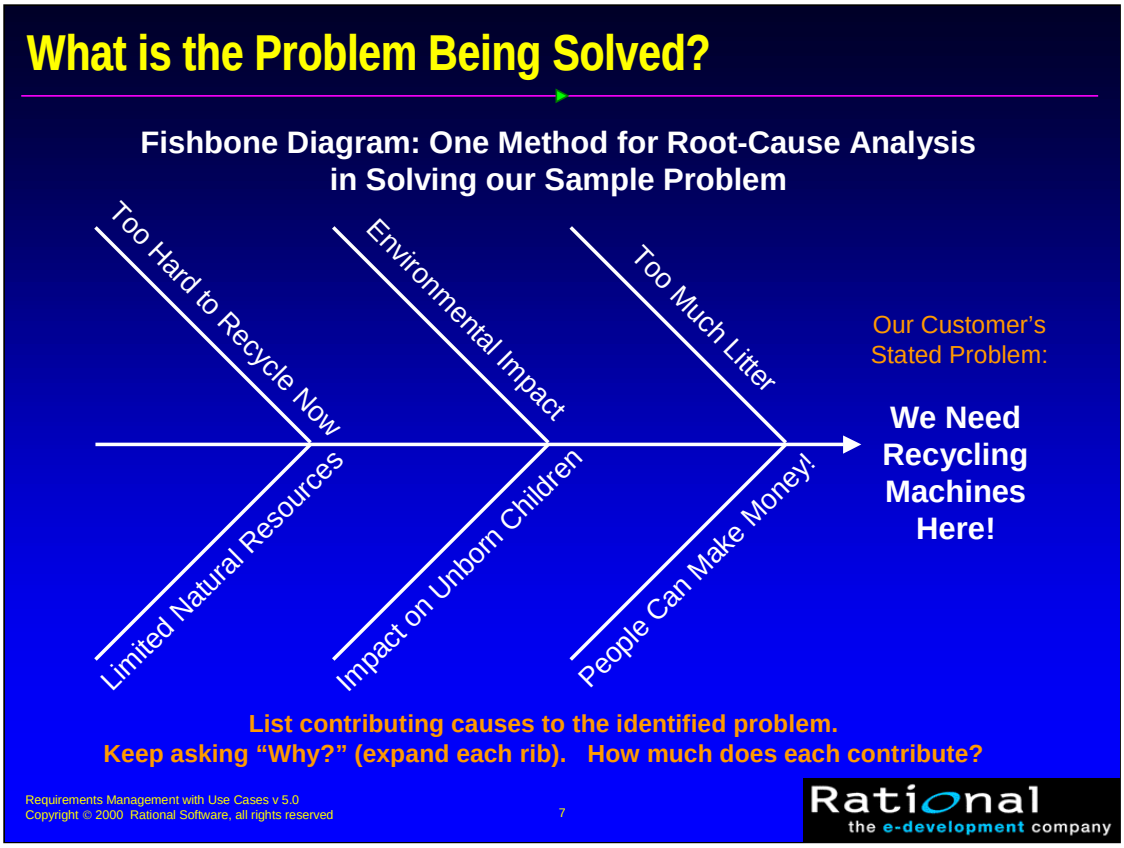
5
Rational
the e-development company

Each of these steps will be discussed in the following slides.

Step 1 - Gain Agreement

- What is the problem?
  - *We technologists tend to rush headlong into solution providing - rather than taking time to truly understand the problem*
  - *Suggestion: Write it down, see if you can get everyone to agree on it*
- What is the problem, really?
  - *Searching for root causes - or the "problem behind the problem" - often leads to a clearer understanding of the real problem*

Don't accept the customer's first statement of a problem. Continue to ask "why?" to find out what the problem "really" is.



What problem will this solve? When we ask the customer what problem will be addressed by this system, he answers, "We need recycling machines here!"

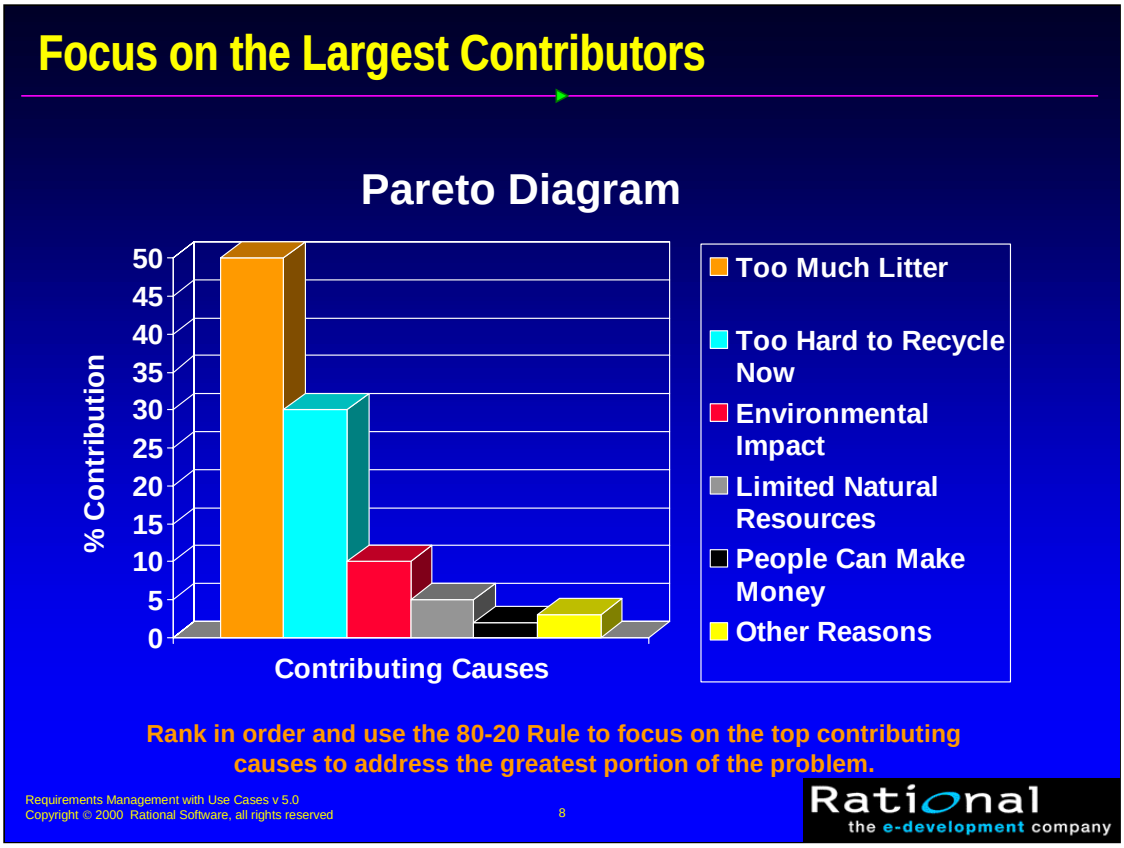
Is this really the true problem?

The fishbone diagram is one method for finding the "root cause" of a problem.

The spines are contributing causes to the problem.

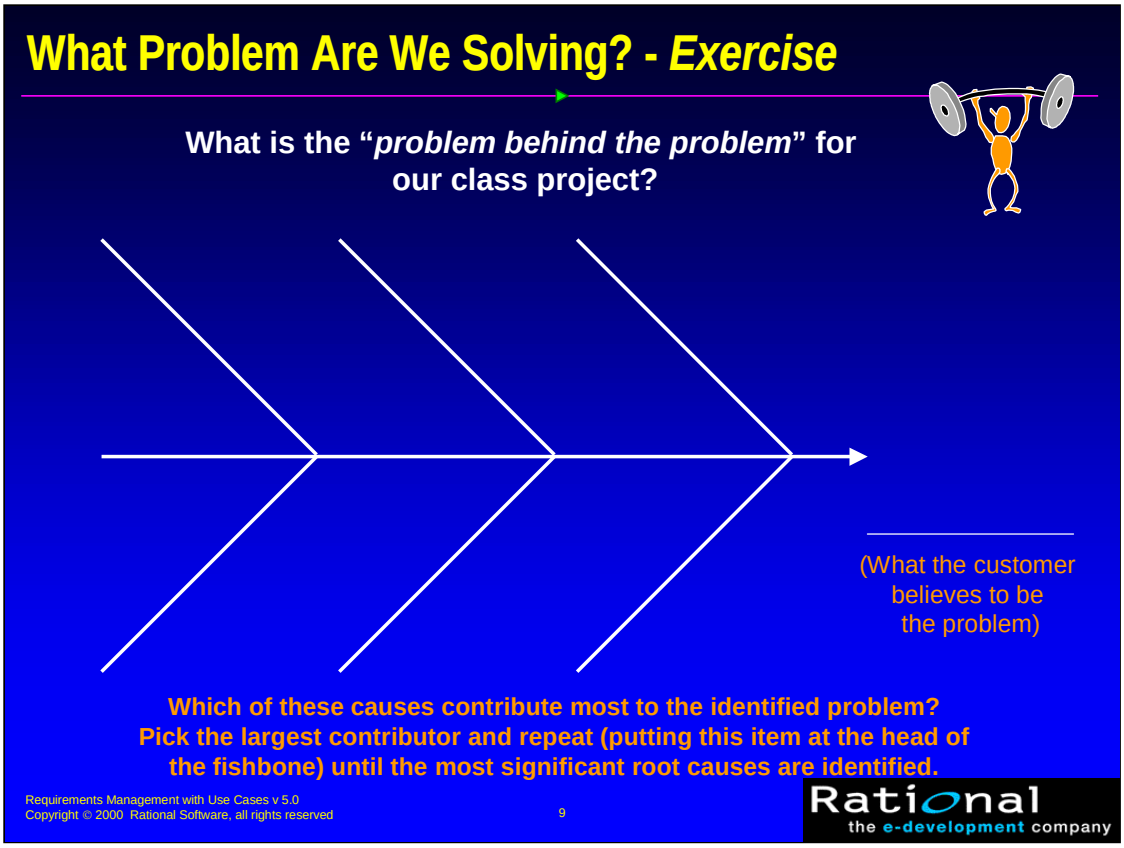
Once these have been defined, try to pick the ones that contribute most to the problem (80-20 rule) and then focus on each of those. Building other spines branching off each could perhaps reveal contributing factors to these causes.

What we often discover is that there may not only be **one** simple problem involved. Instead there may be many dimensions and perceptions of what really is the problem



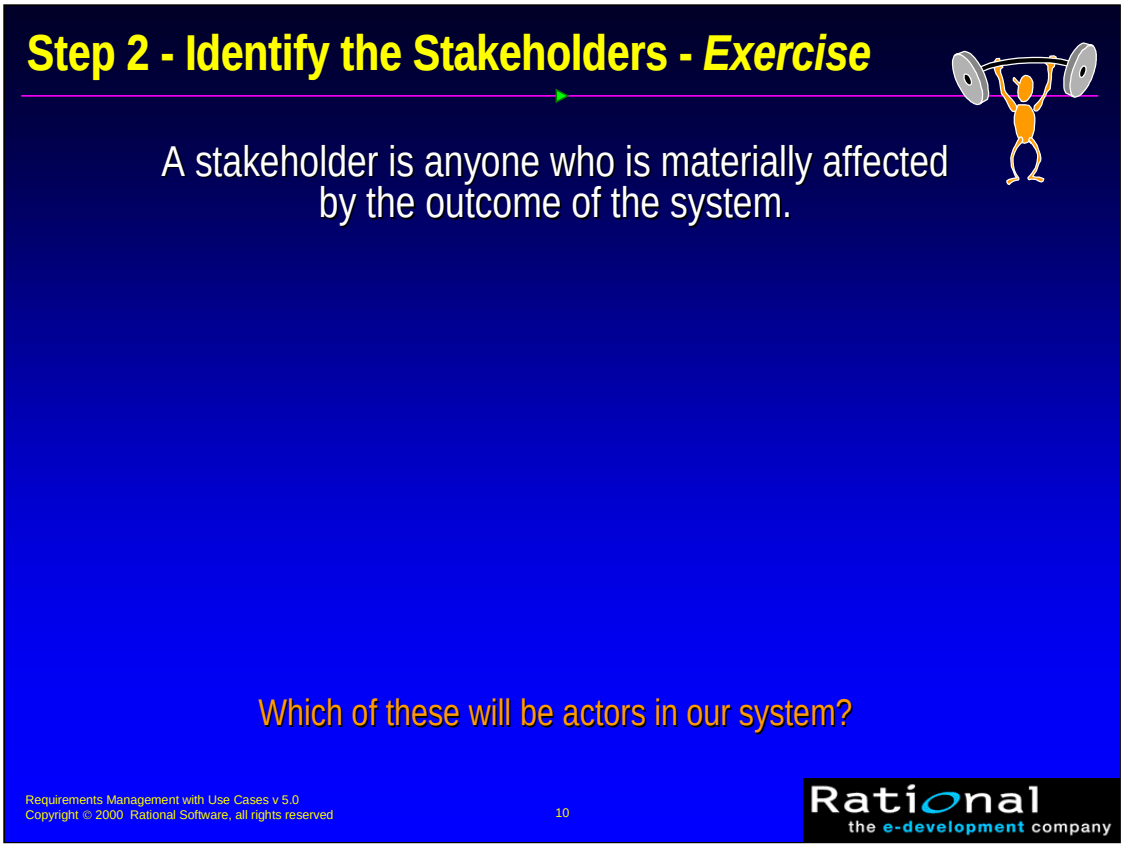
A Pareto chart ranks the contributing causes in the order of their percent contribution to the problem, so it is easier to see which are the largest contributors.

Notice that in this example, 80% of the problem is caused by the top 2 contributors.



The purpose of this exercise is to give you a tool to help you discover any hidden problems that may exist in the problem domain of your product. The "real" problem(s) may not be those most obvious at the beginning.

What problem(s) would your project be addressing?



The purpose of this exercise is to understand the “real” stakeholders in your project -- beyond the obvious customer, who is paying the bill.

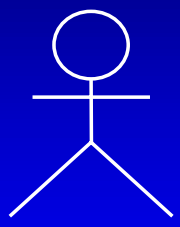
Make sure you consider all those affected by the outcome of the system, including “shareholders”, maintainers, developers, etc.

Who are possible stakeholders for your project?

Based on our definition of actor, which of these will be actors?

Step 3 - Define the System Boundaries

Use Actors to Help Define Boundaries



Actor

- Actors are ***not part of the system***
- Actors represent ***roles*** a user of the system can play
- An actor can represent a ***human***, a ***machine*** or ***another system***
- An actor can ***actively interchange information*** with the system
- An actor can be a ***giver of information***
- An actor can be a ***passive recipient of information***

11

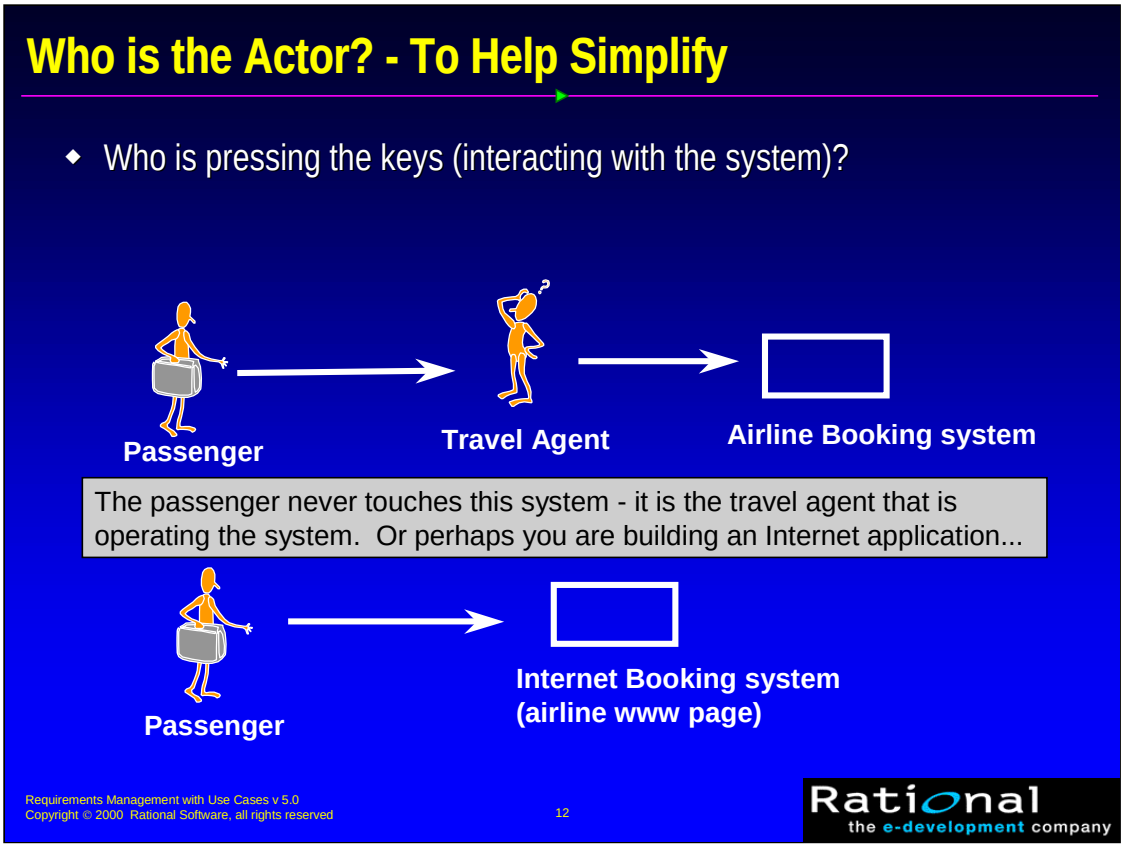
Rational
the e-development company

An **actor** defines a coherent set of roles that users of the system can play while interacting with it. A user can either be an individual or an external system.

To fully understand the system's purpose you must know **who** the system is for, that is, who will use it. Different user types are represented as actors.

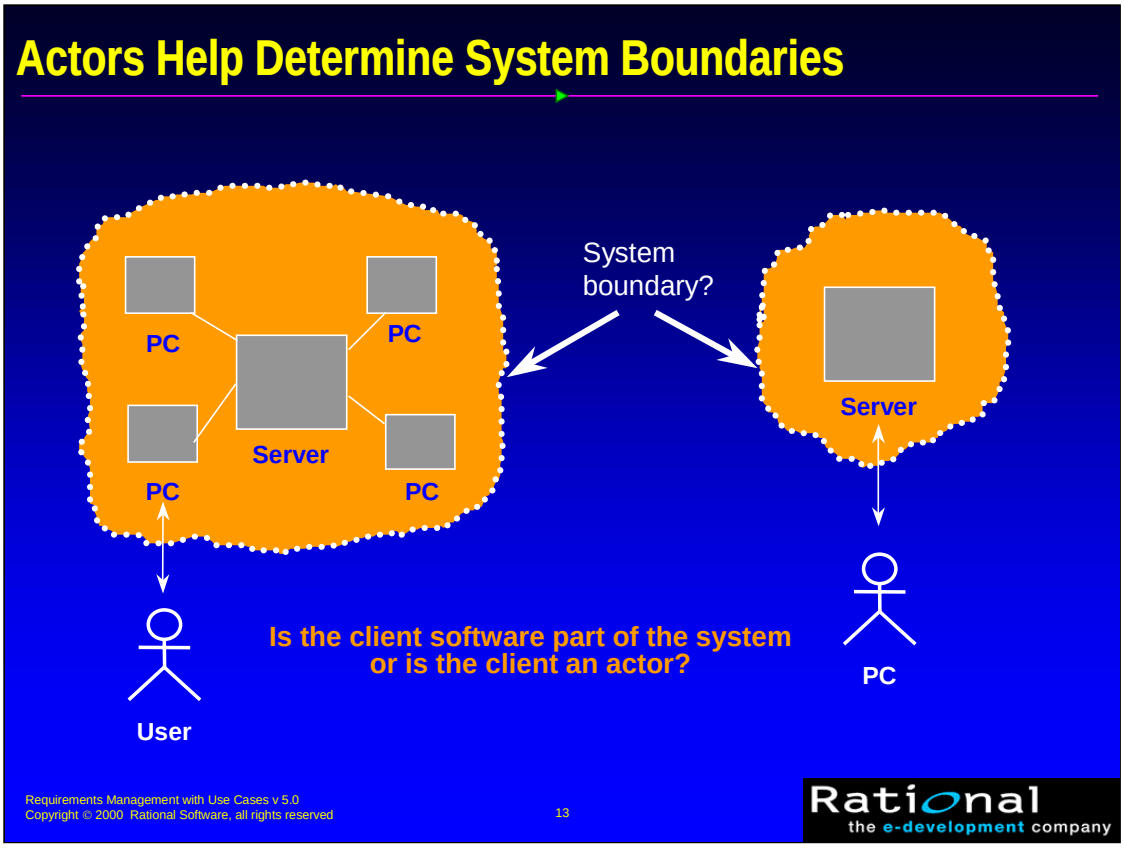
An actor is anything that exchanges data with the system. An actor can be a user, external hardware, or another system.

The difference between an actor and an individual system user is that an actor represents a particular class of user rather than an actual user. Several users can play the same role, which means they can be one and the same actor. In that case, each user constitutes an instance of the actor.

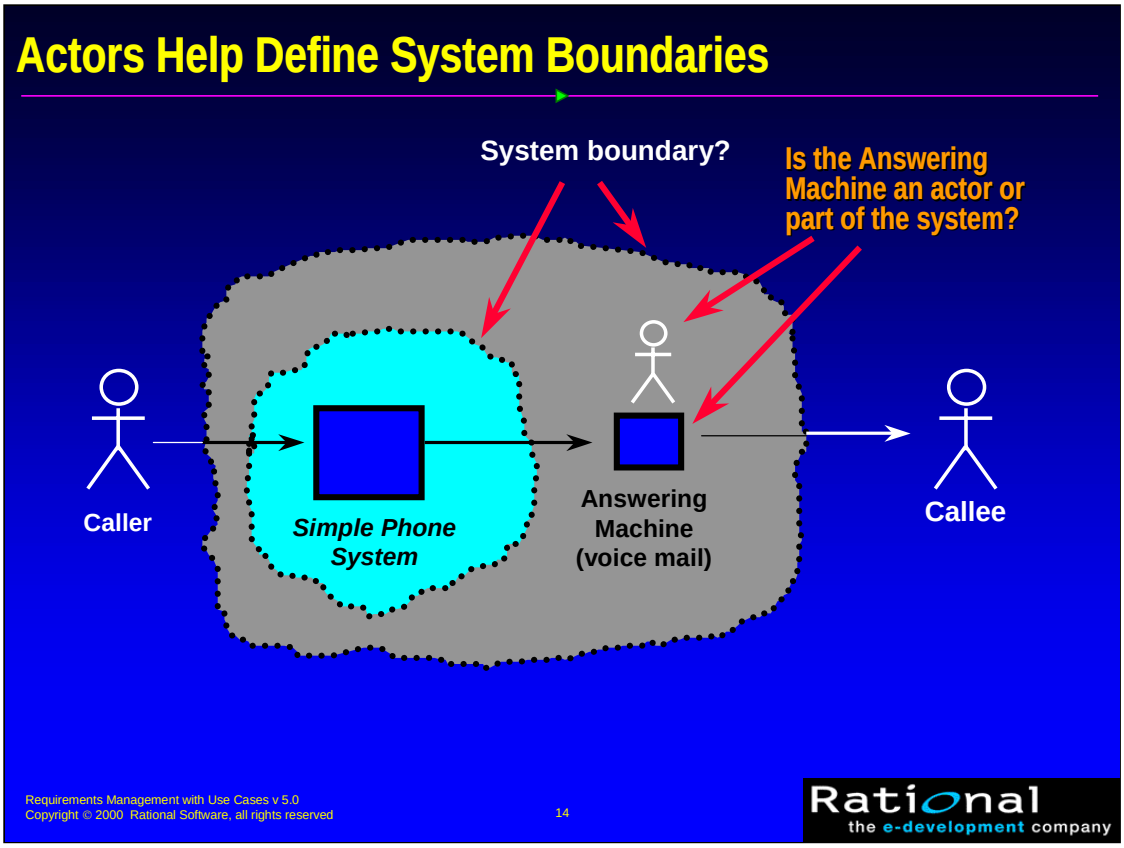


For a person interacting with the system, this is the simplest question to ask: "Who is doing the actual interaction?" This will help determine a name for this role. The actor is the one interacting with the system. If a person is talking with someone else who interacts with the system, then there is the actor.

What if the person is using a speech recognition system? Then the actor is the one talking with the system.



If we are developing the client software for the PCs, then the “User” would be the actor, if we are developing only the server software, then the “PC” is the actor.



An answering machine external to the system would be an actor. If we include voicemail in our system, then the callee is the actor.

Useful Questions in Identifying Actors

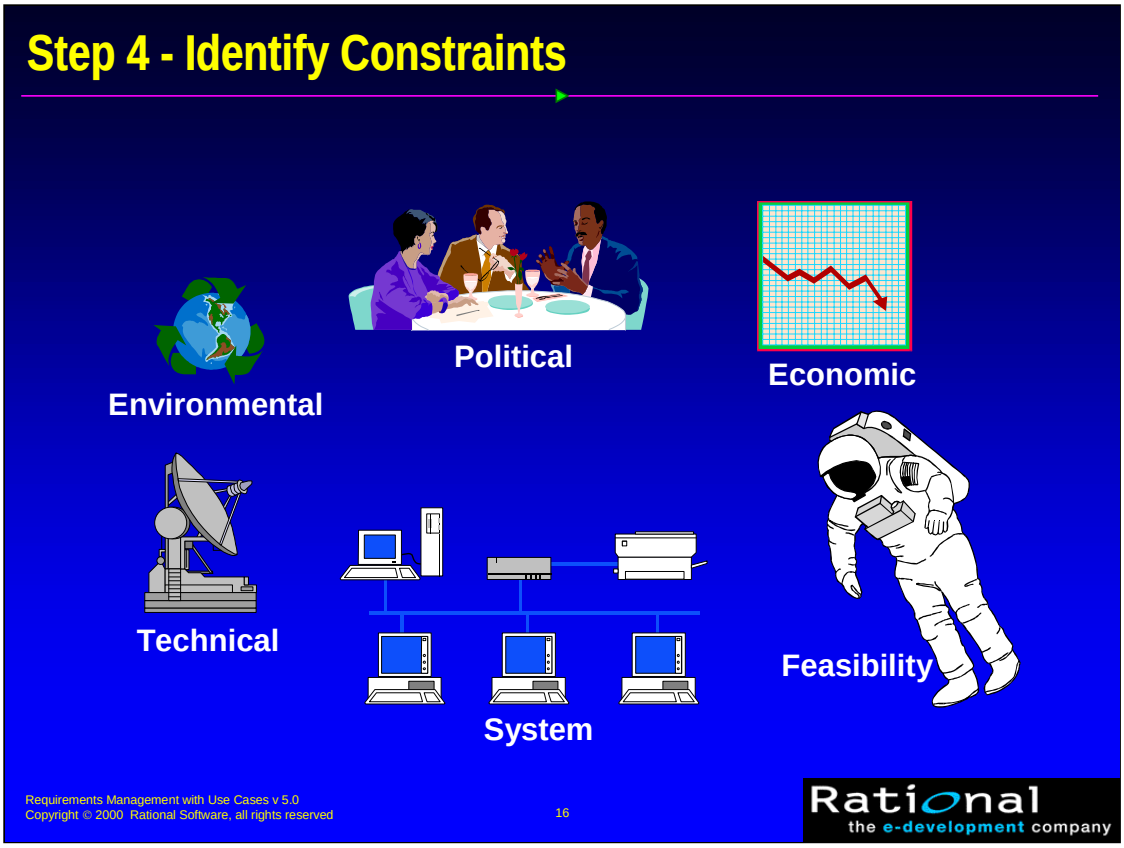
Actor

- Who will supply, use, or remove information?
- Who will use this functionality?
- Who is interested in a certain requirement?
- Where in the organization is the system used?
- Who will support and maintain the system?
- What are the system's external resources?
- What other systems will need to interact with this one?

15

Rational
the e-development company

Here are some useful questions that may be used to help us identify the actors in our system.



What types of constraints have you experienced in your projects?

Developing a Glossary

Glossary

Brief Description
A brief description of the role and purpose of the *Glossary* is included here.

Terms
A list of the terms and their definitions, possibly structured in subsections.

- The glossary defines ***important terms*** used in the project.
- It is important to have a ***common understanding*** of the terminology used in the project.
- A common glossary will help ***prevent misunderstandings*** in the project.
- Start developing the glossary as soon as possible and ***continue throughout the project***.

Look at the glossary at the back of this course material for a sample.

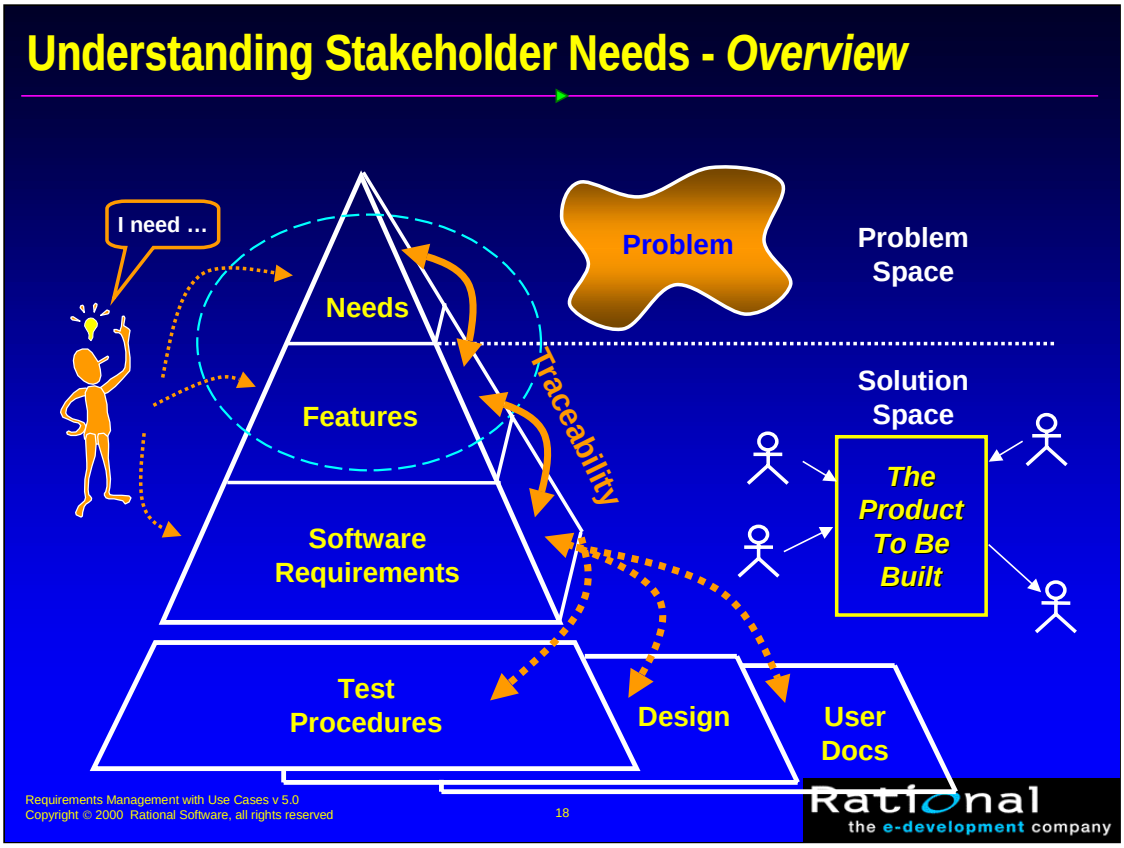
17

Rational
the e-development company

There should be one Glossary for the system. This document is important to all stakeholders, especially when they need to understand and use terms that are specific to the project.

All textual descriptions of the system, especially use-case descriptions should use and refer to the terms defined in the glossary.

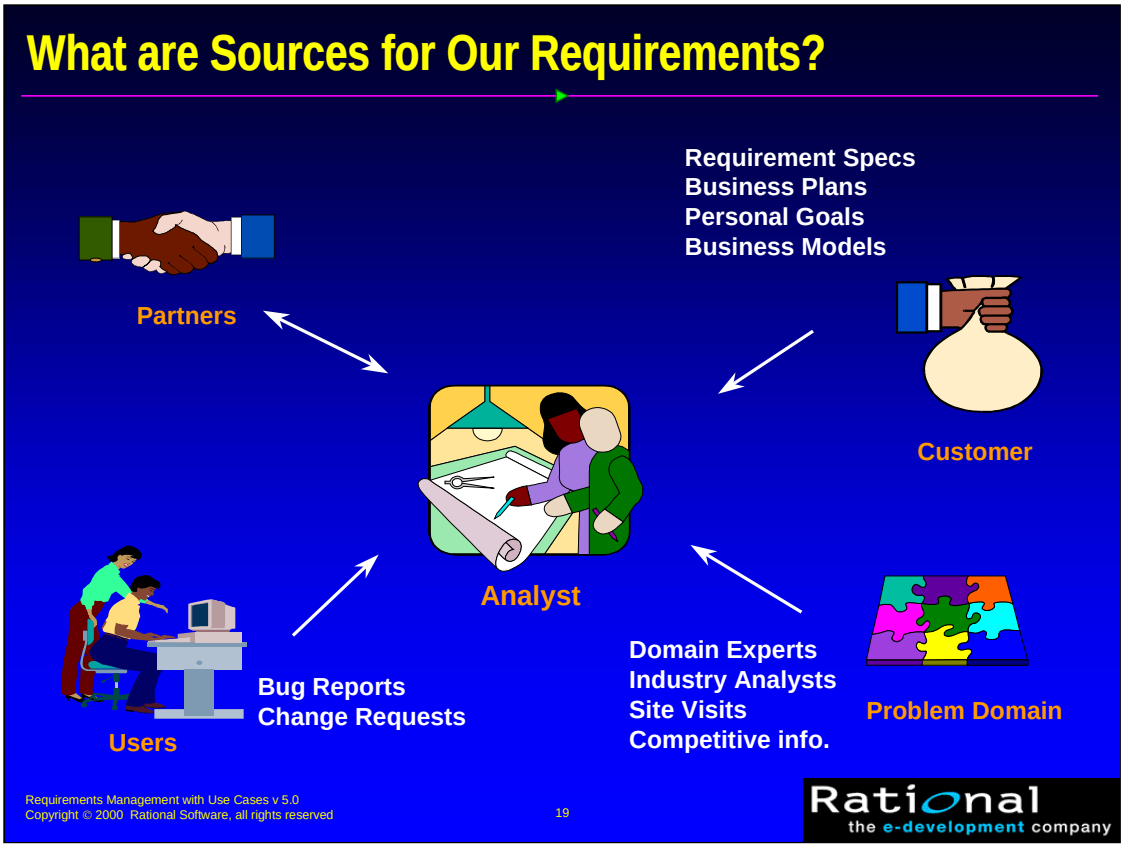
In some cases, you may build a domain model to further visualize the terms in the glossary.



At this point, we will work on making sure that we have considered all stakeholder requests, what needs are addressed by these (relating to the problem we have agreed to address), and what features in the system we might use to address these needs.

Stakeholder requests come in at many different levels of detail -- often not expressed as a real need, but sometimes as a feature of the system, or even as a change to a software requirement.

In this unit we will discuss how we can effectively communicate with our stakeholders to figure out what their “real needs” are.



We need to capture requests from all stakeholders, as well as how these requests will be addressed. Although the system analyst is responsible for gathering this information, many people will contribute to it: marketing people, end users, customers -- anyone who is considered to be a stakeholder in the project.

Other examples of external sources for requirements are:

- ☑ statement of work
- ☑ request for proposal
- ☑ mission statement
- ☑ problem statement
- ☑ business rules
- ☑ laws and regulations
- ☑ legacy systems
- ☑ business models
- ☑ any results from requirements elicitation sessions and workshops

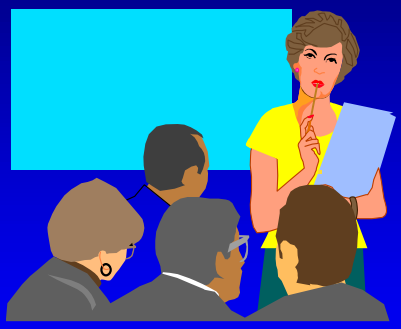
Techniques for Eliciting Stakeholder Needs

- Requirements Workshop
- Brainstorming & Idea Reduction
- Use Cases
- Interviews
- Questionnaires
- Role Playing
- Business Modeling
- Reviewing Customer Requirement Specifications

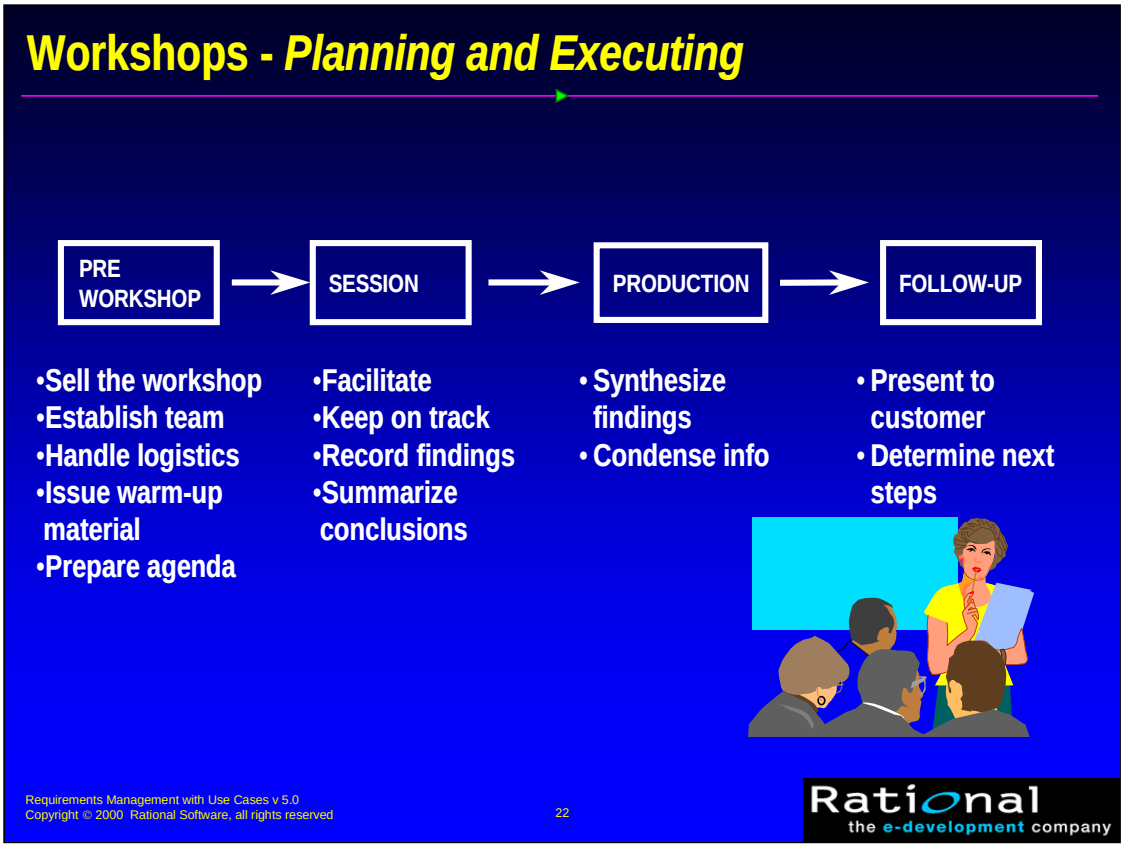
We will discuss briefly each of these techniques in the following slides.

Requirements Workshops

- Accelerate the Elicitation Process
- Gathers all stakeholders together for an intensive, focused period
- Facilitator runs the meeting
- Everyone gets their say
- Results immediately available
- Provide a framework for applying the other elicitation techniques we will be discussing



This is one of the most cost-effective and time-efficient means to get feedback. The same concepts are used in JAD (Joint Application Development) or RAD (Rapid Application Development) sessions.

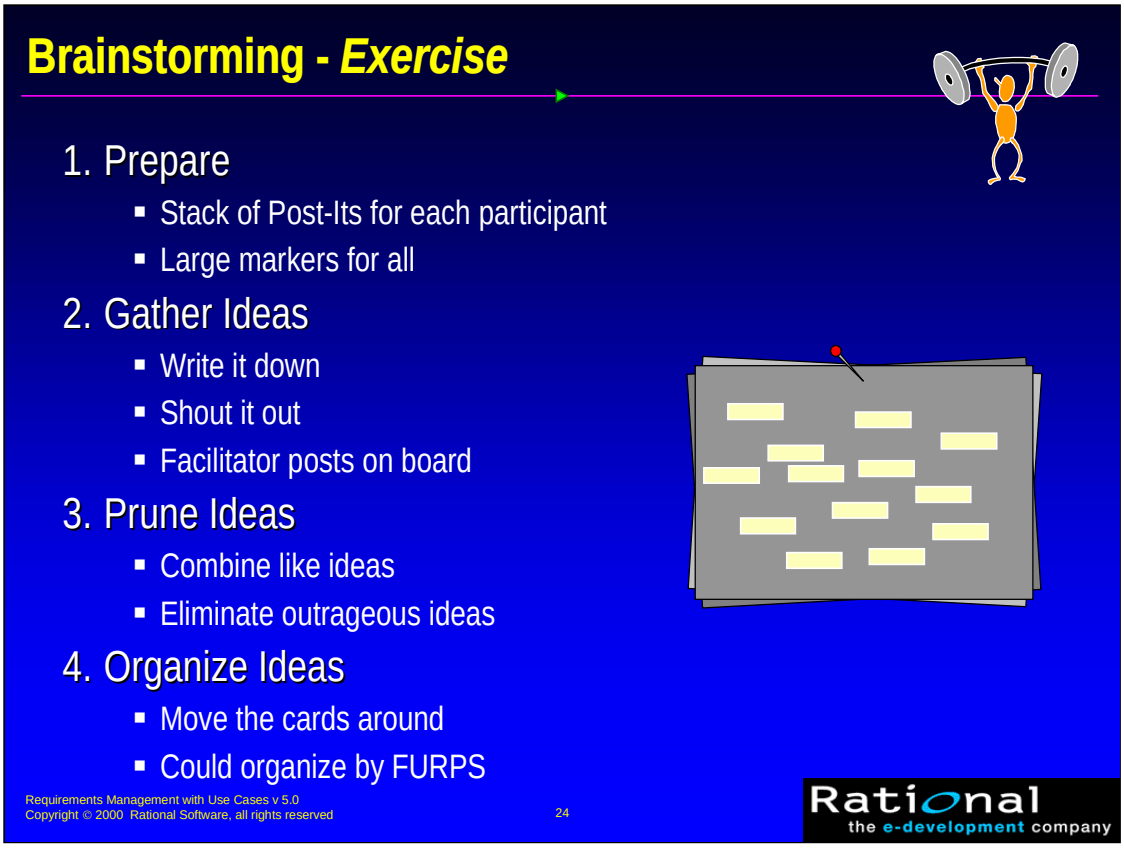


Here are some suggestions on planning and executing a requirement workshop. Make sure you gather the right stakeholders. Who will they be?

Brainstorming

Rules for Brainstorming

- Clearly state the objective of the session
- Generate as many ideas as possible
- Let your imagination soar
- Do not allow criticism or debate
- Mutate and combine ideas



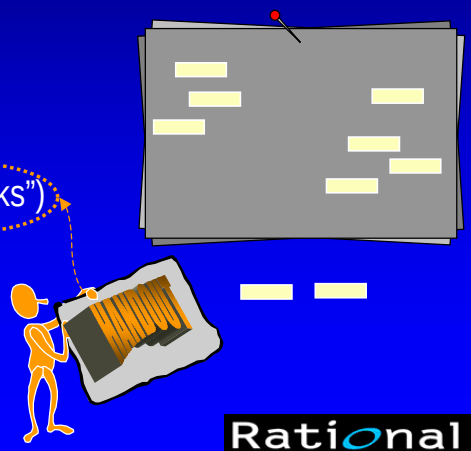
The purpose of this exercise is to help you experience and try some useful techniques that can be used with brainstorming.

Brainstorming - *Idea Reduction*

- Discard redundant and outrageous ideas
- Store “needs more development” ideas
- Blend ideas

and for those that remain ...

- Vote
  - Single vote
  - Cumulative voting
    - Buy features (e.g. “RU Bucks”)
- Apply evaluation criteria
  - Non-weighted
  - Weighted



25

Rational
the e-development company

Now, what do you do after you have gathered all these “great ideas”?

What is a Use Case? - Key Words and Phrases

A

use case

defines a sequence of

actions

performed by a system that yields an observable result of value to an actor

- A use case describes a set (*class*) of possible executions of the system
  - A specific execution (*instance)* of a use case is a ***scenario***
  - Work on the *class* level to identify and describe the use case

- Set of atomic activities, decisions and requests
  - May be performed fully or not at all
  - Started by actor

26

Rational
the e-development company

This slide and the following slide will point out the key principles that should be kept in mind when defining use cases. There are many pitfalls that can be avoided by keeping these key words and phrases in mind.

What is a Use Case? - Key Words and Phrases

A use case defines a sequence of actions

performed by a system → Describes functions of the system

that yields an

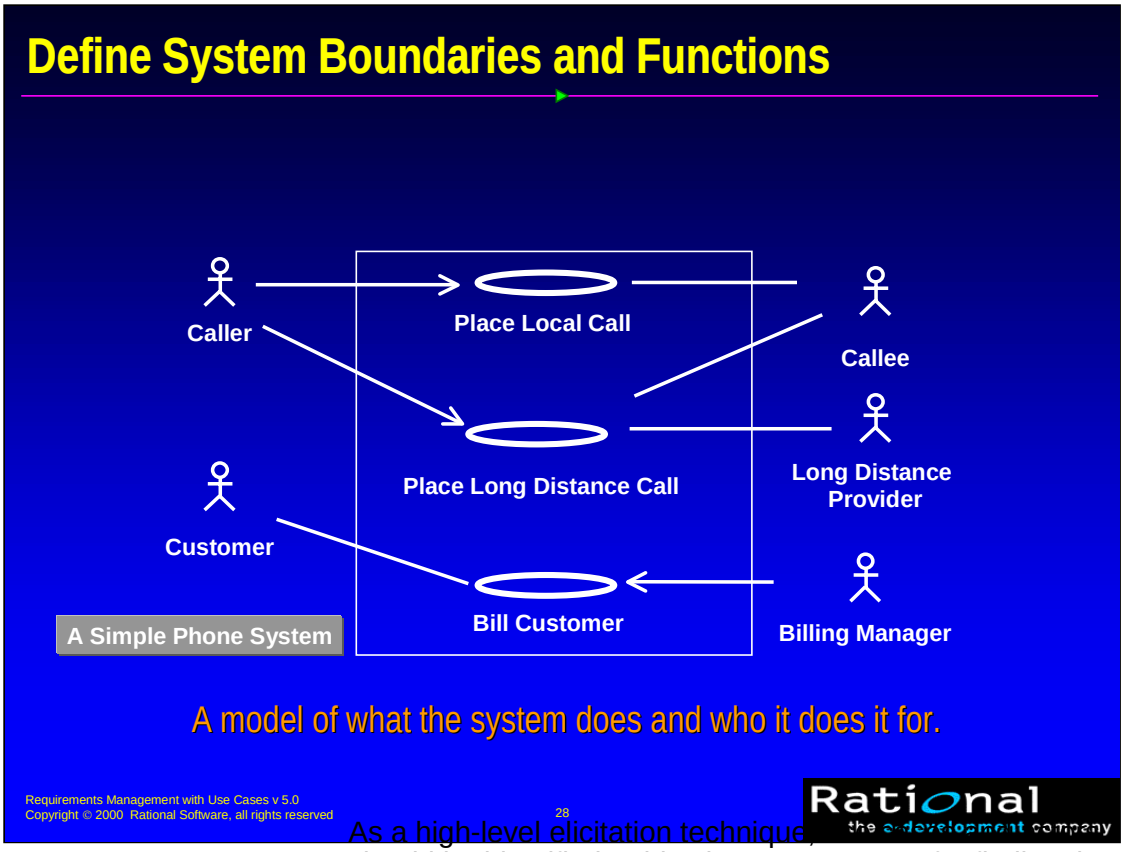
observable result of value → To avoid too detailed use cases

to

an actor → To avoid too complex use cases

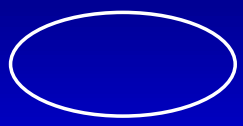
27

Rational
the e-development company



As a high-level elicitation technique, [illegible] should be identified. This gives us a context for finding the needs of the stakeholders in terms of the functionality and interfaces of the system.

Useful Questions in Identifying Use Cases

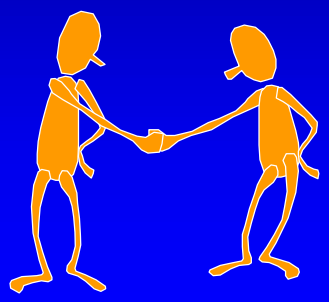


- What are the primary tasks the actor wants the system to perform?
- Will the actor create, store, change, remove, or read data in the system?
- Will the actor need to inform the system about sudden, external changes?
- Does the actor need to be informed about certain occurrences in the system?
- Will the actor perform a system startup or termination?

Here are some questions that are useful for helping you find the use cases in your system.

Interviews

- A direct technique that can be used in both problem analysis and requirements elicitation
- Designed to gain an understanding of real problems and potential solutions from the perspectives of the users, customers and other stakeholders



30
Rational
the e-development company

Interviewing is probably the most useful technique to directly get needs from your stakeholders. It is helpful to interview key stakeholders for your project.

Questionnaires

- Widely used
- Appear scientific because of statistical analysis
- Applicability to broad markets where questions are well defined
- Assumptions
  - Relevant questions can be decided in advance
  - Phrased so reader hears in intended way
  - Suppresses much that is good about analysis
- Can be powerful, but not a substitute for an interview

31

Rational
the e-development company

These can be useful for gathering data, but it should still be prefaced by interviews to determine the correct questions to ask. How many customers should be interviewed to determine a good cross-section of questions? Usually 12-15 is adequate.

Most items should be categorized to allow for statistical analysis.

Always leave some open-ended questions to allow for new ideas.

Role Playing

- Tools and Techniques
  - Scripted walkthroughs
  - Scenario analysis
  - Class Responsibility Collaboration (CRC) Cards

- Have the analyst play the role of user or customer to gain real insights into the problem domain
- Have the customer play the role of a user to understand the problems they may face

32
Rational
the e-development company

If the people you are getting requirements from are not the actual user, sometimes it can be useful to have them take on the role of a user and walk through some user scenarios.

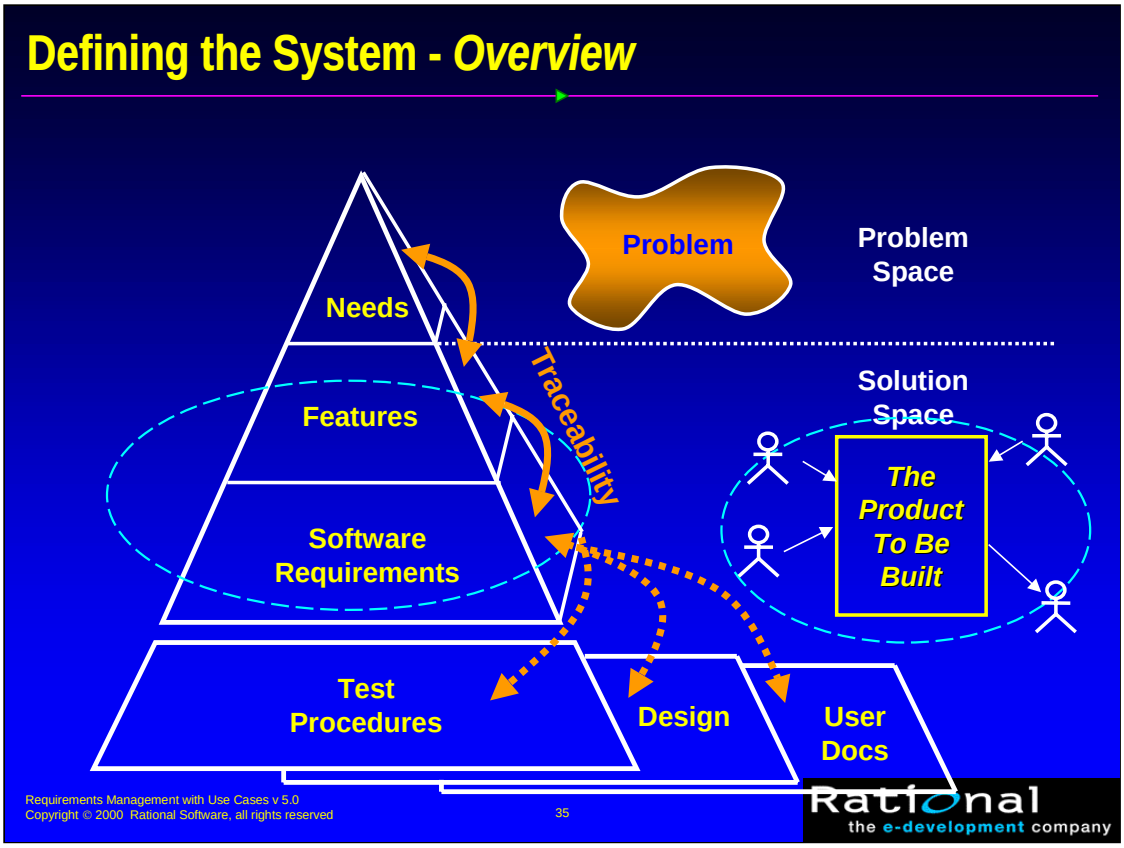
What About Business Modeling?

- From a business perspective, a business model may be used to:
  - Understand the organization
  - Visualize the organization and its processes
  - Find ways of making the organization more efficient
  - Re-engineer the organization
  - Provide proof that the information technology adds value

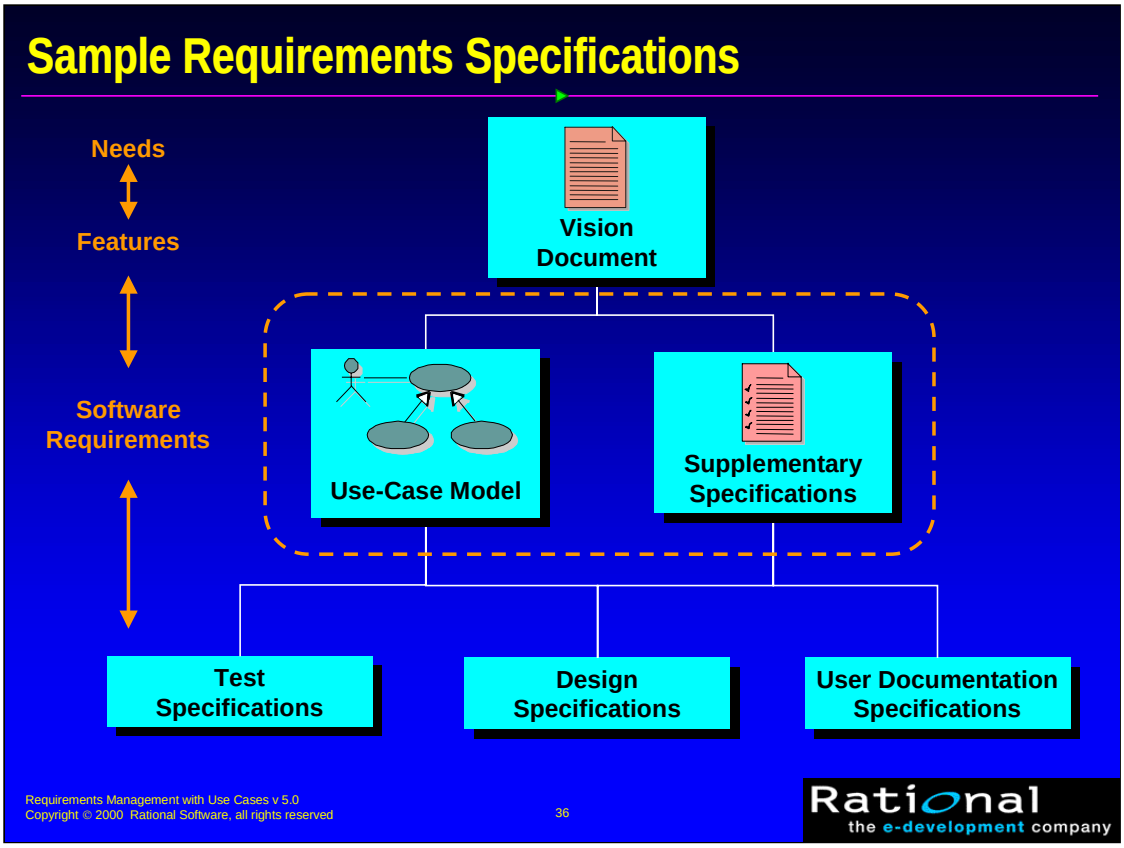
The approach to business modeling presented in the Rational Unified Process includes a concise and straightforward way to generate requirements on supporting information systems. A good understanding of business processes is important in order to build the right systems. Even more value is added if you also use people's roles and responsibilities, as well as definitions of what "things" are handled by the business, as a basis for building the system. It is from this more internal view of the business (captured in a business object model) that you can see its tightest link to other models of the system and how they should appear.

Why Itemize Requirements?

- Know what the requirements are
- Know how many requirements you have
- Use to establish a system baseline
- Basis for scope management and change control
- Basis for project management
- Basis for test
- Basis for meaningful metrics
  - Number of requirements implemented
  - Number of requirements changes in period



In this module we will focus on determining the features for our product and begin to develop the use-case model for our system.



Here is the approach that we will use in this class for structuring our requirements and their relevant documentation. We will be focusing especially on the Vision Document and the Software Requirement Specification, or SRS. This model is made up of Use Cases and Supplementary Specifications.

Understand that these will be used to drive the specification of the lower level documents -- for test, design and user documentation -- this will not be discussed in this course.

The Vision Document

- System-level document which describes the “Whats” and “Whys” of the product or application
- Focus is on:
  - User needs
  - Goals and objectives
  - Target markets
  - User environments and platforms
  - Product features

Vision Document

37

Rational
the e-development company

The Vision document provides a high-level (sometimes contractual) basis for the more detailed technical requirements. There can also be a formal requirements specification. The Vision captures very high-level requirements (features) and design constraints, to give the reader an understanding of the system to be developed. It provides input to the project-approval process, and is therefore intimately related to the Business Case. It communicates the fundamental "whys and whats" related to the project and is a gauge against which all future decisions should be validated.

In general, the Vision document will be read by managers, funding authorities, workers in use-case modeling, and developers .

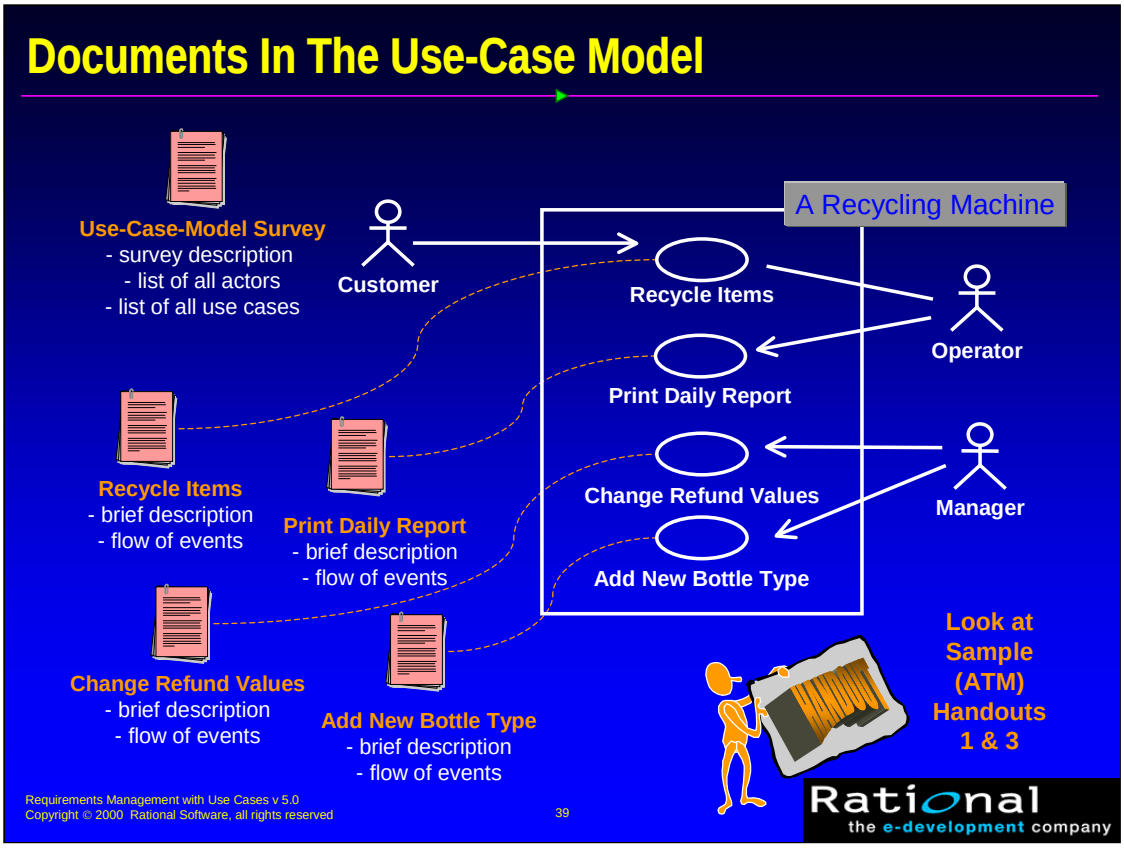
Role of the Vision Document

- Communication between management, marketing and the project team
- Initial customer feedback document
- Foster understanding of the product in its most general terms
- Establish general scope and priorities of high level features
- Record future features and ideas

A document which gets
"all parties working from the same book"

38
Rational
the e-development company

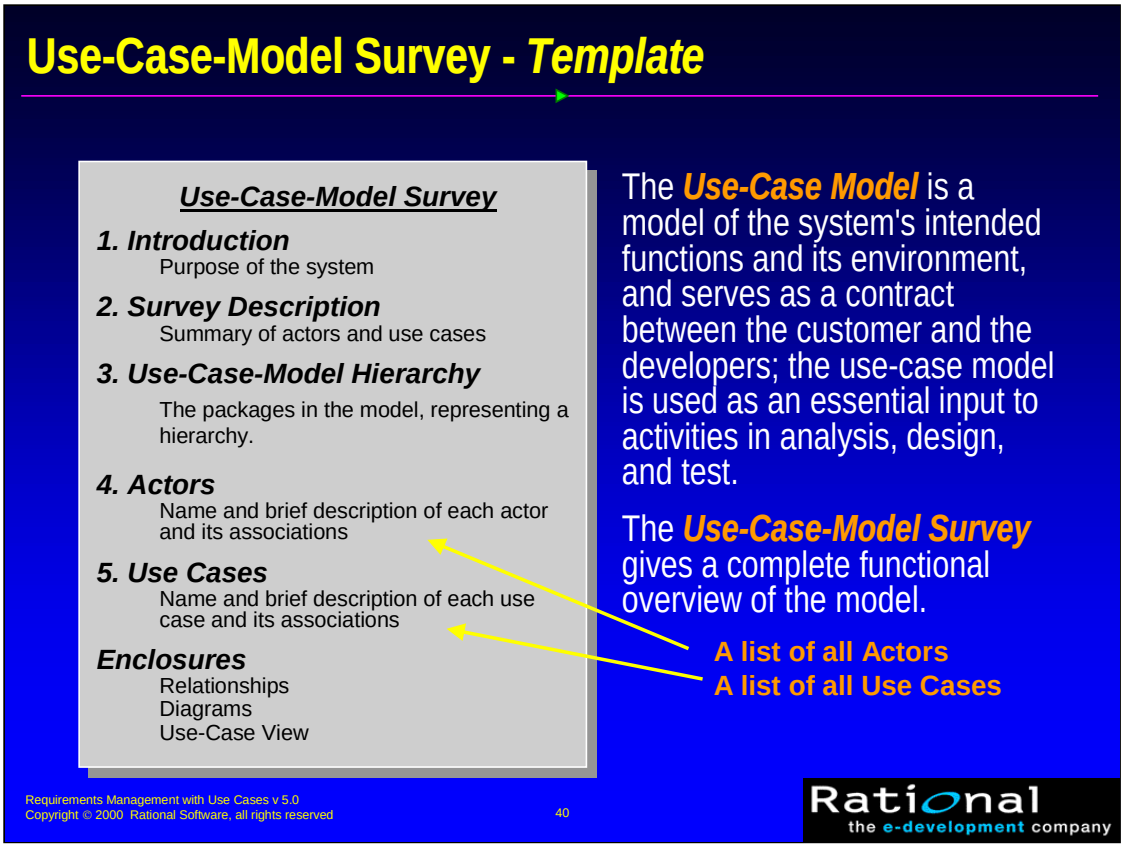
The primary purpose of "any" document is communication. We should always be aware of who will be reading the document and ensure that they understand its content.



The important part of the use-case model is the text. Frequently, many people get the wrong idea of the word “modeling”, and believe use cases are just about drawing figures and arrows. Use cases involve writing text. Drawing the pictures are, at the most, 1/4 of the effort -- usually much less. Don’t stress the modeling work - stress the text work -- a use case without the text is nothing.

More than 75% of all effort during the requirements capture is writing the textual description of the flow of events.

Again, you may want to look at the sample Use-Case Model included in the student handouts.



This report describes the use-case model comprehensively in terms of how the model is structured into packages and what use cases and actors there are in the model. If you are using packages, the document shows the model structure hierarchically. The report can be used to describe the entire use-case model at different stages:

- ☑ during inception, such as when you have defined the scope of the system
- ☑ during elaboration, such as when the use-case model is more stable
- ☑ during construction, when requirements is complete

This report is used by various people interested in the use-case model, such as the customer, users, architects, use-case authors, designers, use-case designers, testers, managers, reviewers, and writers.

Brief Description of an Actor - *Examples*

- Customer
  - The Customer collects bottles, cans and crates at home and brings them back to the shop to get a refund.
- Operator
  - The Operator is responsible for maintenance of the recycling machine.
- Manager
  - The Manager is responsible for questions about money and the service the store delivers to the customers.

41
Rational
the e-development company

Note: The Actor name is a “label” which can be used to refer to it throughout the model. The brief description should clarify any ambiguity that the name might suggest

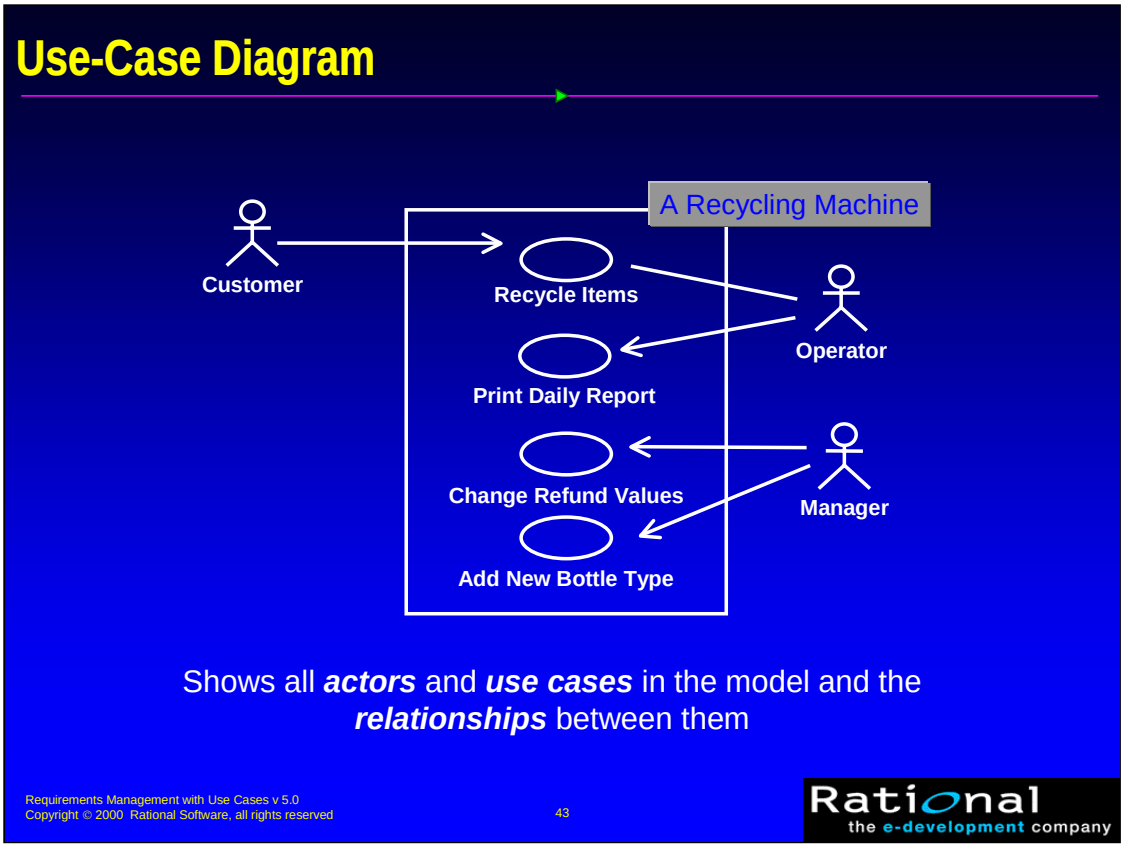
Suggestion: Try to avoid an actor called “User” -- try instead to figure out the role of that particular user.

Brief Description of a Use Case - *Examples*

The brief description reflects the role and purpose of the use case.

- Recycle Items
  - The user uses this machine to automatically have all the return items (bottles, cans, and crates) counted, and receives a receipt. The receipt is to be cashed at a cash register (machine).
- Add New Bottle Type
  - New kinds of bottles can be added to the machine by starting it in 'learning mode' and inserting 5 samples just like when returning items. In this way, the machine can measure the bottles and learn to identify them. The manager specifies the refund value for the new bottle type.

Two samples from the recycling machine.



Often a global Use-Case diagram will be included in the Use-Case-Model Survey to give a graphical overview of the system.

This should include “all” use cases, actors, and their relationships.

Describing a Use Case - *Step-by-Step Outline*

Recycle Items

Usually just handwritten at this point.

Brief Description

The user uses this machine to automatically have all the deposit items (bottles, cans, and crates) counted, and receives a receipt. The receipt is to be cashed at a cash register

Outline for Flow of Events **[Basic Flow, in step-by-step format]**

1. The customer presses the Start-Button.
2. The customer inserts deposit items.
3. The system checks the type of the inserted deposit items.
4. It increments the day's total of the types of items received.
5. The customer presses the Receipt-Button.
6. The system prints out the receipt.

(ATM) Handout 2

44

Rational
the e-development company

Two samples from the recycling machine (again).

Look again at the sample step-by-step outlines in the ATM Use-Case handout. These are not official documents, but a start toward developing a document. These would probably be sketched on easel paper at a Requirements Workshop.

Identify Alternative Flow of Events

- Purpose:
  - Find all possible scenarios for the Use Case.
  - List all questions to ask the user.
- Procedure:
  - Work on paper with the Users.
  - Ask - what may go wrong?
  - Ask - what may not happen?
  - Ask - what kind of resources can be blocked?
  - Enumerate them A1, A2, A3 and so on.
  - You can describe them in detail but usually it is enough to just identify them.

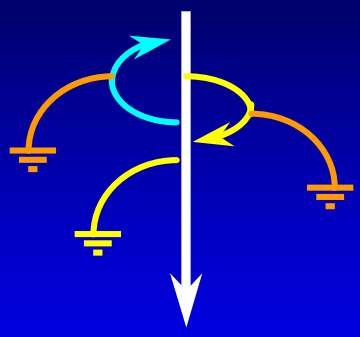
45
Rational
the e-development company

Alternative flows are the most important questions for determining the system architecture. Here are all of the descriptions of how the system should behave when things go wrong! This is important to be included in the use cases.

When working on the alternative use cases, questions regarding the behavior of the system are bound to surface. List these questions and have the users answer what the system is supposed to do, and how they expect to interact with the system in this scenario. This is an important step in clarifying the requirements.

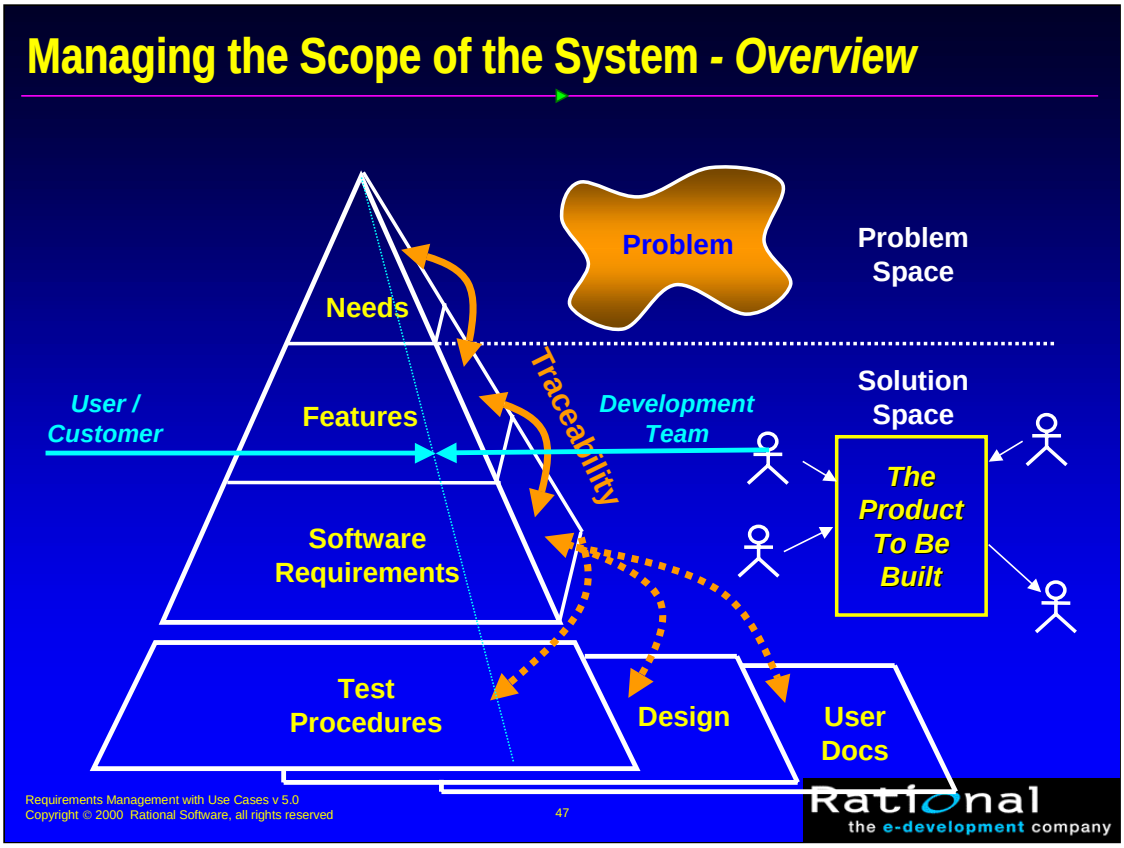
Use Case - *Different Flows of Events*

- Has one normal, *basic flow* (Happy-Day Scenario!)
- Several *alternative flows*
  - **Regular variants**
  - **Odd cases**
  - **Exceptional flows** handling error situations



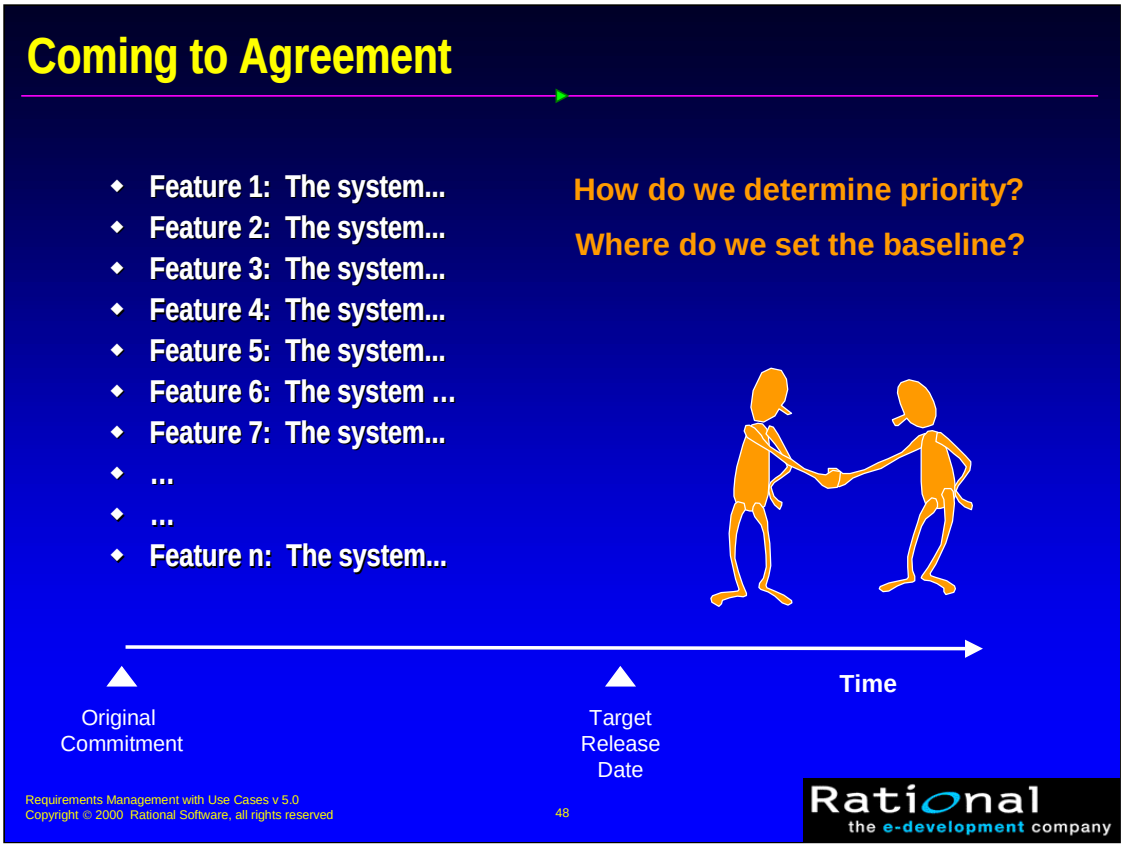
46
Rational
the e-development company

The description should cover all the flows, the normal as well as the alternative and exceptional ones. Structure the description in such a way that it is easy to follow the different flows and to be able to understand what happens when.



Scope management is maintaining the “healthy tension” between what the customer wants (maximum features) and what development believes they can deliver in a fixed time frame.

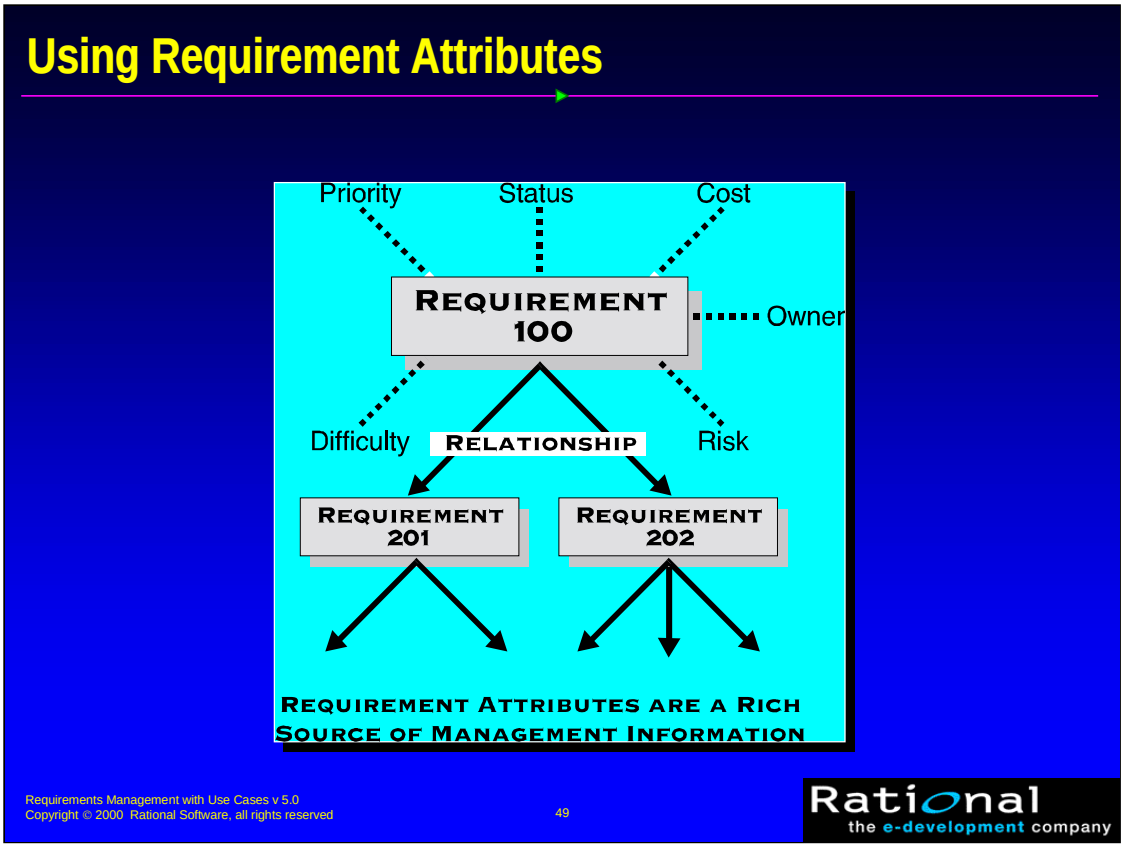
A good way to achieve this is by using iterative development and providing incremental “slices of the pie”.



Where do we draw our baseline?

The key is to under-promise and over-deliver -- but not too much! We want to maintain our credibility.

What factors might influence the order in which we rank these?



Wouldn't it be helpful if we could collect all these attributes for each of our different types of requirements and then have them available later when we need to make important decisions regarding them?

Requirement Attribute Guidelines

Sample Attributes

Rationale	Reason for the requirement
Development Priority	Order/priority of development
Status	Proposed, Approved, Incorporated, Validated
Risk	Probability of adverse project impact- schedule, budget, technical
Safety/Criticality	Ability to affect user health, welfare, or economic consequence of failure
Responsible Party	Pull down list
Origin	Source of requirement
Stability	Probability understanding of requirement will change

50
Rational the e-development company

Here is a proposed outline of a Requirements Attributes Guidelines document:

1. Brief Description

A brief description of the role and purpose of the Requirements Attributes Guidelines.

2. References

A description of related or referenced documents.

3. Requirement Attributes

3.1 Attributes for <type of requirement>
For each type of requirement you have identified, list what attributes you will be using, and briefly explain what they mean. For each requirements attribute, specify a value type and a default value.

4. Traceability Criteria

4.1 Criteria for <type of requirement>
For each type of requirement you have identified, list what criteria (what you should trace to) you use when establishing traceabilities.

Uses for Requirement Attributes

- Assigning resources
- Assessing status
- Calculating Software metrics
- Managing project risk
- Estimating costs
- Assuring user safety
- Managing project scope

51
Rational
the e-development company

What can we use these attributes for?

Using Attributes to Prioritize - *Exercise*

Features	Priority	Difficulty	Risk	Stability	Action
FEATURE1: Save and restore sort and filter criteria	Med High	Low	Low	High	
FEATURE2: Ability to save a Requisite document as a Word document.	Med High	Low	Low	High	
FEATURE3: Ability to see deleted requirements in a view window.	Medium	Med High	Medium	Medium	
FEATURE4: Support for Currency datatype attributes.	Medium	Medium	Med Low	Medium	
FEATURE5: Support the "All" document type (provides an easy way to define common attributes across multiple document types).	Med High	Medium	Medium	Med High	
FEATURE6: Ability to select requirement in a view and GoTo in Word document.	Med High	Medium	Medium	Med High	
FEATURE7: Display a requirement's attribute in the text of the requirement's document.	Medium	Medium	Medium	Med High	
FEATURE8: New project wizard	Med High	High	Med High	Medium	
FEATURE9: Fast creation of a requirement (avoid the requirement dialog on creation).	Med High	Med Low	Med Low	High	
FEATURE10: Autosave of a project (project archive).	Medium	Med Low	Medium	Medium	
FEATURE11: Change one or more attributes for a selected set of requirements.	Medium	Med High	Medium	Medium	
FEATURE12: Ability to Clone a project's structure to allow users to easily create new projects from old.	High	Medium	Medium	Low	
FEATURE13: Performance enhancements for printing, requirement identification.	High	Med High	Medium	Med High	
FEATURE14: Windows95 Port.	High	Medium	High	High	

52
Rational
the e-development company

The purpose of this exercise is to explore how we might use attributes of requirements (in this case, Features) to make a decision about the relative importance of each when determining what to cut when managing scope.

Each group should come up with a ranking showing the order in which the tasks should be considered for inclusion in the release of the product, using the following input (attributes):

- ☑ the text of the requirement
- ☑ priority (input from the customer)
- ☑ difficulty (input from development)
- ☑ risk (to the project)
- ☑ stability (of the requirement)

Gain Control of the Requirements Output

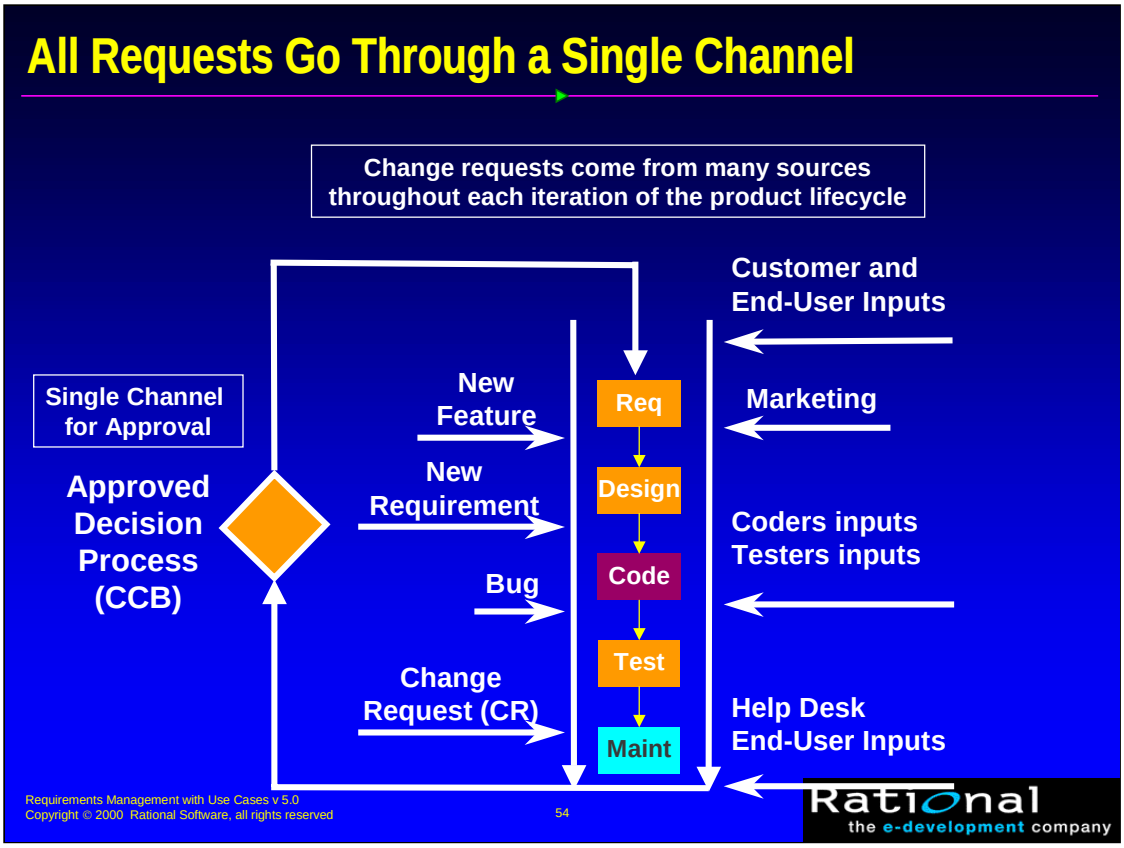
- Document the requirements
  - Write them down!
  - Collect in a central repository
  - Make them accessible and visible to all stakeholders
  - Keep them reasonably stable
  - Control the changes (avoid “scope creep”)
- Use requirement attributes to gather useful information from the right people.

Weinberg, ‘95

53

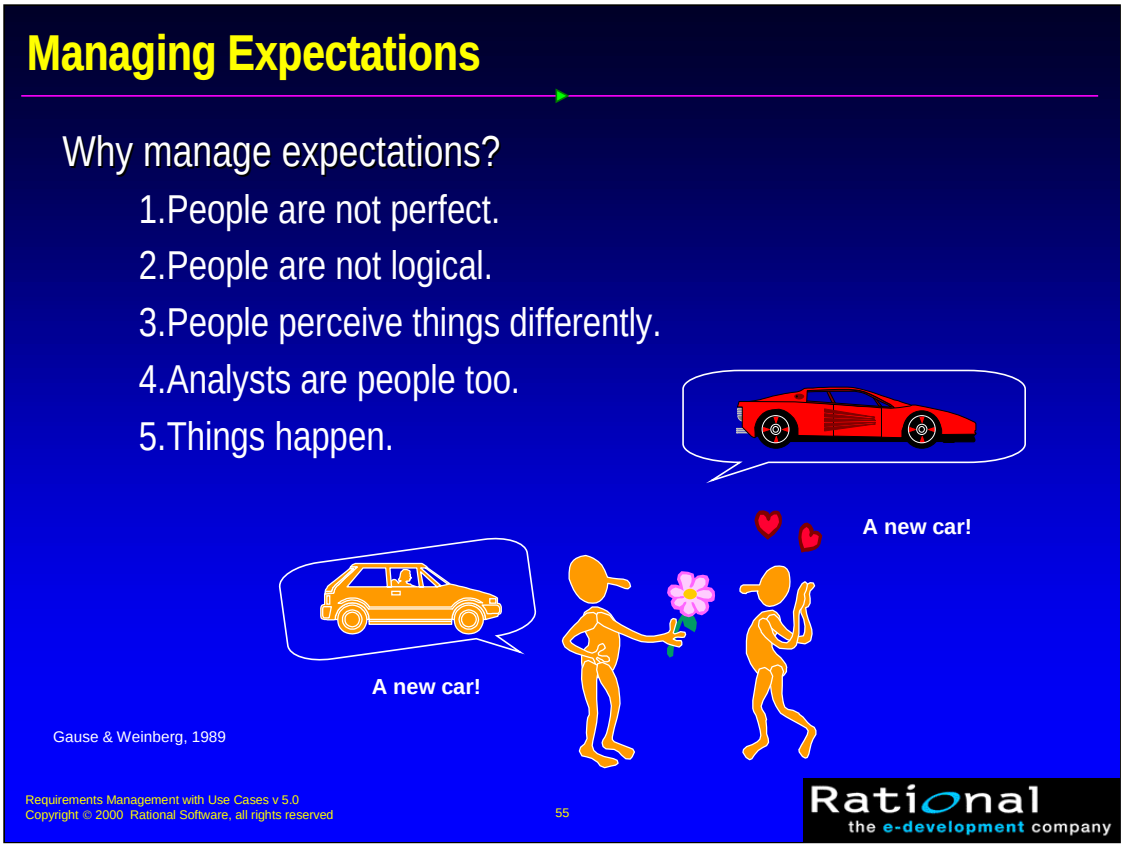
Rational
the e-development company

A key here is to keep the requirements visible (write down what is agreed on) and accessible to all team members.



As requests come in during our lifecycle, we need to intercept them and make sure that they pass through a single approval channel.

This may be one person (like a project manager), or perhaps a group of representatives from each of the relevant groups that would be affected (e.g., a CCB, Change -- or Configuration -- Control Board).



One of the keys to having a happy customer at delivery time is to manage their expectations of what they are to receive.

Improve Your Negotiation Skills

- Key to any successful, multi-party program
- A normal, professional activity
- Tips
  - Start high, but not unreasonable
  - Separate the people from the problem
  - Focus on interests, not positions
  - Understand your BATNA (Best Alternative To a Negotiated Arrangement)
  - Invent options for mutual gain
  - Use diplomacy

Improve your skills at soonest opportunity!

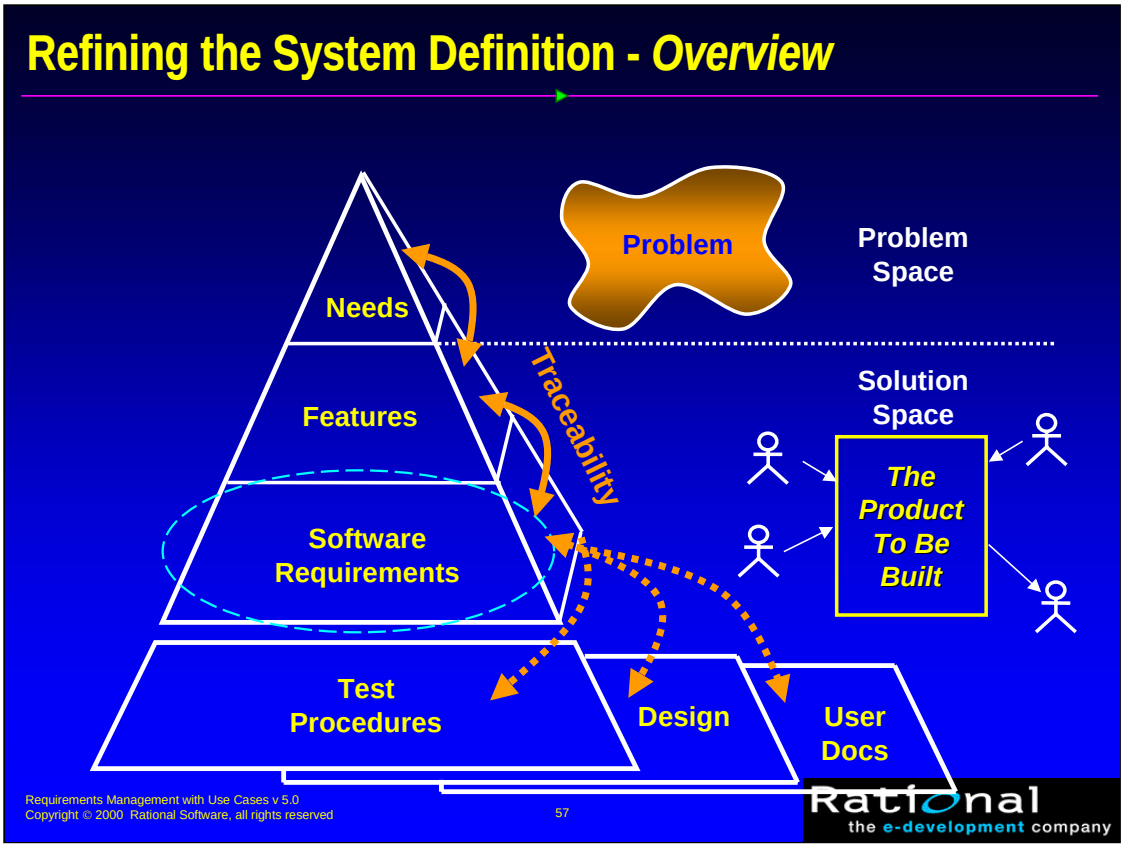
Fisher, Ury, Getting to Yes, 1991

Rational the e-development company

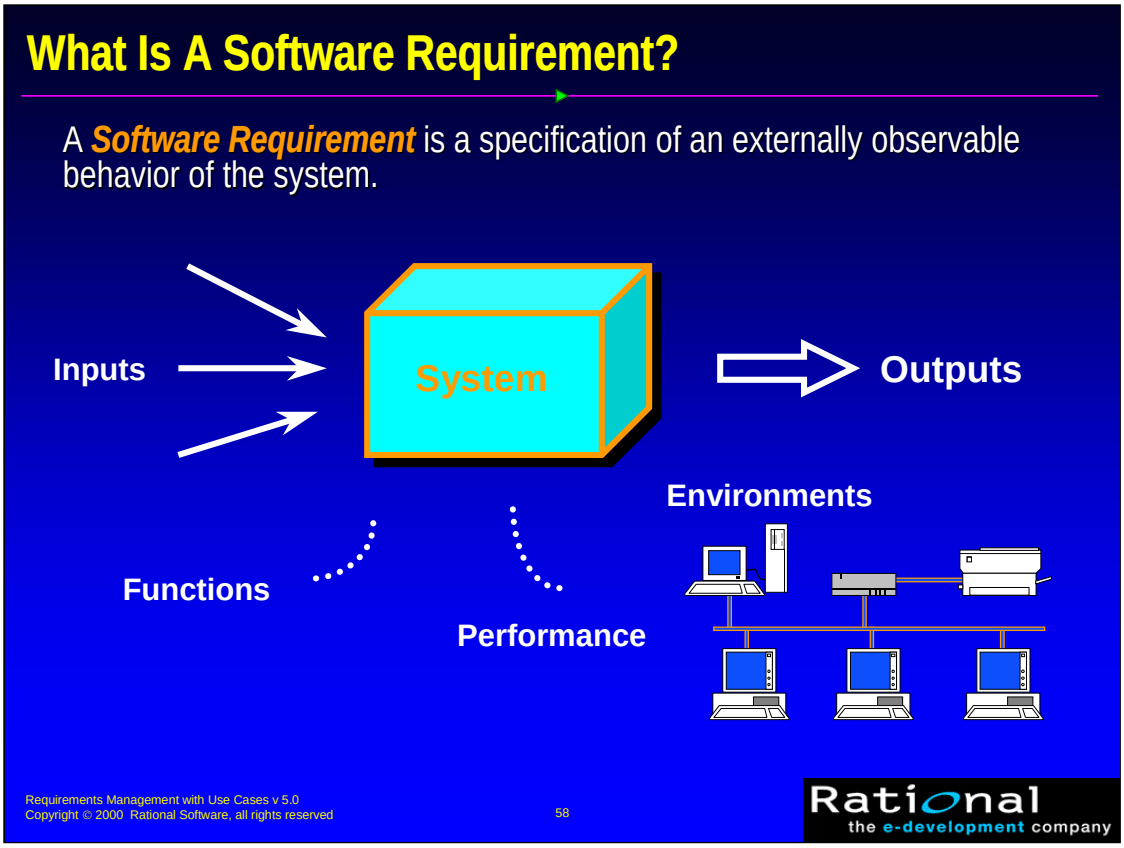
This is from Fisher & Ury, Getting to Yes. Fiser and Ury worked with the Harvard Negotiation Project, who studied high-level (mostly diplomatic) negotiations and evaluated what made them successful. One example in the book describes the negotiations for agreement between the Israelis and the different Arab factions.

The concept of BATNA (Best Alternative To a Negotiated Agreement) encourages you to also look at the consequences of “not” getting an agreement. How important is that to you? What if the customer cancels the project? This gives you a “bottom line” from which to work.

The key is to focus on the interests of all involved and attempt to come up with creative options that would satisfy both sides.



Now we will move into the heart of the requirements -- the software requirements.



The SRS will provide an almost “black-box” definition of the system, defining wherever possible only those externally observable “what”’s of the system, rather than the “how” it will be accomplished.

Of course, there will have to be enough of the “how”’s specified to build the right system, but these should be identified as design constraints.

Use-Case Report - *Template*

<Use-Case Name>

1. Brief Description
The use case's role

2. Flow of Events
The use case's behavior

3. Relationships
Actor communicates-associations, uses-and extends-relationships in which the use case participates

4. Special Requirements
Collection of non-functional requirements pertaining to this use case

Enclosures
Diagrams
Pre-Conditions (optional)
Post-Conditions (optional)
Pictures of the (graphical) user interface

The Use-Case Report contains information regarding an individual use case within the use-case model.

Basic flow of events
Alternative flows of events

59

Rational
the e-development company

The purpose of this document is to describe the flow of events of a use case, specifically in a way that the customer understands:

Brief Description should describe the purpose of the use case in a few sentences.

Flow of Events should represent what the use case does in the system, how and when it begins and ends, when the use case interacts with an actor, and what information is exchanged. We will discuss this in more detail in the following slides.

Associations list all the associations that this use case has to other use cases, with brief descriptions where applicable.

Special Requirements is a collection of all the requirements that are not covered by the flow of events that could influence the design. Requirements on characteristics, e.g., performance (response times,...).

A **view** could be an enclosed diagram, showing the use case, its actors, communication associations and related use cases.

Illustrate the flow events with pictures of the user interface, e.g., sketches or printouts from a prototype.

Who Reads the Flow of Events?

- **Customers:** approves the result, what the system should do
- **Users:** help understand what the system should do
- **Use-Case Specifier:** to have a good documentation of system behavior
- **Reviewers:** examine the flow of event
- **Designers:** to find design classes
- **System Tester:** as a base for test cases
- **Project Manager:** follow up the project
- **Technical Writer:** when writing the user's guide for the end users

60
Rational the e-development company

Before we go into the details on how to write the document, we must know who will read the Use-Case Description. You may find your readers on the list, or maybe you will have others. The readers vary from system to system (project to project).

It is important to keep the actual readers in mind throughout the work in the document, in order to focus on the right:

- ☑ contents
- ☑ level
- ☑ way of describing, e.g., words to use

Flow of Events - *Guidelines*

1. Don't describe what happens outside the system.
2. Describe what ***data is exchanged*** between the actor and the use case.
3. Do ***not*** describe the ***details*** of the ***user interface***, unless it is an important requirement.
4. Describe the ***flow of events***, not only the functionality. To enforce this, start every action with "When the [actor] ... ".
5. Describe ***only*** the events that ***belong to the use case*** -- not what happens in other use cases or outside of the system.
6. ***Avoid vague terminology*** such as "for example", "etc. " and "information".
7. Detail the flow—***all "whats"*** should be answered.

61
Rational the e-development company

1. You should not describe what happens outside the system since that cannot be controlled.
2. It is important to know what data has been exchanged.
3. If you include too detailed of a description of the user interface, you will have to change the description each time you change the way of using the user interface. Try to keep this part in a prototype where it will be easy to change.
4. Understand what really happens in the system, so we don't miss anything. Describe the flow of events.
5. Describe only what happens in this use case so that if anything changes, you will only have to look inside this one use case to make the change.
6. Unclear expressions like these may hide important information. Do not wait to understand the system, bring every thing out in the light.
7. Be sure that your description is complete, i.e., that you have answered all the "whats".

Subflows - Structuring the Flow of Events

- A use case's flow of events may be divided into subflows
  - ***Optional*** sequences
  - ***Parallel*** sequences
  - ***Large segments*** of a given flow of events
  - ***Exceptional*** flows
    - To help the main flow to stand out more clearly.
  - Sequences that can be ***executed at several intervals*** in the same flow.
  - To ***clarify traceability*** to other project elements.

Once we start filling in the details, it may help in the understanding if we can provide some structure to the text in the description.

We start this by looking for subflows. These may be signified by using different header levels for clarification purposes and dividing the flow of events into different sections.

Use-Case Subflows as Separate Sections

- **Basic** flow of events
  - What normally happens if nothing goes wrong
- **Alternative** flows of events
  - Use to specify
    - **Variants** to the basic flow of events
    - **Optional** flows of events
    - **Exceptions**, that is, error cases
  - Occupies a large segment of the flow of events
  - Can be executed at several intervals
- The flow of events of a use case can be ***5-15 pages*** long, depending on the complexity of the use case.

63
Rational the e-development company

You can have two main sections: Basic Flow of Events and Alternative Flow of Events. These should be divided into subsections that are given titles. The goal is to make the document easy to read!

Flow of Events - *Guidelines for Structure*

- Describe how the use case ***starts*** and ***ends***
- Start by defining the ***basic flow*** of events
- Structure the basic flow of events into ***sub-flows***, give each sub-flow a ***title***
- Describe all ***alternative*** subflows
- Describe the ***order*** of the subflows
- Only describe the order of the subflows as ***fixed***, if it is. In other case point out that it is ***unfixed***

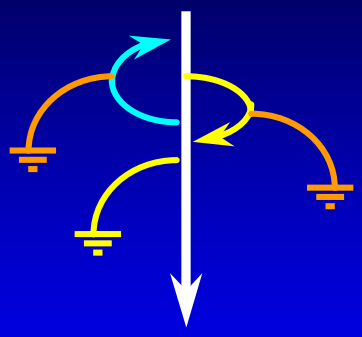
This type of written text, structured into consecutive subsections, by its very nature, will imply to the reader that there is a sequence between the subflows.

To avoid misunderstandings, you should always point out whether the order of the subflows is fixed or not. Considerations of this kind are often related to:

- ☑ Business rules. For example, the user has to be authorized before the system can make certain data available.
- ☑ User-interface design. For example, the system should not enforce a certain sequence of behavior that may be intuitive to some but not to other users.

Structuring of Alternative Flows

- Define ***where*** the alternative flow ***starts***
- Define the ***condition*** for this alternative
- Describe the ***behavior*** in the alternative flow of events
- Describe ***where*** the basic flow is ***resumed***
- Use one ***section*** for each alternative flow



65
Rational
the e-development company

There are always different ways to structure the text, but we will not tell you that you always should do it one particular way or another. Here are some suggestions as to when you should think of using sections:

- ☑ You can have two main sections: Normal Flow of Events and Alternative Flow of Events.
- ☑ Under the Alternative Flow of Events you can create sections and give them names, e.g., Normal Variants, Exceptions, etc.
- ☑ The main goal is, as usual, is to make the document easy to read.
- ☑ Describe the alternative flows in their own section, not within the basic flow (although some have found it useful to put pointers where these may be inserted for clarification purposes).

Specific Alternative Flows

- Specific alternatives
  - Occur ***at a specific step*** in another flow.
  - Example:

A1: Bottle stuck
The customer inserts a bottle that falls over, is too big, or jams with another bottle. The sensors around the gate and the measuring gate detect this problem. The conveyer belt is stopped and an alarm is issued to call for the operator. The operator fixes the problem and the machine continues.

Here is one way to write an Alternate Flow that only occurs at a specific place.

General Alternative Flows

- An alternative flow may pick up the sequence of actions ***anywhere***:
  - Example:

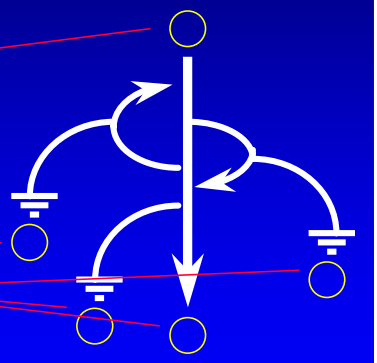
A2: Front panel is removed

If at any point, somebody (probably the operator) removes the front panel to the machine, then the can compression is deactivated. It will not be possible to start the can compression with the front panel off. The removal will also activate an alarm to the operator. When the front panel is closed again will the machine resume the operation.

Here is an example of an Alternative Flow that may occur anywhere!

Use of Pre- and Post-Conditions

- Using Pre- and Post-Conditions in the flow of events:
  - The pre- and/or post-conditions should be ***observable*** to the user
  - Use ***only*** if needed for ***clarification***
- A pre-condition
  - A ***constraint*** on when the use case can start
  - ***NOT*** the event that starts the use case
- A post-condition
  - should ***apply regardless*** of alternative flows
  - may have ***different variants***



If you decide to use this, first describe the pre-conditions in the flow of events and then lastly the post-conditions in the flow of events.

These states should be observable to the user, e.g. “the user has logged on to the system” or “the user has opened the document”.

A pre-condition for a use case is not a pre-condition for only one sub-flow, though pre-conditions may be defined for sub-flows as well. Note that it is not the event starting the use case.

A post-condition for a use case should be true regardless of which alternative flows were executed; it should not be true for the main flow alone. If something should fail, you should use a post-condition like, “The transaction is completed or, if something failed, the transaction is not performed” rather than “The transaction is completed”.

Example of a Pre-Condition

Cash Withdrawal

Pre-condition

- The customer has a personally-issued card that fits in the card reader, has been issued a PIN number, and is registered with the banking system.

Use only if needed for clarification!

69
Rational the e-development company

The states described by pre- or post-conditions should be stated in order that the user can observe. "The user has logged on to the system" or "The user has opened the document" are examples of observable states.

A pre-condition is a constraint on when a use case can start. It is not the event that starts the use case.

A pre-condition for a use case is not a pre-condition for only one subflow. You can also define pre- and post-conditions at the subflow level.

Example of Post-Condition

Cash Withdrawal

Post-condition

- At the end of the use case, all account and transaction logs are balanced, communication with the banking system is reinitialized and the customer has been returned his card or informed of where it will be sent.

- **Use to clarify what the final state of the system should be when the use case ends.**
- **This can help make sure the use case ends properly when alternate flows are specified.**

A post-condition for a use case should be true regardless of which alternative flows were executed; it should not be true only for the main flow. If something should fail, you would cover that in the post-condition by saying, "The action is completed or, if something failed, the action is not performed", rather than just, "The action is completed".

Note: When you use post-conditions together with extend-relationships, you should take care that the extending use case will not introduce a subflow that violates the post-condition in the base use case. This will be discussed more in depth in the next module.

Post-conditions can be a powerful tool for describing use cases. You must first define what the use case is supposed to achieve -- the post-condition. You can then describe the different ways to reach this condition (the flow of events needed).

What about Non-Functional Requirements?

- The "URPS" of FURPS
  - Usability
  - Reliability
  - Performance
  - Supportability
- Compliance with Legal and Regulatory requirements
  - FCC
  - FDA
  - DOD
  - ISO
- Design Constraints
  - Operating systems
  - Environments
  - Compatibility
  - Application standards

What are some others?

Where should they be specified?

71
Rational the e-development company

What other types of non-functional requirements need to be specified?

The "URPS" of FURPS

Functionality	Feature Set Capabilities	Generality Security
Usability	Human Factors Aesthetics	Consistency Documentation
Reliability	Frequency/Severity of Failure Recoverability	Predictability Accuracy MTBF
Performance	Speed Efficiency Resource Usage	Throughput Response Time
Supportability	Testability Extensibility Adaptability Maintainability Compatibility	Configurability Serviceability Installability Localizability Robustness

Which of these might be captured in the use-case model?

With which ones might this not be possible or practical?

What should you do with them?

Grady, 1992

72

Rational
the e-development company

This is a reminder of the FURPS discussion which appeared at the beginning of the course. Here is a checklist of items to make sure we are building a quality system.

The FURPS acronym represent a portion of the types of requirements for which we should be searching; it reminds us to get non-functional (URPS) as well as functional (behavioral) requirements.

What are some other examples of the types of non-functional requirements we should be on the lookout for?

Maintainability

Robustness

Testability

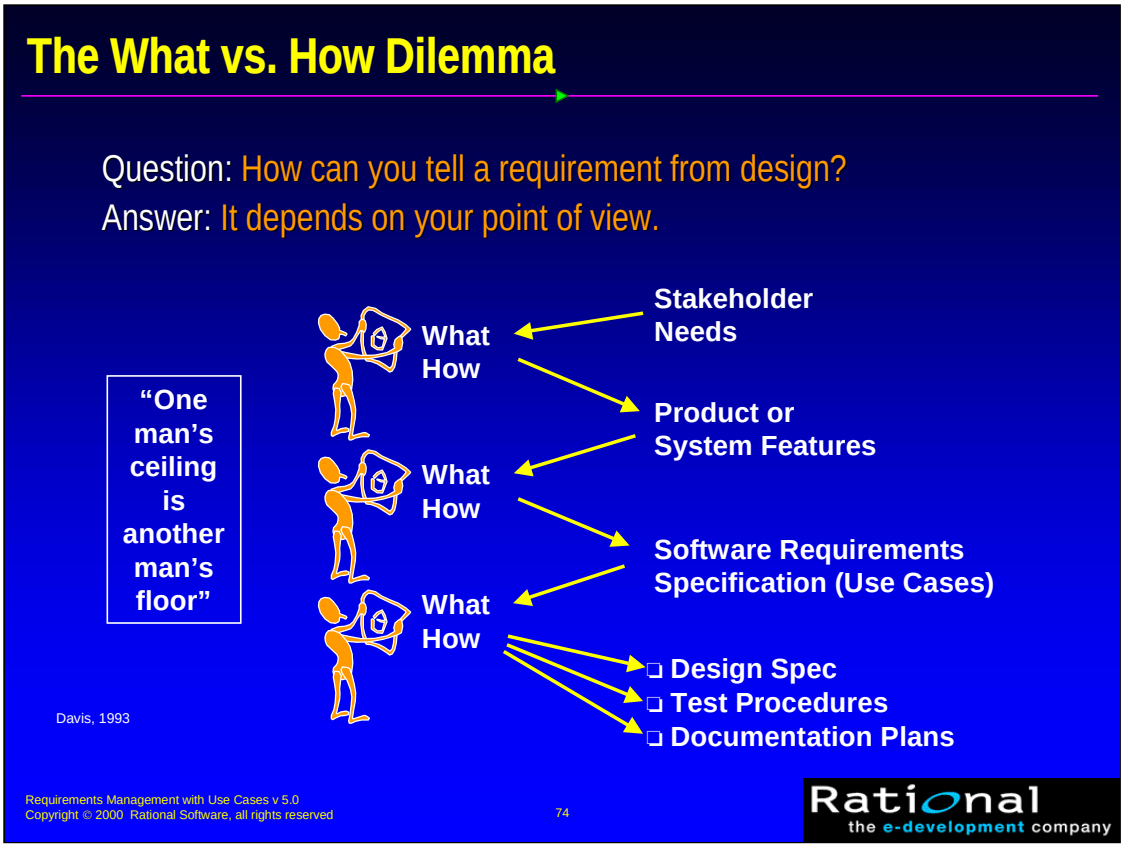
…

Where would they be specified? (see previous slide)

What about Design Constraints?

- A ***requirement*** should allow more than one design option.
  - A design is a choice among options.
- A requirement that leaves no options is a ***design constraint***.
  - Distinguish it from a requirement
  - Place in a **special section** of your software requirements
  - Identify the ***source*** of each
  - Document the ***rationale*** for each

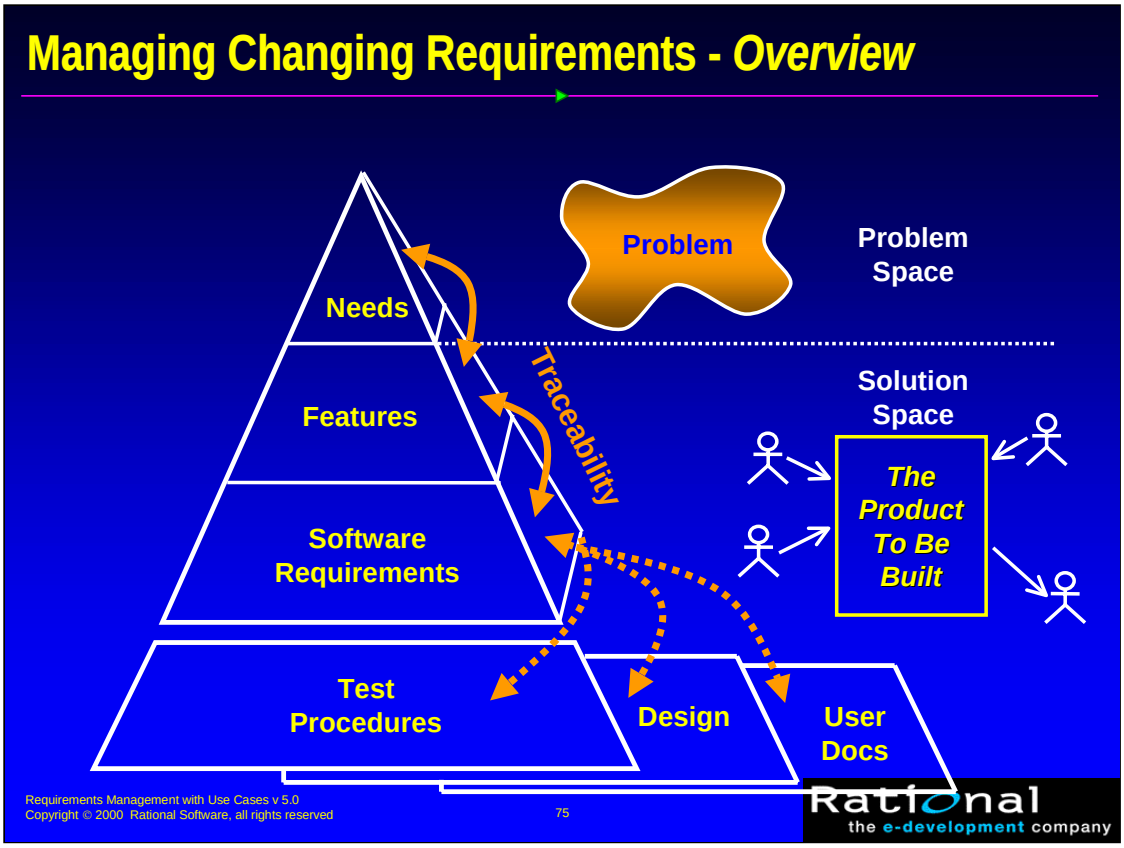
Make sure that you can understand and make clear which requirements are constraints on the design.



A common question that frequently comes up when writing requirements is, "How can I write a requirement that tells the audience 'what' do to without specifying 'how' to do it?"

Unfortunately, there is no one right answer. It should depend on the intended point of view of the particular document reader: For example:

- ☑ When writing the Vision Document, the "what" are the user needs, and the "how" are the features
- ☑ For the SRS (or use cases) the "what" are the features, and the "how" are the software requirements -- use case specification or supplementary requirements.
- ☑ For the others, … ?



In this module we will discuss various ways to manage changing requirements.

By using an iterative process, this will allow us to refine the model in smaller increments at each iteration and to help us maintain control of the changes throughout the product lifecycle.

Why Do Requirements Change?

- We failed to ask the right people the right questions at the right time
- The problem being solved changed
- The users changed their minds or their perceptions
- The external environment changed
- We failed to create a process to help manage change

76
Rational the e-development company

Has anyone ever worked on a project where there were “no” changes in the requirements from the initial agreement through the end of the project?

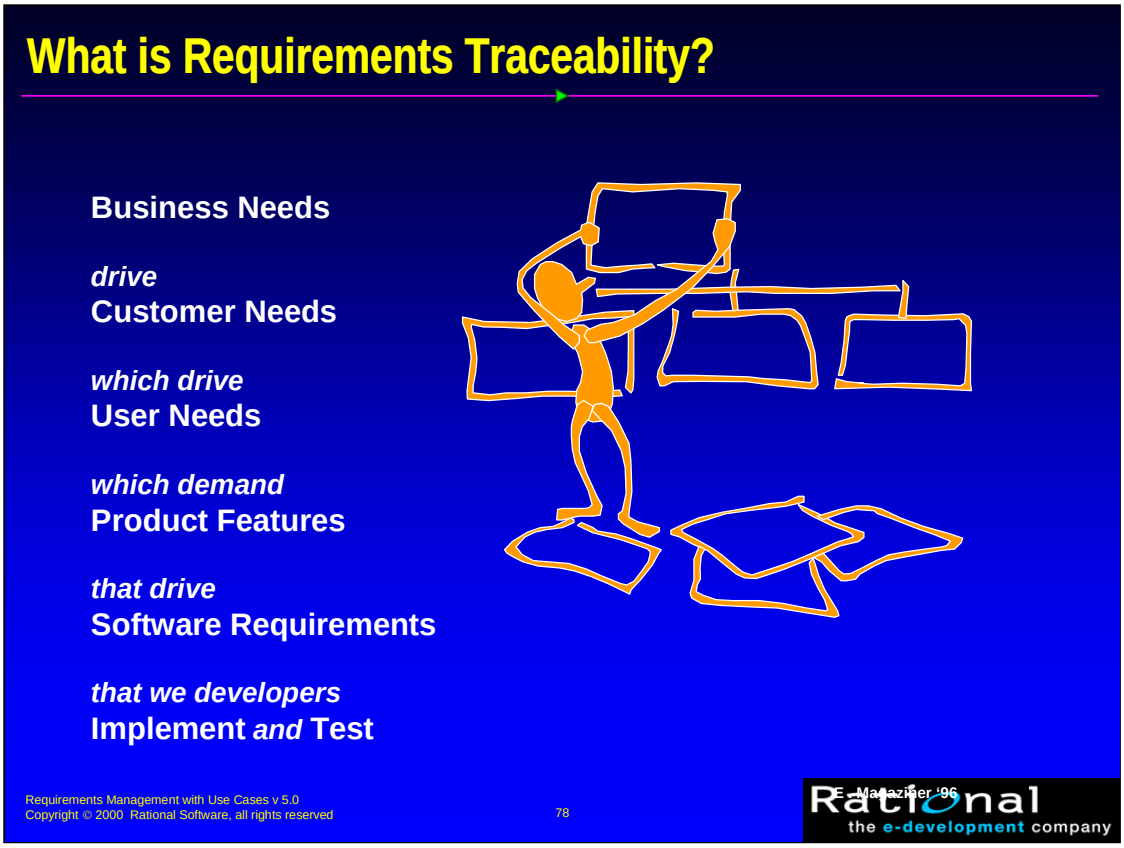
What About Metrics for Requirements Management?

- What types of questions might be asked of the repository?
  - How many requirements do we have?
  - What percentage are in the baseline?
  - How many critical requirements haven't been implemented?
  - How many changes since the last customer review?
    - Who authorized the changes?
    - What's the impact on test?
  - What's the estimated cost of the proposed changes?
  - What resources would be needed to put in this new feature?

See "Metrics for Requirements Management" in the Handouts

77

Rational
the e-development company

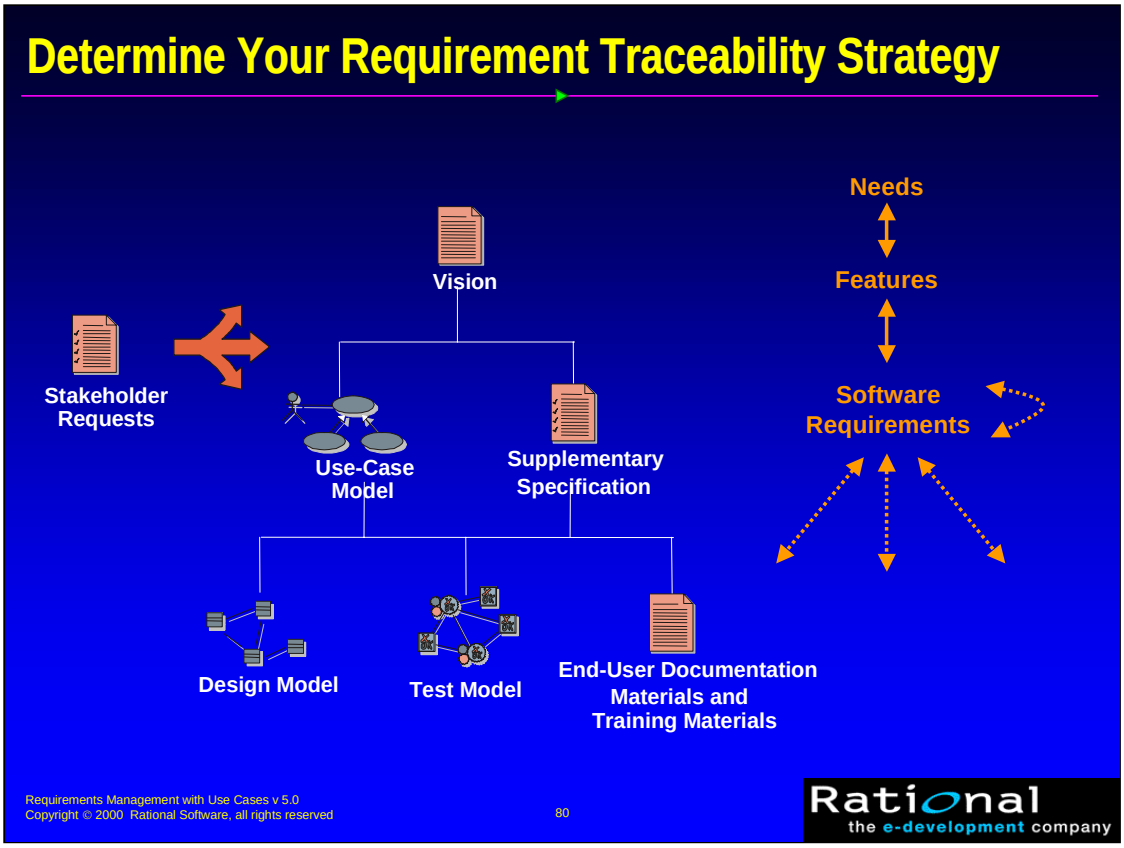


Here are some of the different levels of requirements we may wish to trace.

Why Use Requirements Traceability?

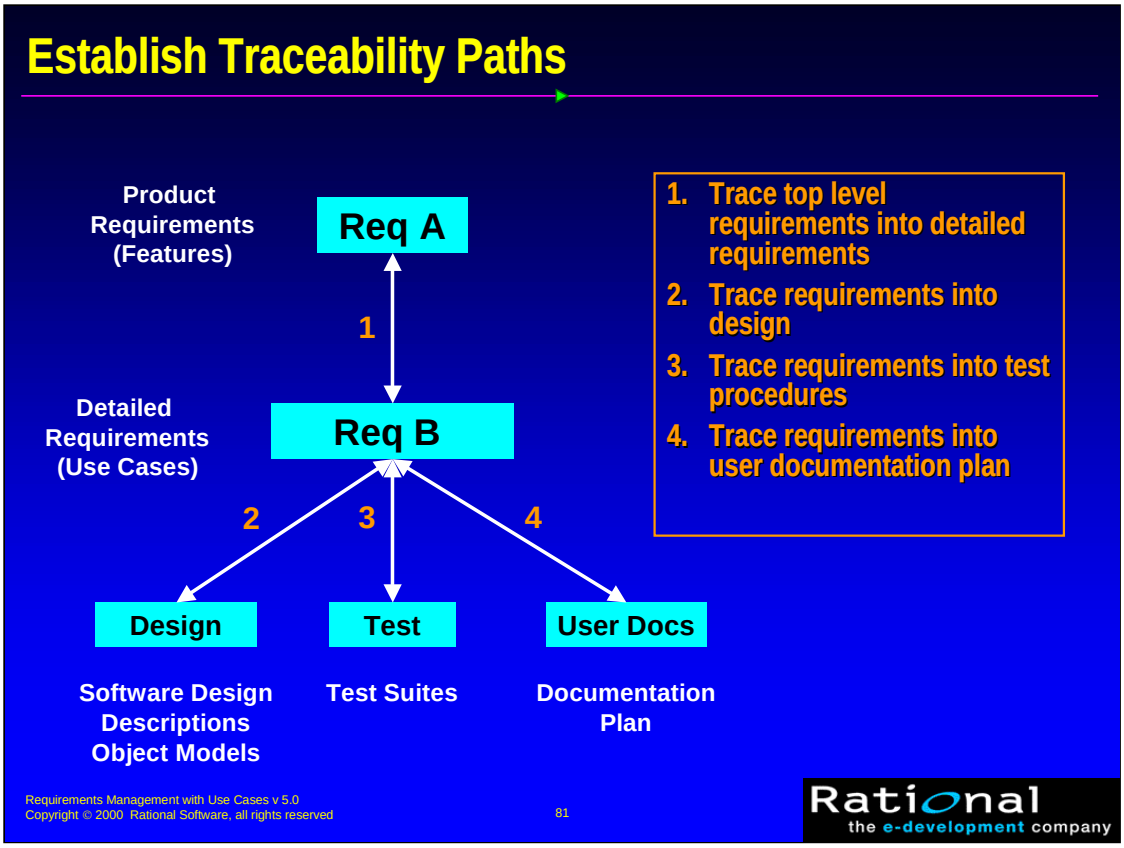
- The purpose is to:
  - Verify that all requirements of the system are fulfilled by the implementation.
  - Verify that the application does only what it was intended to do
  - Help manage change
- A proven technique for assuring quality
  - Practiced by high-reliability system developers
  - Mandated by standards (e.g.: DOD, FDA)
  - Recommended by IEEE and CMM
- A proven technique for understanding the impact of changes

One of the keys to effectively change the management of requirements is to maintain traceability of the requirements to other project elements.



The first step is to establish your requirements structure and the relationship of different types of requirements to each other.

See white paper on “Traceability Strategies for Managing Requirements with Use Cases” (in the Student Handout book) for a more complete discussion of different options.



Based on this structure, we then need to set up traceability links between all associated requirements or other project elements.

Link Feature To Use Case

Vision_Document
FEAT10:The recycling machine will allow the addition of new bottle types.

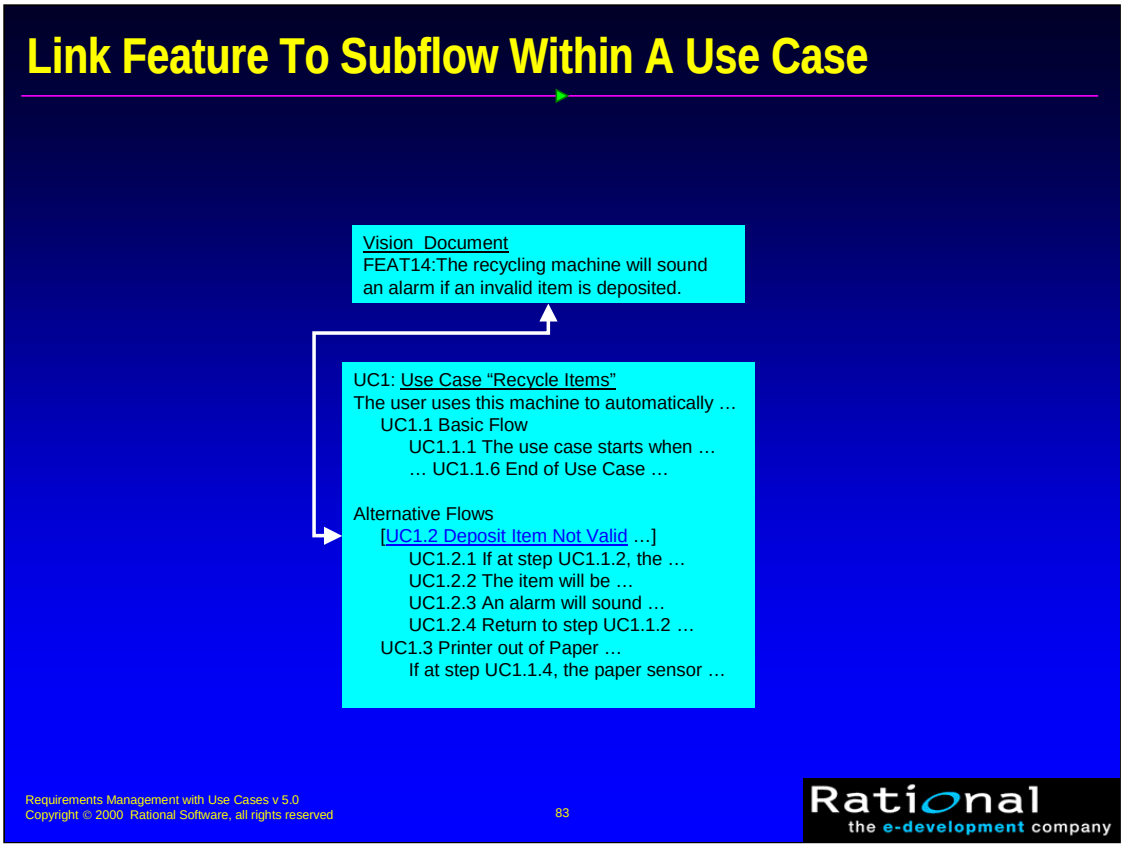
[UC4: Use Case "Add New Bottle Type"]
New kinds of bottles can be added to the machine by starting it in 'learning mode' and inserting 5 samples …

Verifies that all lower level (derived requirements) are consistent with stakeholder needs.

82
Rational the e-development company

Here are some types of links that may be appropriate in our requirements.

We may trace a feature directly to a use case.



It may also be linked to a subflow of a use case (such as an alternative flow).

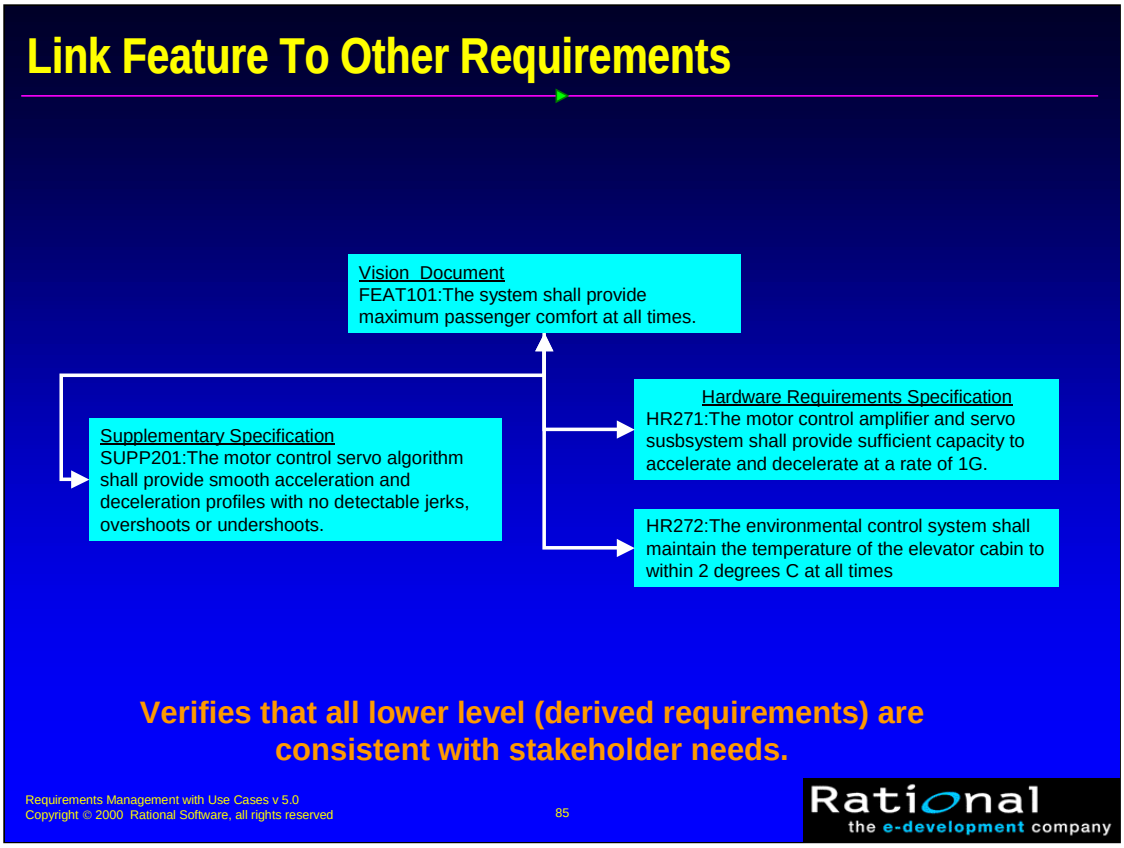
Link Feature To Section Of A Use Case

Vision Document
FEAT24:The recycling machine shall recognize deposit items with 95% reliability.

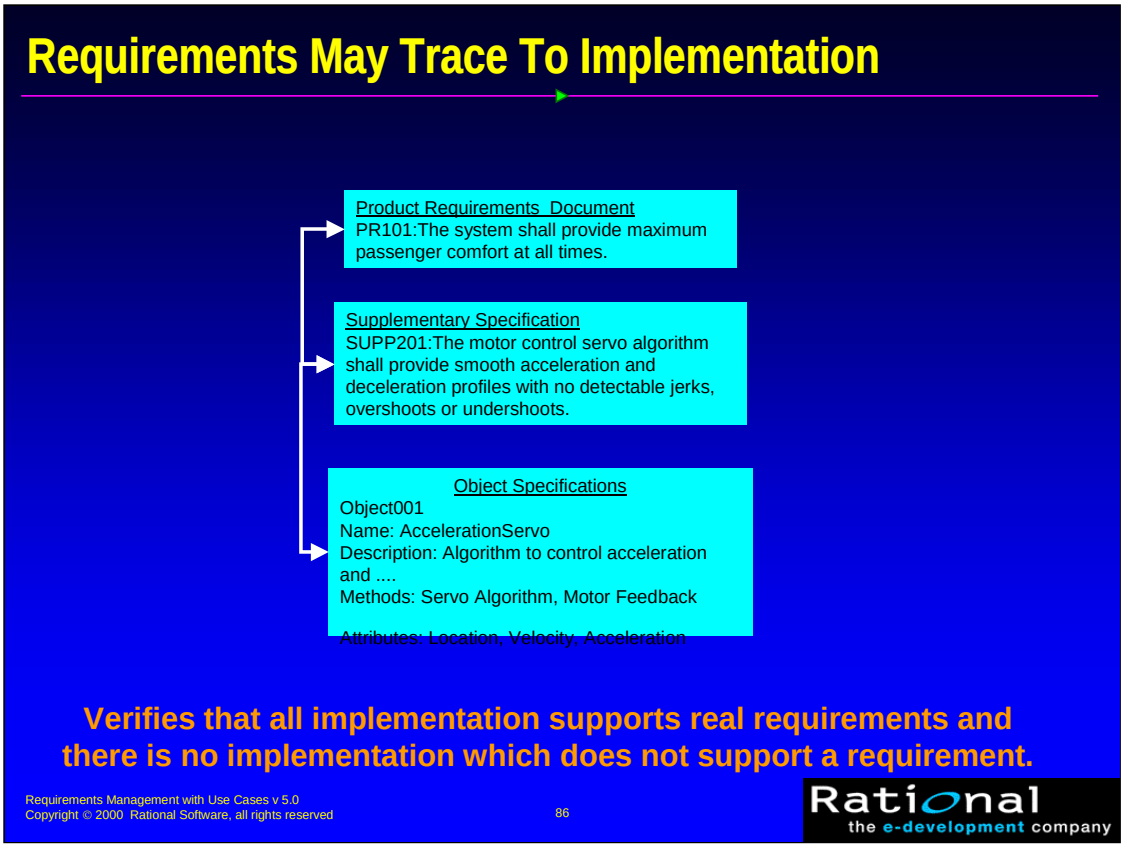
UC1: Use Case "Recycle Items"
The user uses this machine to automatically …
UC1.1 Basic Flow
...
Alternative Flows
UC1.2 Deposit Item Not Valid …
...
Special Requirements
[UC1.17 Recycle items must be recognized with 95% reliability]
...
Use-Case Diagram

...

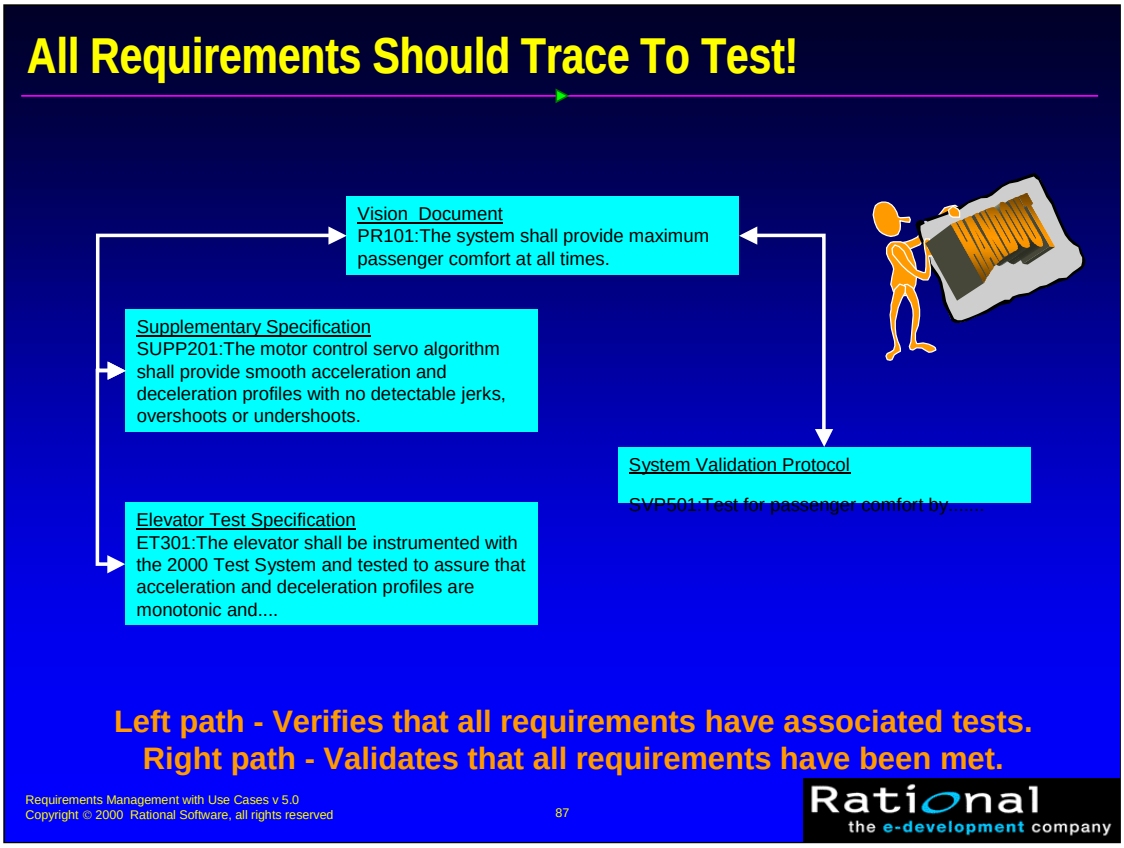
It could also be traced to some other section of the use case description, e.g., the special requirements for non-functional requirements that apply to a particular use case.



In our model, we may also trace to supplementary requirements (non-functional that apply to the whole system), or perhaps even to hardware requirements, should that be the way the requirements need to be met.



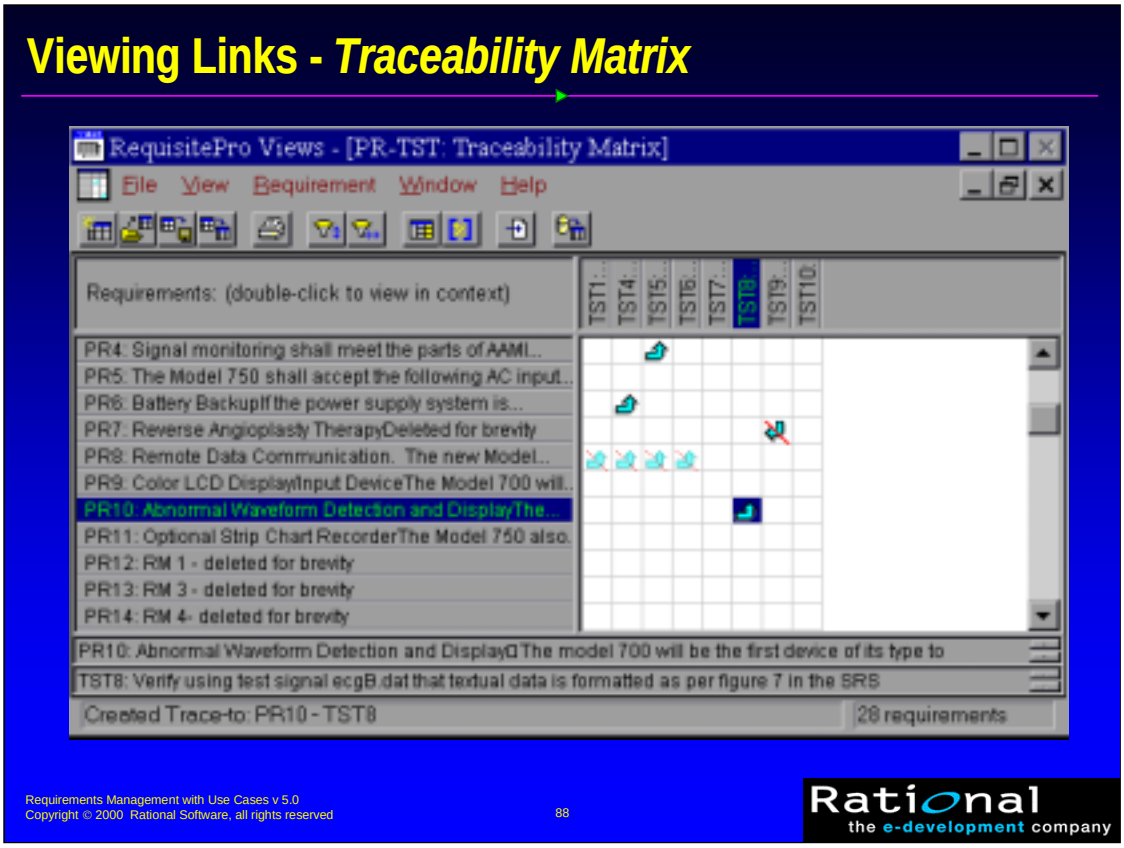
From these links to the SRS level, we can then trace down into the object model. This is only feasible if you have tools to assist (e.g., Rose/ReqPro).



Most importantly, make sure it can be verified that it actually met.

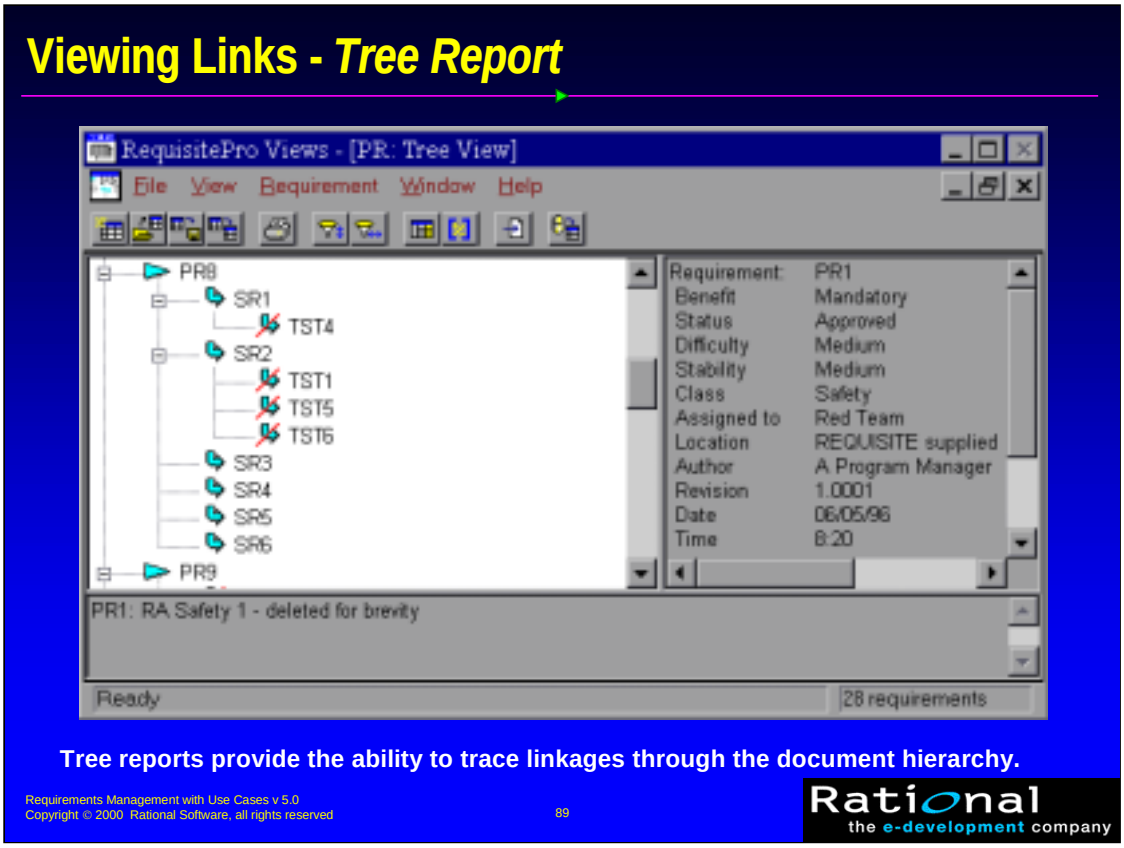
Look at the (ATM) Sample Test Case (Handout #6) matrix that outlines the different scenarios of the Withdraw Cash use case so that the different variants can be tested.

There is also a sample Test Requirement Specification in the “Sample Document Templates” section of your handouts.



Here are some ways you might view the traceability links using a requirements management tool, such as Rational's RequisitePro.

A traceability matrix presents a 2-dimensional view of the traces from one requirement type to another (or to the same) type.



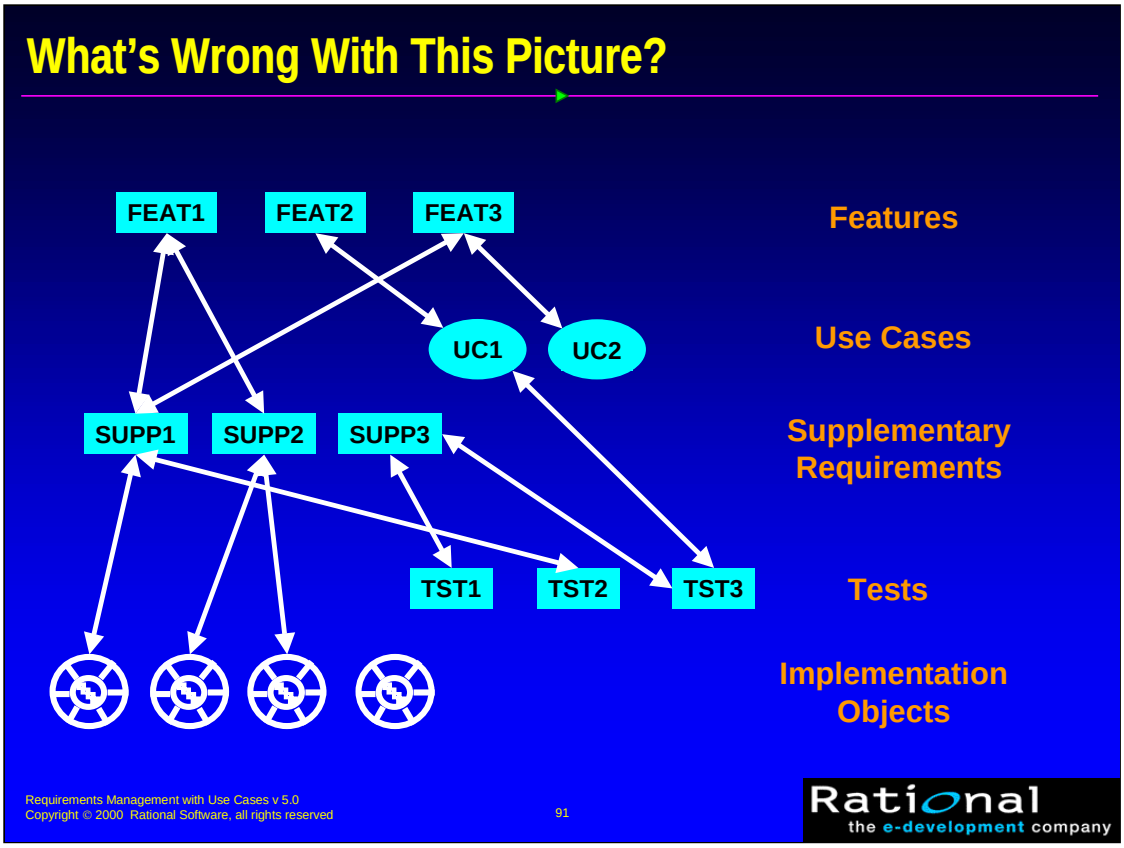
The Tree view allows you to expand a particular requirement so that you can see its full depth of traceability.

In this case, we are also displaying the selected requirement's associated attributes that are also contained in the repository for each of the requirements (in the right-hand panel).

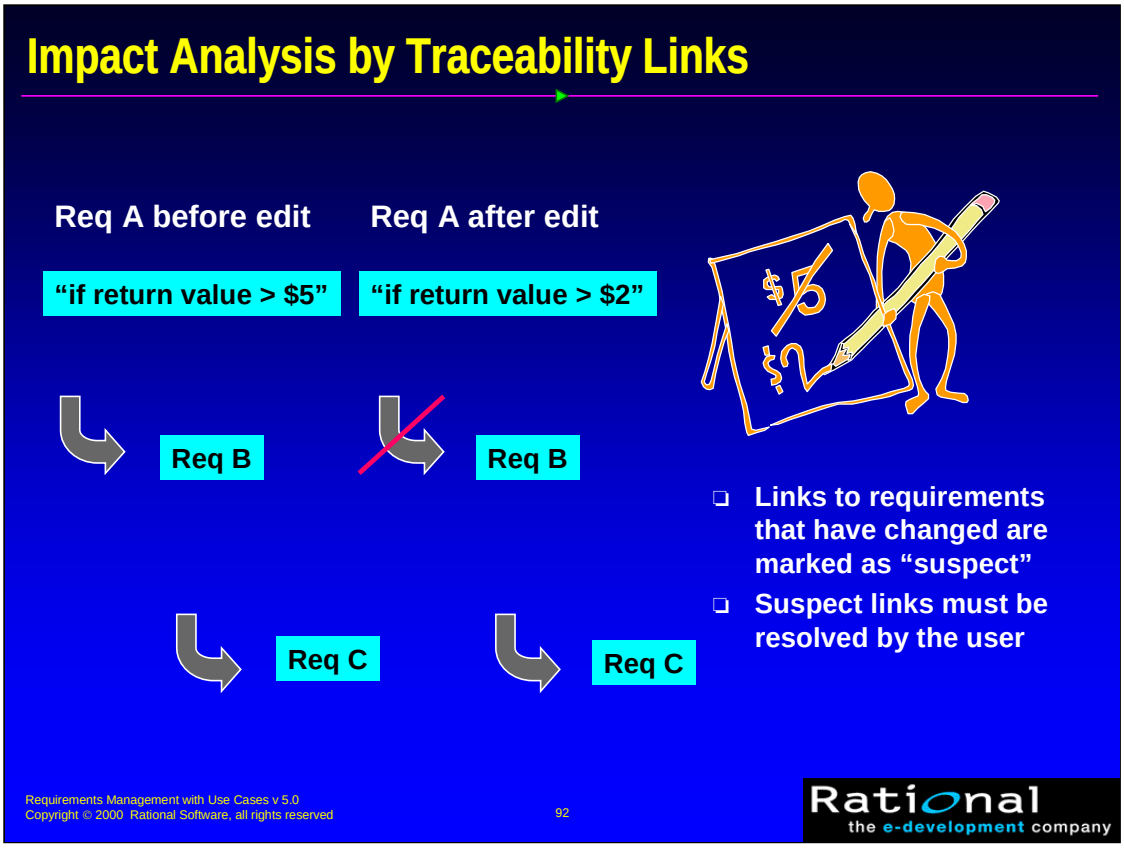
Sample Queries Using Requirement Links

- Show me user needs which are not linked to product features
- Show me the status of tests on all use cases in iteration #3?
- Show me all supplementary requirements linked to tests whose status is untested
- Show me the results of all tests which failed, in order of criticality
- Show me the features scheduled for this release, which user needs they satisfy, and their status

What types of questions might you ask that could be asked by knowing about the links between the requirements?



All top-level requirements should trace to a software requirement (either within a use case, a supplementary spec or a formal SRS) and eventually to test (and also to implementation if that level of traceability is to be maintained).



RequisitePro provides what are called "suspect links", which can notify that an associated requirement has changed.

All directly related requirements should be reviewed to assess whether they are affected.

Why is the link from Req B to Req C not marked as suspect?

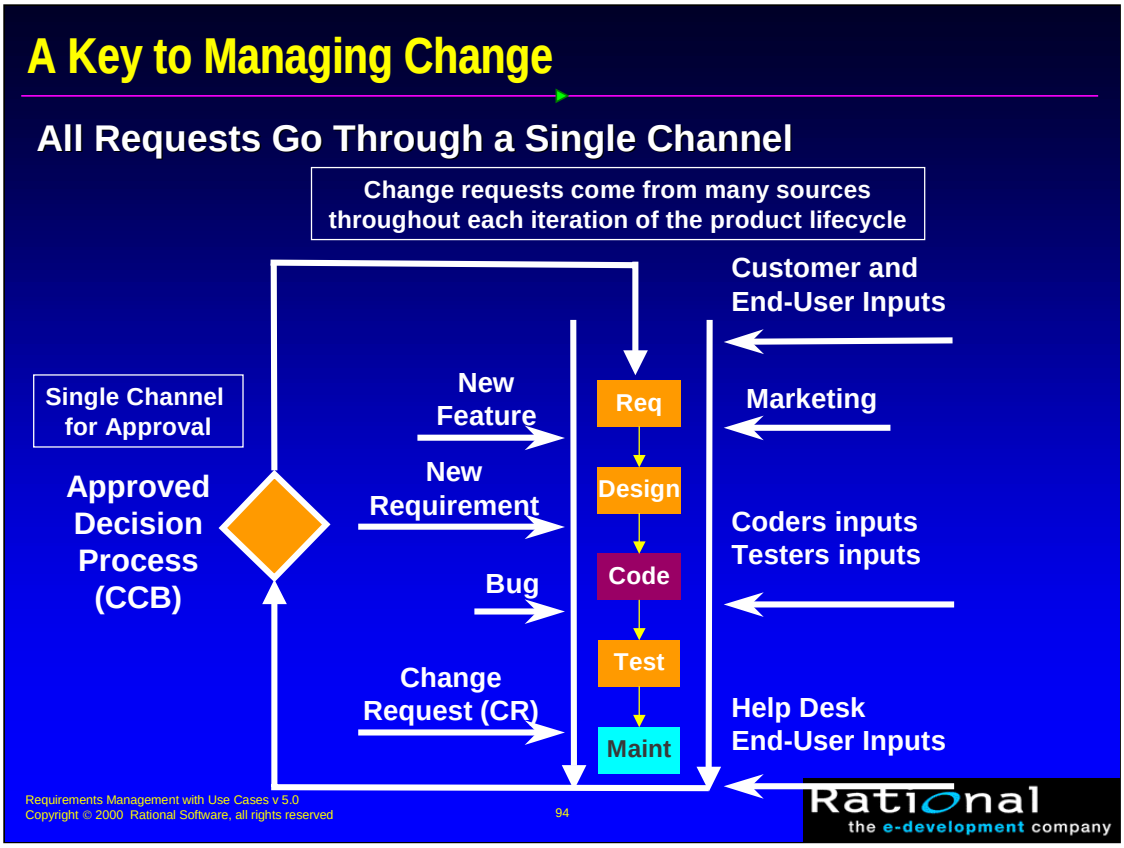
The only way to resolve these are manually (by actually looking at the changes and the affected requirements).

You can probably make a "lot" of money if you could figure out a way to do this automatically (joke)!

Controlling Requests for Change

- Require all changes go through a ***single channel***
- Have a ***documented process*** for handling change requests
- Contents of a ***change request***:
  - What artifact or artifacts it concerns
  - Problem description
  - Suggested solution
  - Priority
- Should be submitted to a ***Change Control Board (CCB)*** or other review authority

What should be contained in a Change Control Process?



Here is the same process that we showed in our scope management unit (control the requirements coming into the project).

We need to continue to follow the same change control process (having all requests go through a single channel) to control changes throughout the product lifecycle.