AARHUS
UNIVERSITET

4. FEBRUARY 2014

TISYE1 – Lecture 2

Concept Development: Needs and Requirements

STEFAN HALLERSTEDE (SHA@ENG.AU.DK)
ASSOCIATE PROFESSOR

Plan for the Lecture

› Before building a system we should ask some questions

› They are – in the given order:
› **Why** (do we need the system)?
› **What** (is required of the system)?
› **How** (do we build the system)?

› Why should we not begin with the "**how**"?

› What are the associated risks?

When to ask “why” and “what”

› Concept Development:

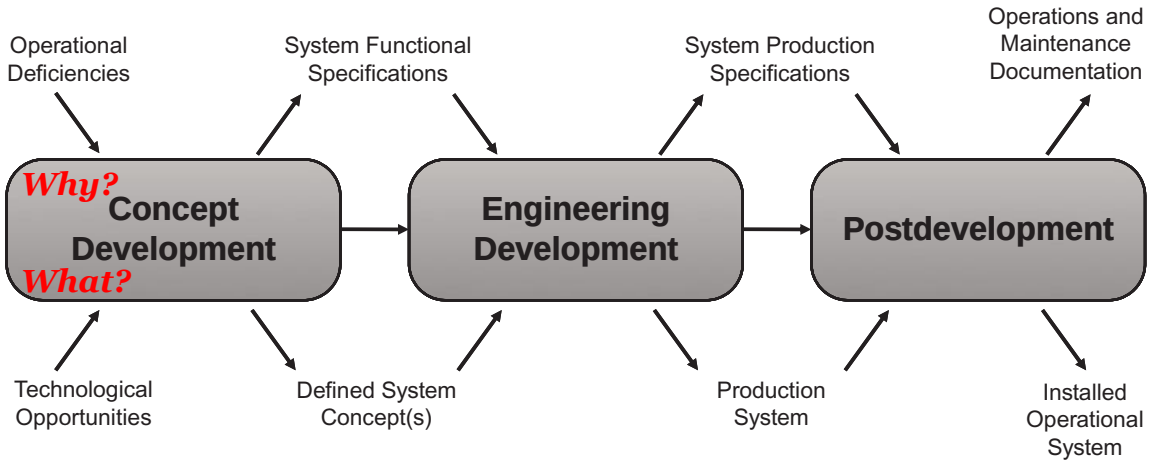


Figure 4.3. Principal stages in a system life cycle.

› Two reasons for starting a development:

› Needs-driven: a new need is identified

› Technology-driven: new possibilities due to technological progress

Needs Analysis

› Concept Development:

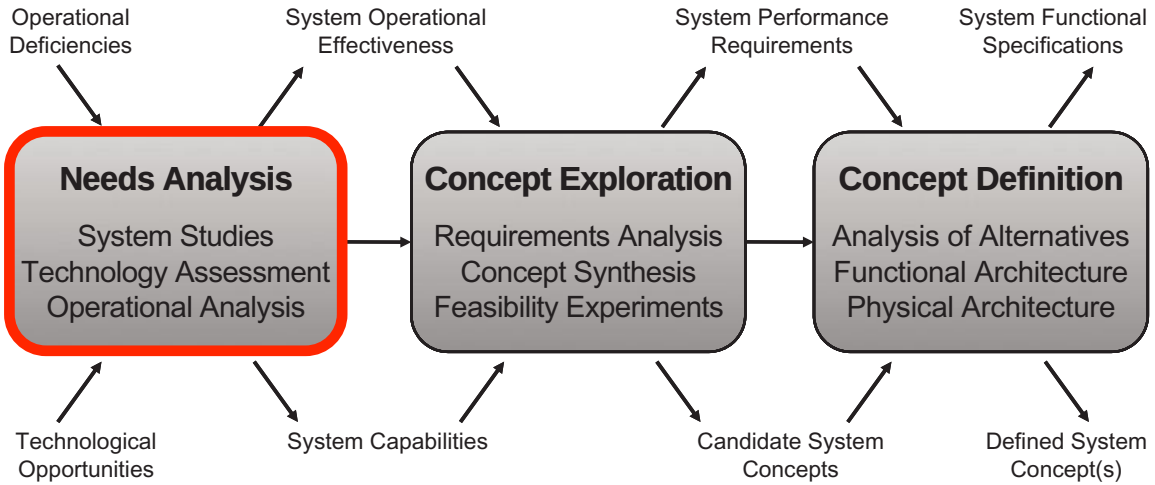


Figure 4.4. Concept development phases of a system life cycle.

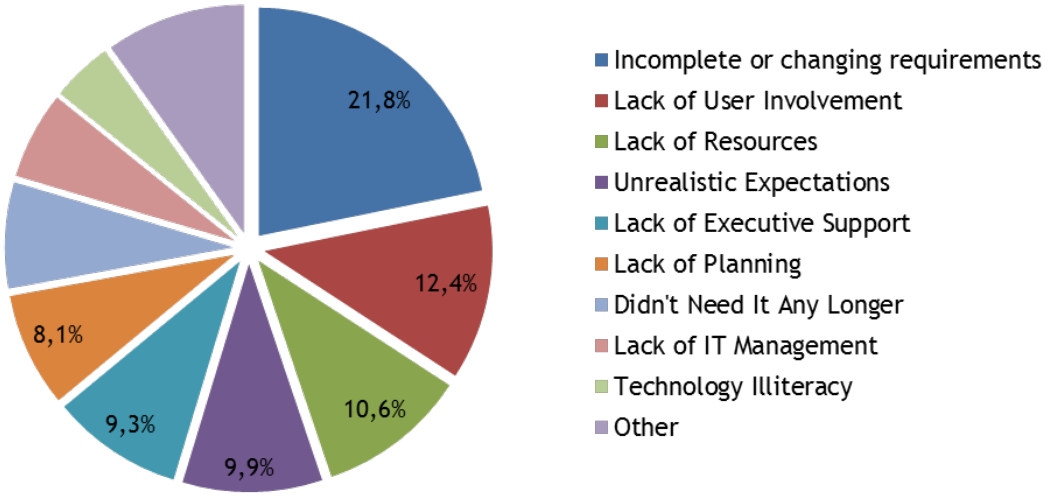
› Today we discuss:

› Needs analysis

› Concept Exploration

On Not Asking Questions

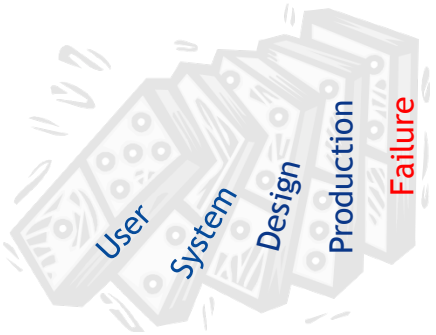
- Statistics on failing IT projects:
  - 30-40% of all projects are cancelled before completion
  - 70% fail to deliver expected features
  - Average project is 189% of original estimate
  - Average project misses schedules by 220%



- From : "CHAOS" by The Stanish Group, 1995

On What is Not Needed

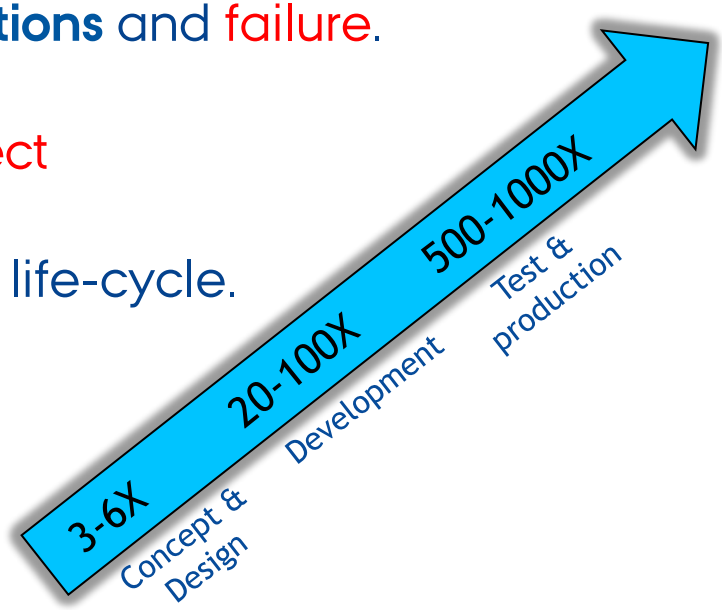
› The domino effect:



› Missing requirements have a domino effect throughout the project life-cycle.

› Missing **user requirements** lead to missing **systems requirements** which eventually lead to missing **design elements**, **missing functions** and failure.

› Recall that the cost of removing a defect increases exponentially as the project proceeds to later stages in the product life-cycle.



Purpose of Concept Development

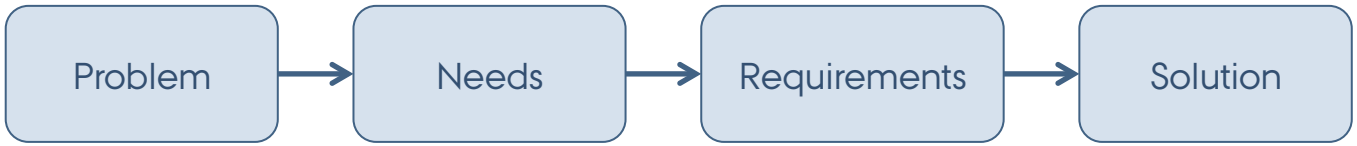
› Clear statements of objectives.

› State the initial problem and the need that is to be satisfied and what is required to achieve this.

› Define the characteristics of the set of acceptable solutions.

› Provide guidance in the selection of the most appropriate solutions.

› SE encompasses a structured process for eliciting needs and requirements.



Needs vs Requirements

› **Need**:
A customer's informal statement of what the customer would like to achieve. Sometimes referred to as a user input to requirements analysis.

› **Requirement**:
A formal statement of a feature, function, or attribute of a product that must be implemented in order for the product to *have* ***the mandated value for the customer, that is, to satisfy the needs***.

Needs Analysis

We must argue that the product satisfies the needs

TABLE 6.1. Status of System Materialization at the Needs Analysis Phase

	Phase					
		Concept development		*Engineering development*		
Level	Needs analysis	Concept exploration	Concept definition	Advanced development	Engineering design	Integration and evaluation
System	Define system capabilities and effectiveness	Identify, explore, and synthesize concepts	Define selected concept with specifications	Validate concept		Test and evaluate
Subsystem		Define requirements and ensure feasibility	Define functional and physical architecture	Validate subsystems		Integrate and test
Component			Allocate functions to components	Define specifications	Design and test	Integrate and test
Subcomponent		Visualize		Allocate functions to subcomponents	Design	
Part					Make or buy	

› Needs analysis takes place in the customer's world

Needs Analysis Process

› A process for arguing why the **requirements** satisfy the **needs**

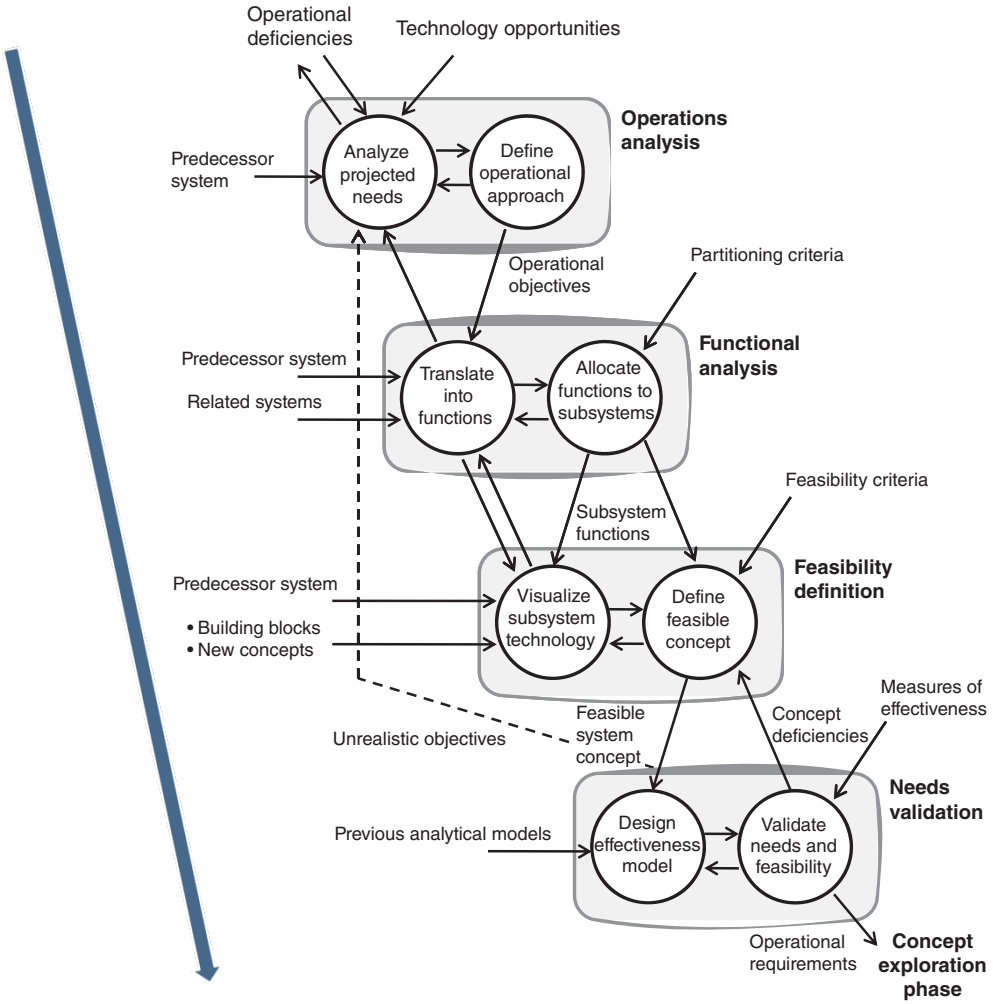


Figure 6.2. Needs analysis phase flow diagram.

From Needs Towards the Product

› Quality Function Deployment

› Our aim is to put forward an argument of why the product will satisfy the needs

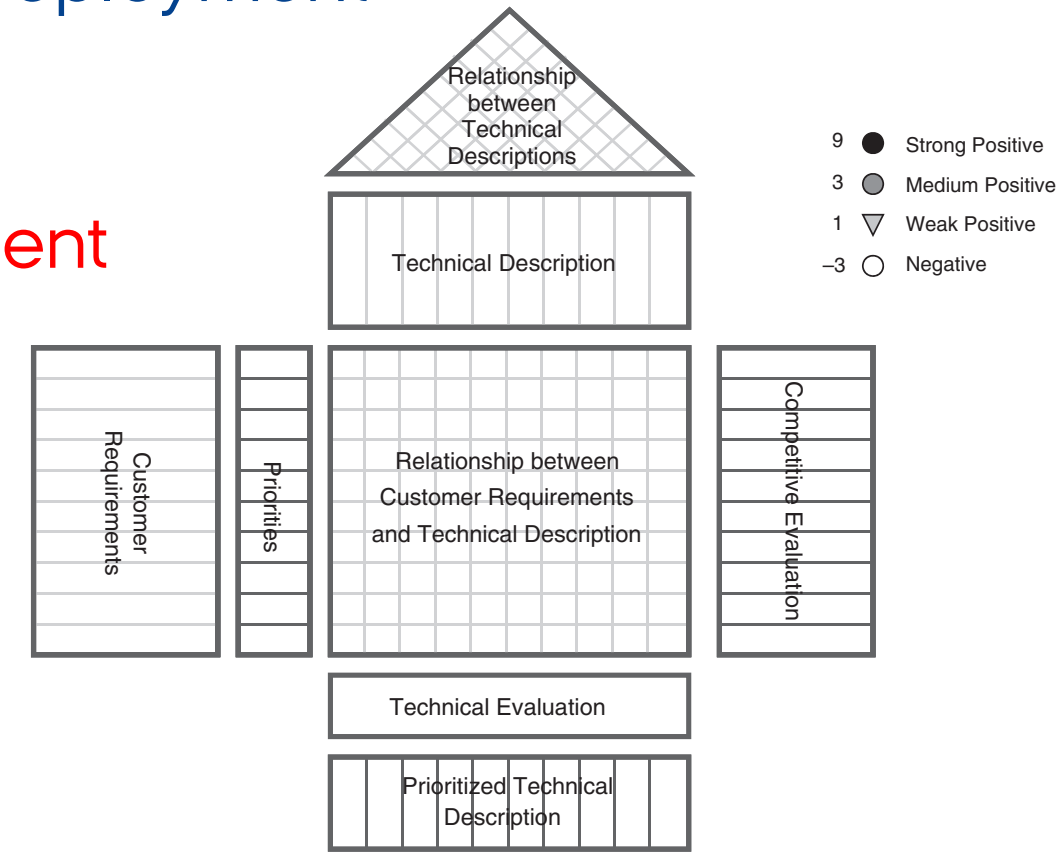


Figure 9.20. QFD house of quality.

› What are the possible outcomes of such reasoning?

From Needs to Requirements

› **Need**: new laws on lowering emissions of automobiles
› Translate into verifiable operational objectives:

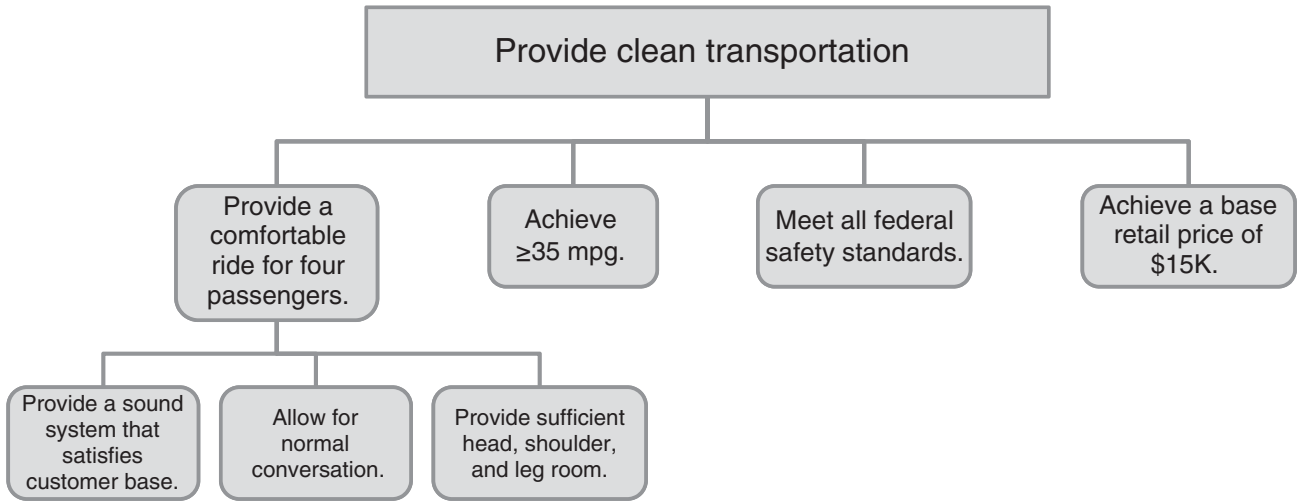


Figure 6.4. Example objectives tree for an automobile.

› Are these operational objectives verifiable? How?
› Are they feasible? Carry out functional analysis.
› Are they suitable for product implementation?

Operational Requirements

- Operational requirements derived from operational objectives
- They are the measureable reference against which the product will be judged
- They must be:
  - clear
  - complete
  - adequate
  - consistent
  - feasible
- Adequacy: Operational effectiveness analysis
- Use system performance parameters:
  - MOE: Measure of effectiveness (system level)
  - MOP: Measure of performance (component level)

Operational Effectiveness

- What can be analysed at this stage?
  - Reliability
  - Performance with respect to objectives
  - Safety
  - Cost

- Types of MOE: Measurement, Likelihood, Binary (Yes or No)

- Operational scenarios form the basis of the analysis
  - Mission objectives
  - Architecture (e.g. infrastructure, organisation)
  - Physical environment (e.g. terrain, recession or growth)
  - Competition (e.g. competitor product, storm, burglar)
  - General sequence of events

Concept Exploration

› Concept Development:

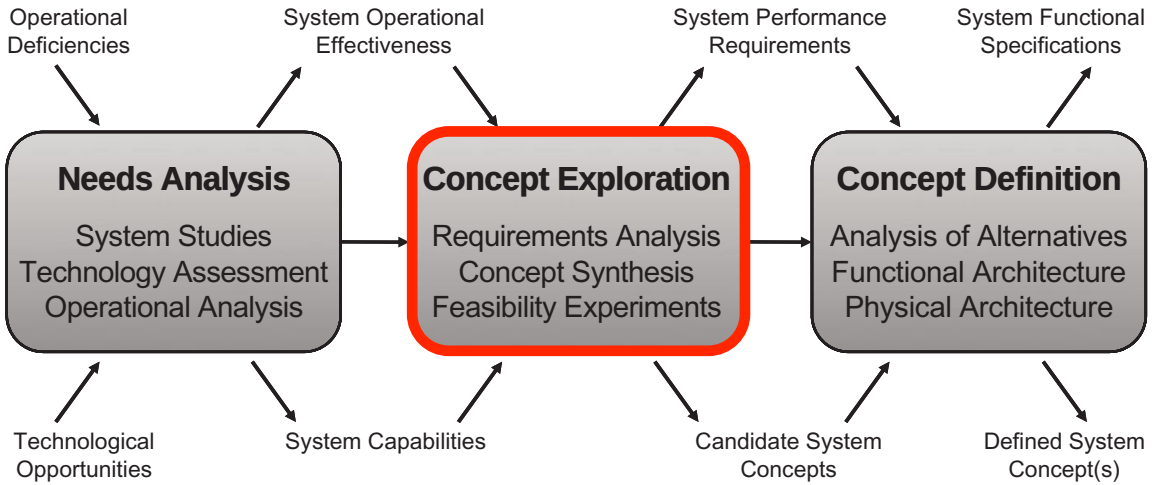


Figure 4.4. Concept development phases of a system life cycle.

› Today we discuss:

› Needs analysis

› Concept Exploration

Concept Exploration

Which solution is best suited to address the needs?

TABLE 7.1. Status of System Materialization of the Concept Exploration Phase

	Phase					
	Concept development			*Engineering development*		
Level	Needs analysis	Concept exploration	Concept definition	Advanced development	Engineering design	Integration and evaluation
System	Define system capabilities and effectiveness	Identify, explore, and synthesize concepts	Define selected concept with specifications	Validate concept		Test and evaluate
Subsystem		Define requirements and ensure feasibility	Define functional and physical architecture	Validate subsystems		Integrate and test
Component			Allocate functions to components	Define specifications	Design and test	Integrate and test
Subcomponent		Visualize		Allocate functions to subcomponents	Design	
Part					Make or buy	

› Needs analysis takes place in the customer's world

Concept Exploration Process

› A process for eliciting requirements and exploring implementations

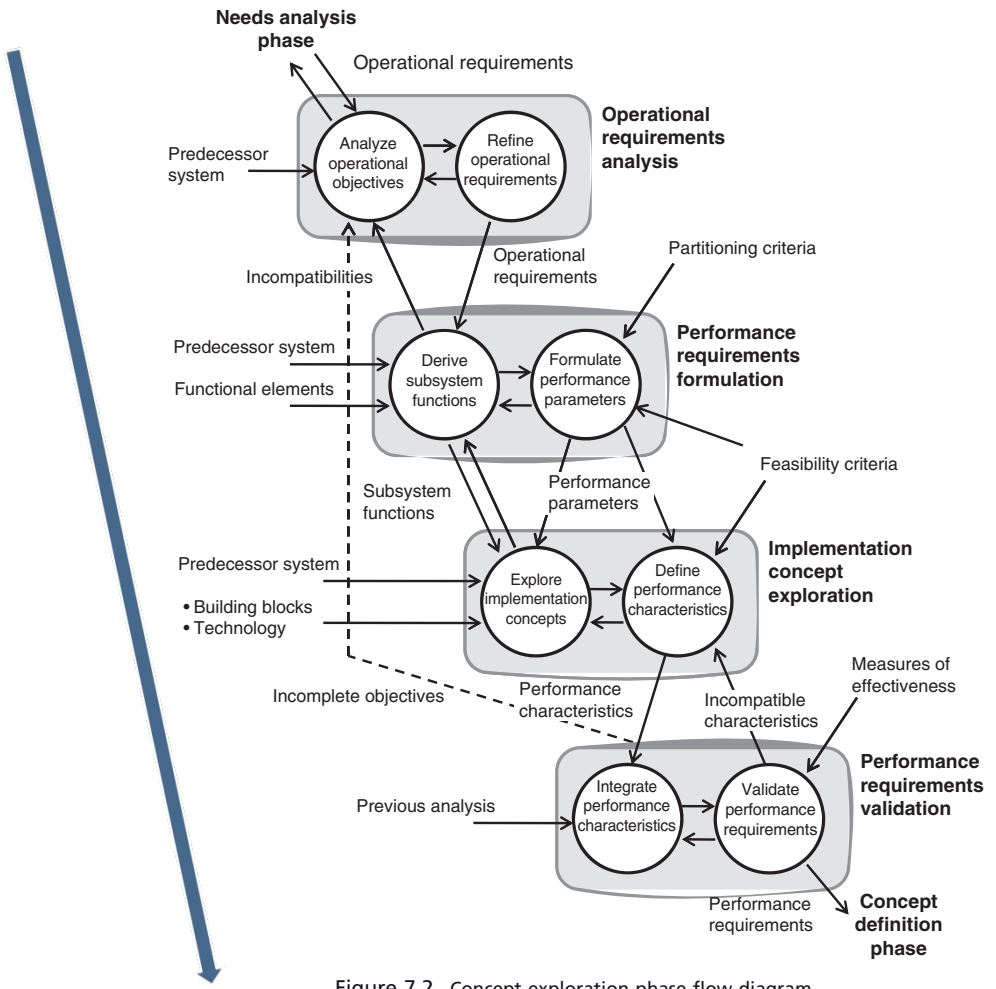


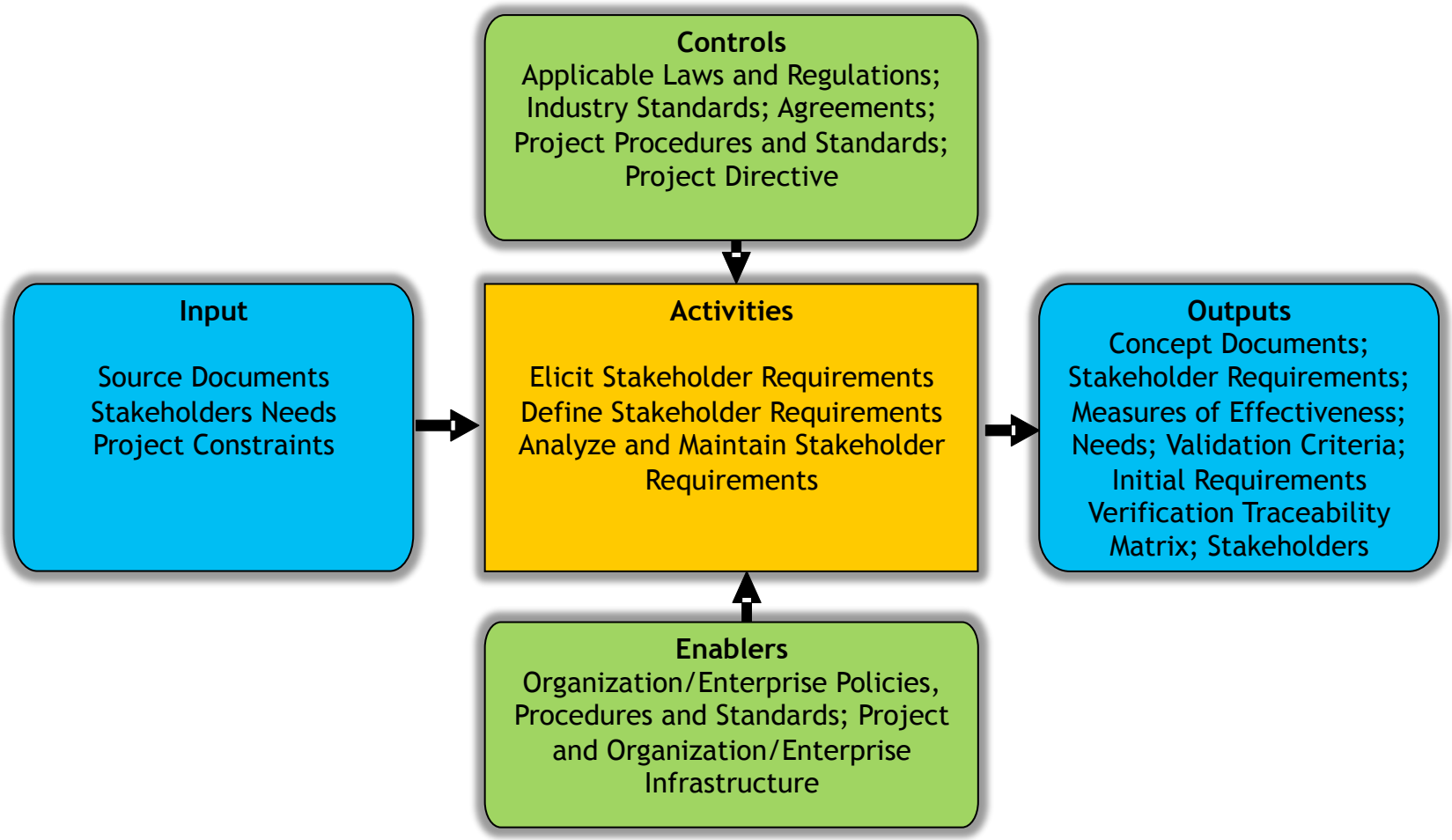
Figure 7.2. Concept exploration phase flow diagram.

Who Knows the Requirements?

- The Stakeholder Requirement Definition (SRD) process is used to:
- Identify who stakeholders are (for all phases of the project)
- Identify how the product is going to be used by the stakeholders
- Elicit, negotiate and define stakeholder expectations (including use cases, scenarios, operational concepts) in ways that can be measured (validated) when the system is completed.

- The result is the foundation from which the system is designed and the product is realised.

SRD Context Diagram



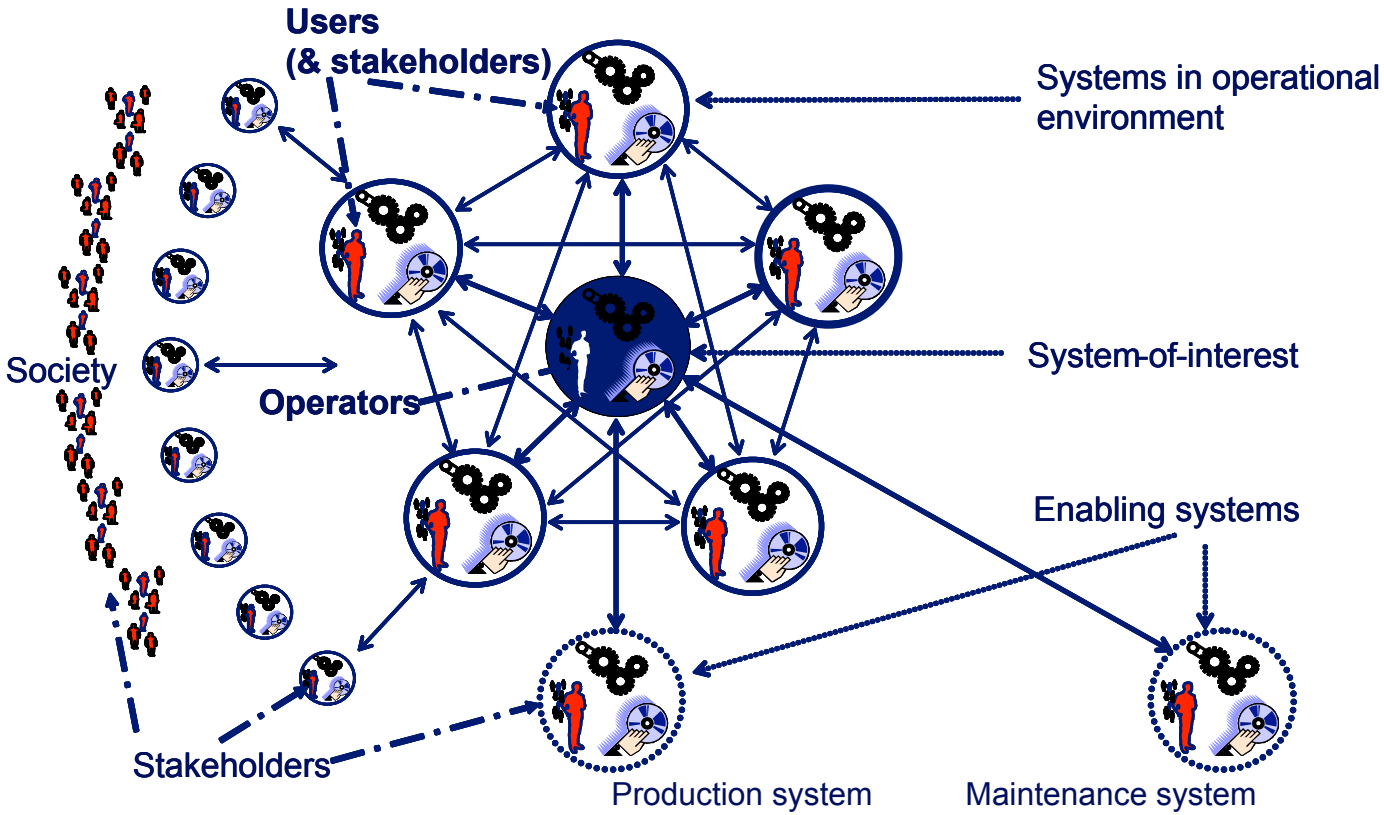
Who is a Stakeholder?

› A group or individual who is affected by or is in some way accountable for the outcome and undertaking.

› Stakeholders can be classified as:
› Customers – An organization or individual that has requested a product and will receive the product delivered.
› Other interested parties who provide broad overarching constraints within which the customer's need must be achieved, or who have influence on the success of the system

Where to Find Stakeholders

› Requirements elicitation captures the needs of stakeholders, operators, and users across systems boundaries.



Examples of Technical Stakeholders

Relative to the organization	Stakeholder	Typical expectation
External	Customer	Expected level of product quality, delivered on-time, affordable, life-cycle support & services
	Subcontractor/vendors	Well-defined requirements
	Local, National, Public	Product must not contaminate environment
Internal	Organization, management	Internal commitment met (cost, schedule), good status provided, compliance with org. policies, directives, procedures
	Project management	Expected technical work product delivered on time and can be used for decision making
	Technical team members	Clear tasks, job security, rewards, team work
	Functional organizations (e.g. test/QA)	Test support products available, clear test requirements, recognition for project help

How to Work with Stakeholders?

- How do we identify what stakeholders really want?
  - Interview and workshops with stakeholders, storytelling/storyboarding, prototyping/demonstration …
- How do we motivate stakeholders to give input to requirements as opposed of designing the system?
  - Ask “what” instead of “how”, focus on the value creating for the system instead of the problems/challenges in design; ask “What do you want this system to accomplish”
- From these inputs we built scenarios that we later can use with methods such as *ConOps* and *Use Cases*

Triumvirate of Conceptual Design

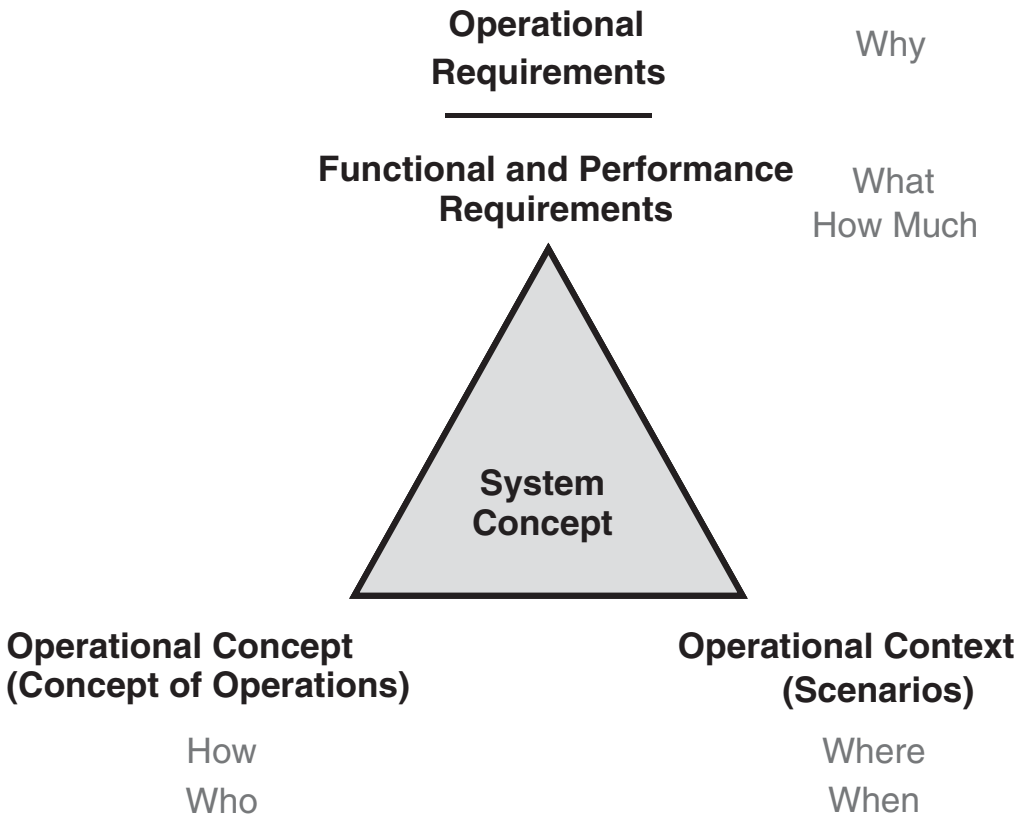


Figure 7.4. Triumvirate of conceptual design.

Concept of Operations

› A Concept of Operation (ConOps) document is produced early in the requirements definitions process to describe how the system will work (not what it will do it) and why (rationale)

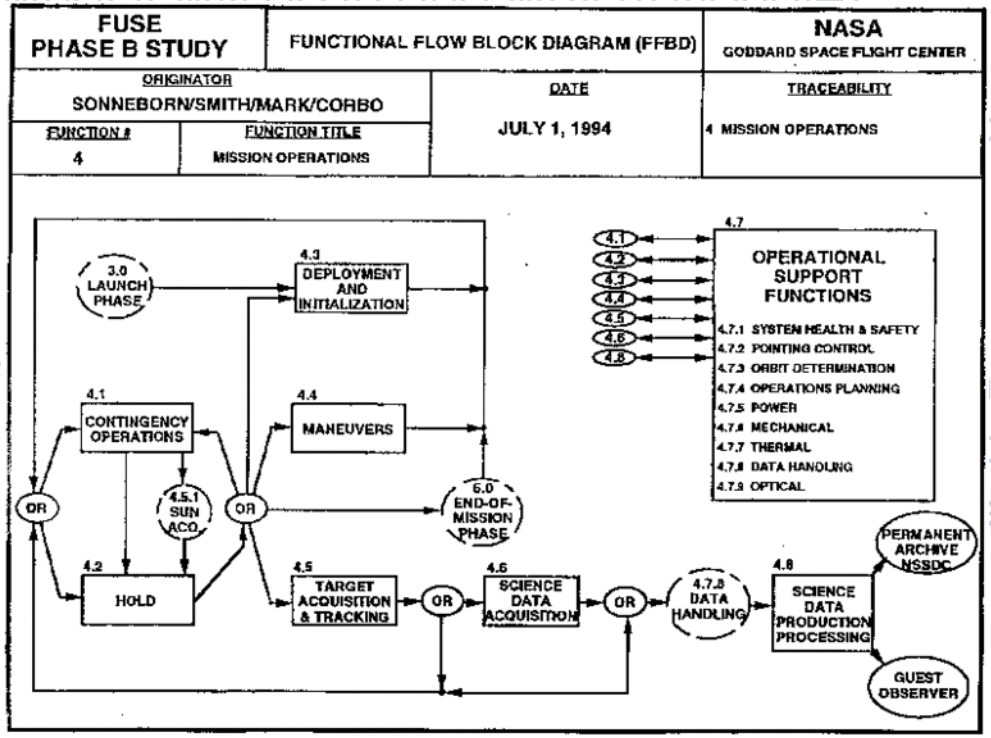


Figure 2.3 Functional Flow Block Diagram for Mission Operations

› al, top-level performance nale.

› functional flow block diagram of al "threads" specified. (At this plete operational concept or to e elements.)

› the customer during the early nal needs are clearly understood and incorporated into the design decision database for later inclusion in the system and segment specifications.

Identifying Requirements

› Scenarios
› are sequences of steps describing interactions between a user and a system.

› Use cases
› are a technique for capturing the functional requirements of a system.
› work by describing the typical interactions between the user of a system and the system itself.
› are sets of scenarios tied together by a common user goal.

Use Case Example

Scenario description:

The customer browses the catalog and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale. The system checks the authorization on the credit card and confirms the sale both immediately and with a follow-up e-mail.

Use case text:

Buy a Product

Goal Level: Sea Level

Main Success Scenario:

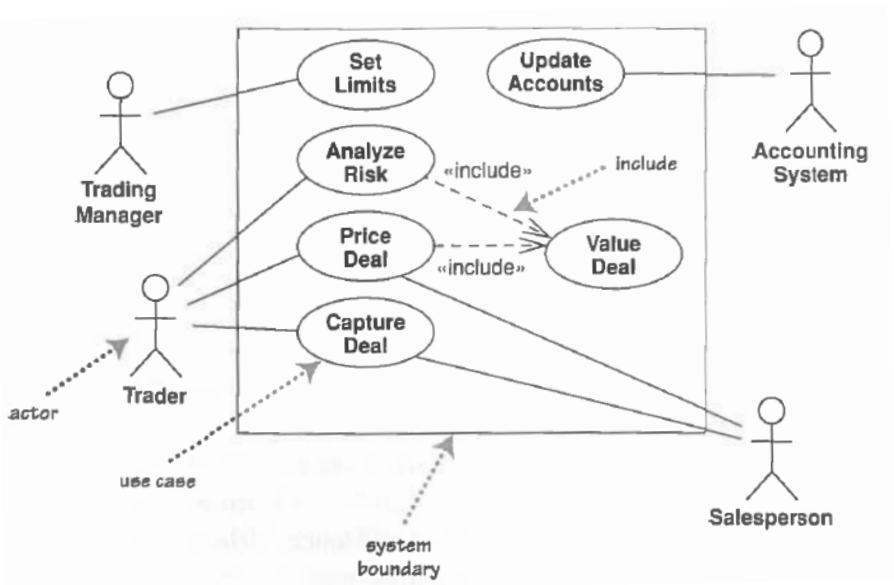
1. Customer browses catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming e-mail to customer

Extensions:

3a: Customer is regular customer
.1: System displays current shipping, pricing, and billing information
.2: Customer may accept or override these defaults, returns to MSS at step 6

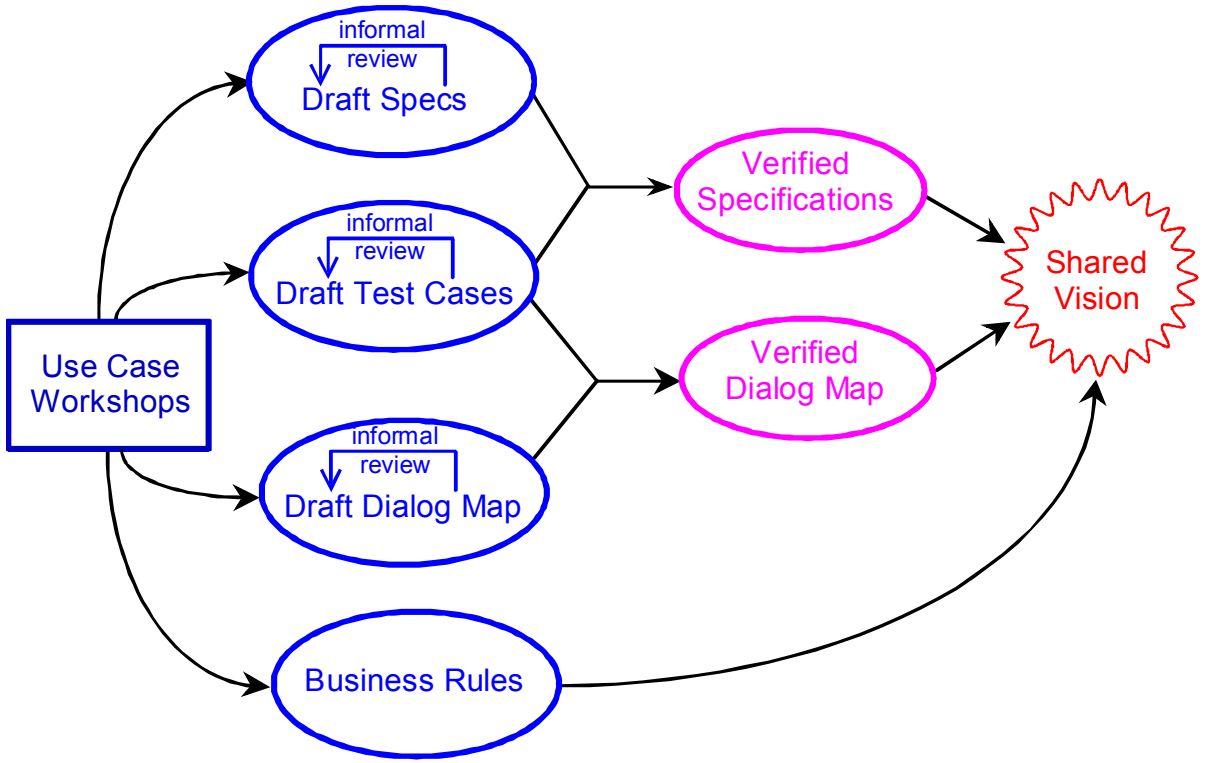
6a: System fails to authorize credit purchase
.1: Customer may reenter credit card information or may cancel

Use case diagram:



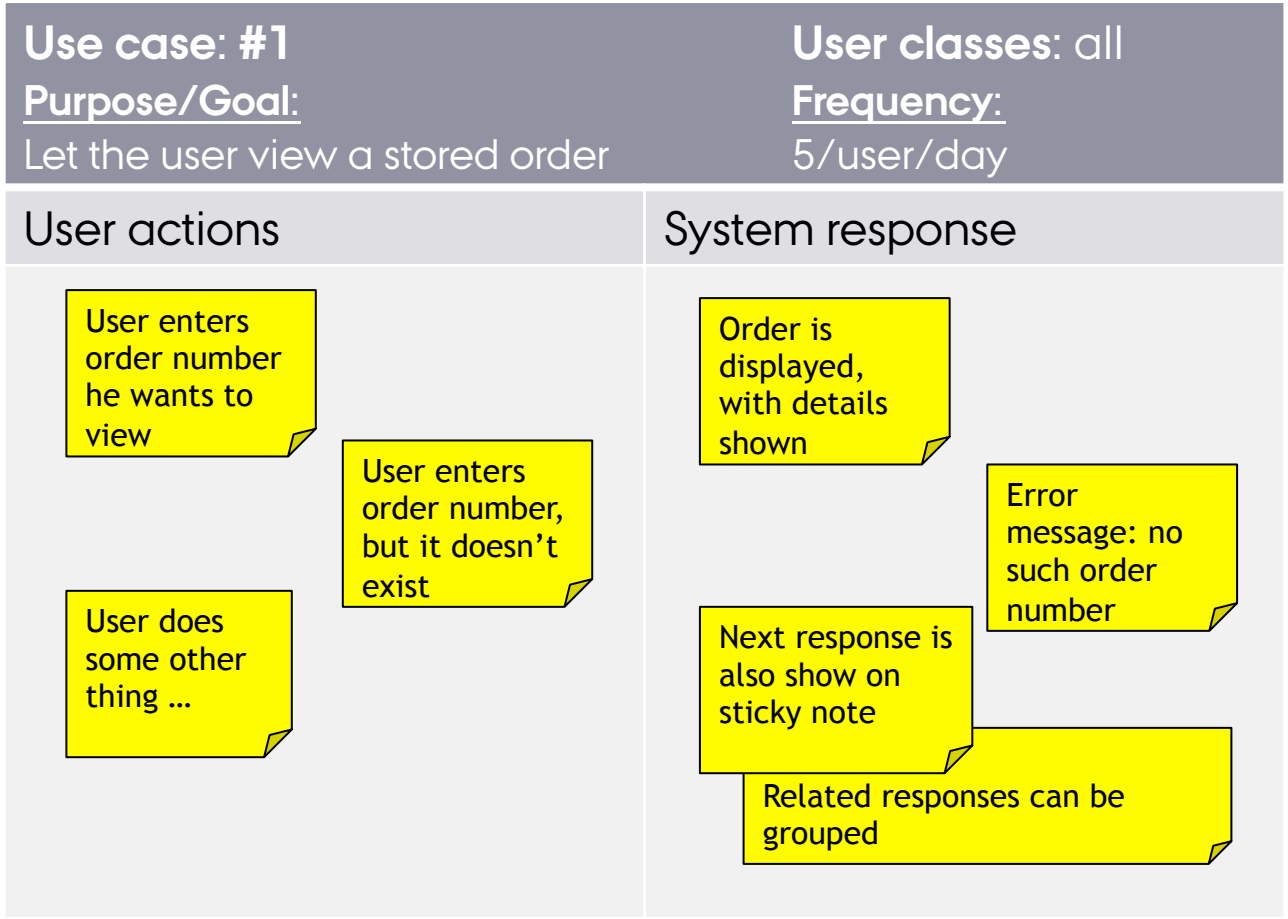
› Source: M. Fowler, UML distilled 3rd ed. Addison Wesley, 2004

Deliverables of the Use Case Method



› Source: "Listening to the Customer's Voice" by Wiegers, 1997

Use Case Flipchart created in a Workshop



› Source: "Listening to the Customer's Voice" by Wiegers, 1997

Requirements Engineering Process

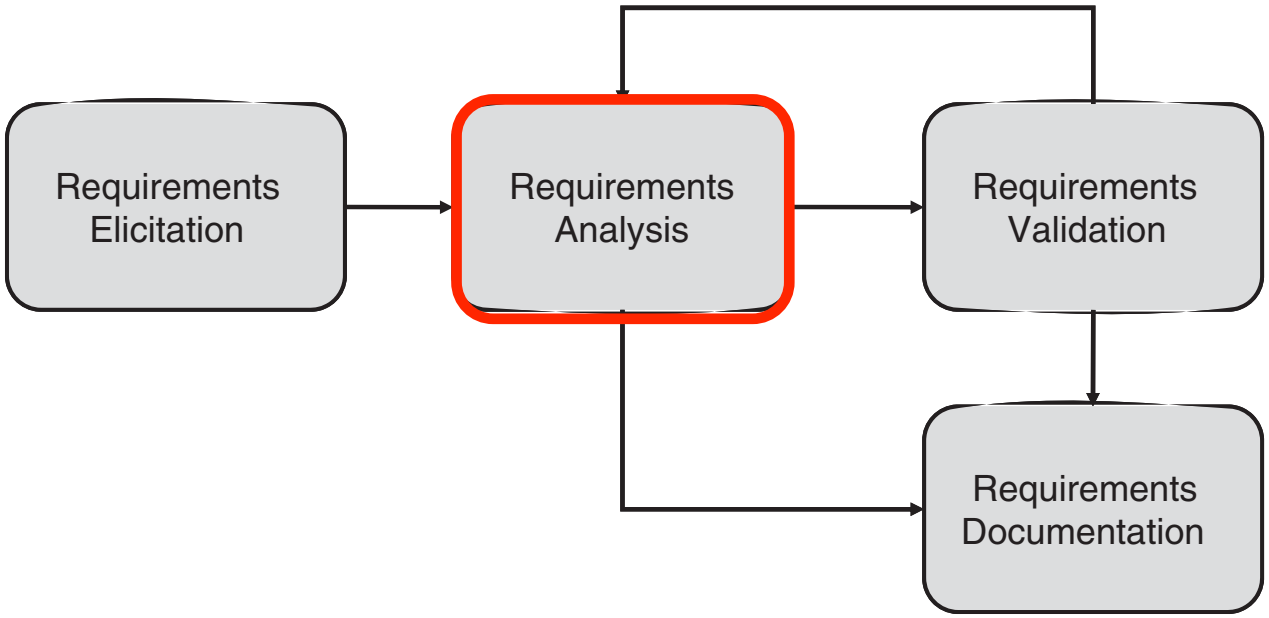
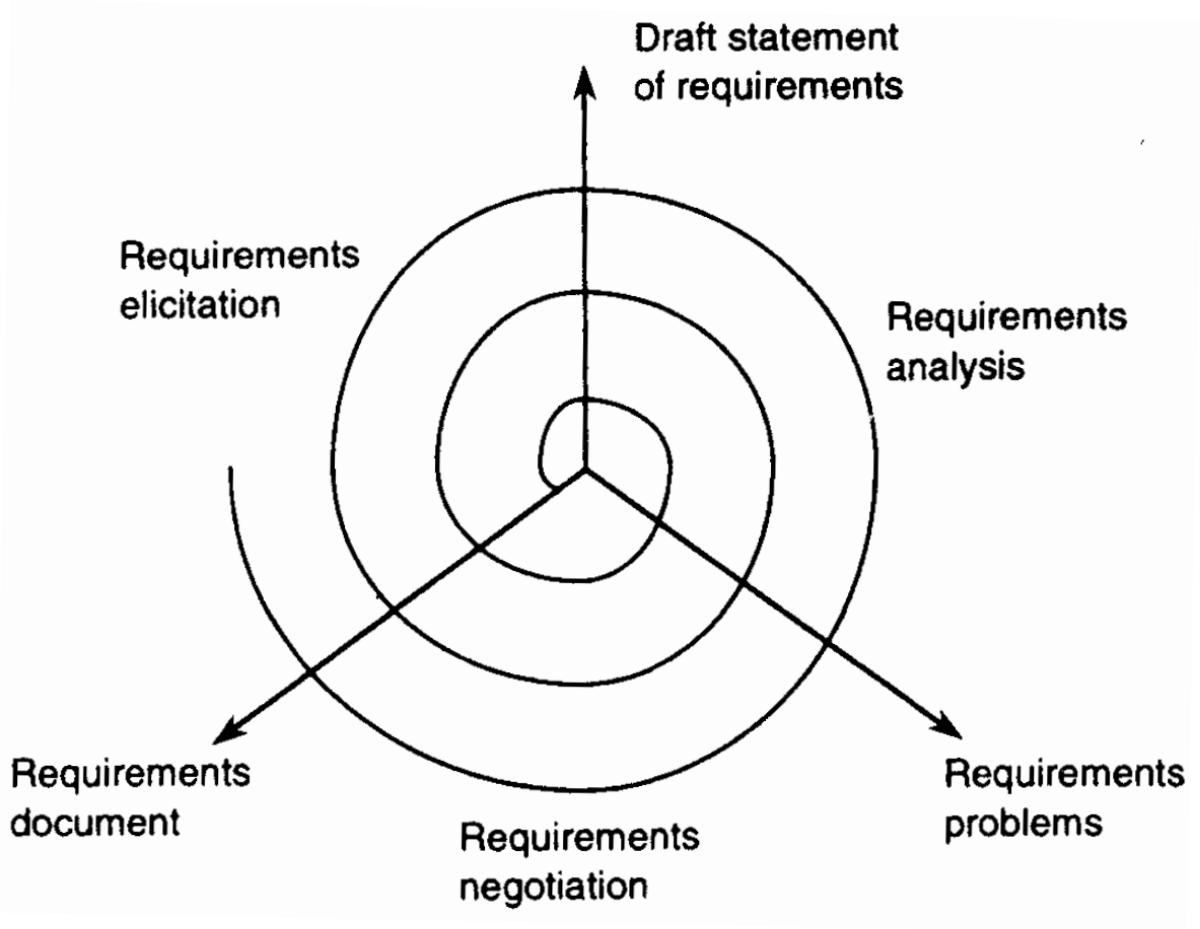


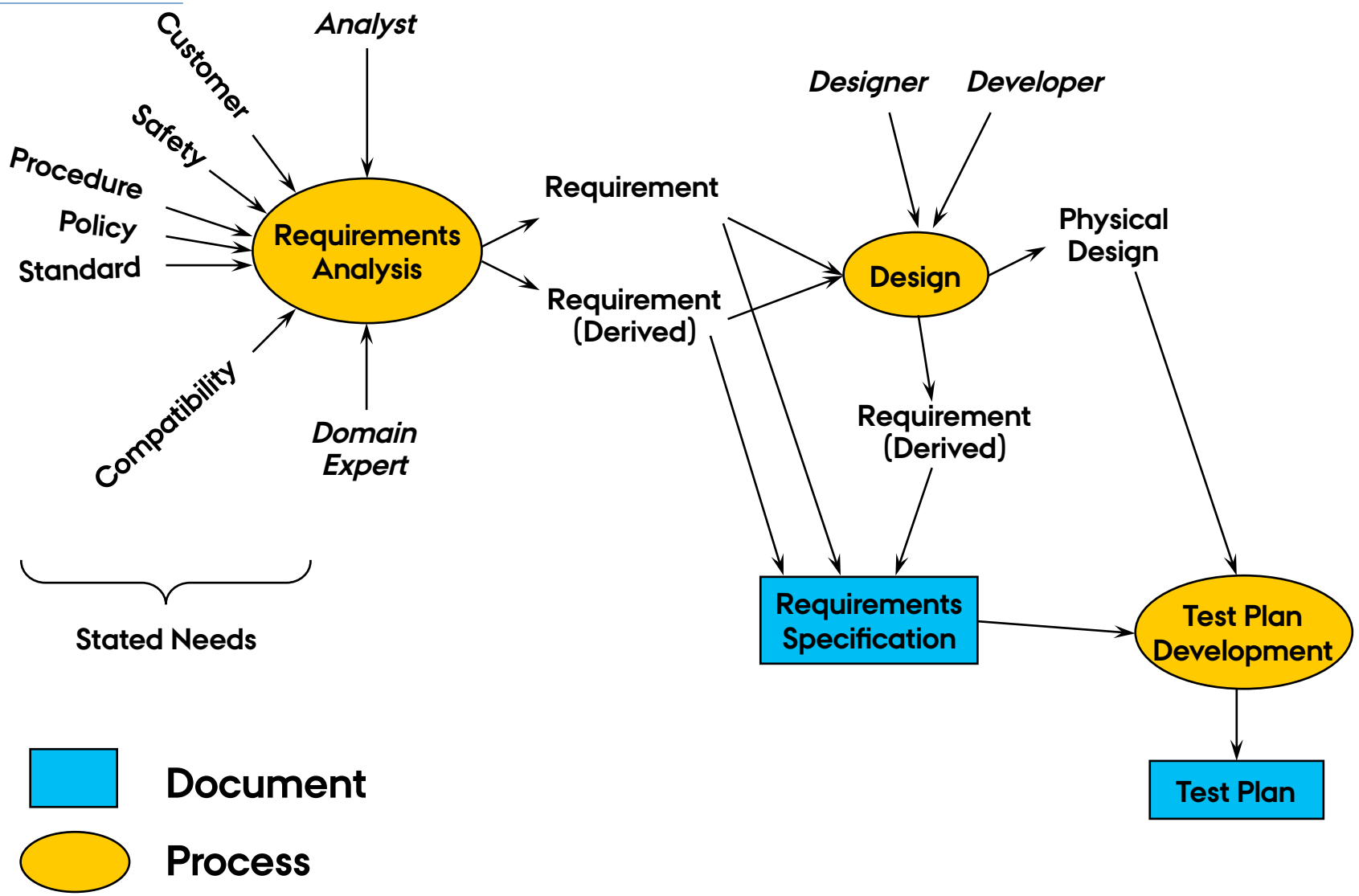
Figure 7.3. Simple requirements development process.

Iterative Requirements Engineering



› Source: Requirement Engineering

Requirements Analysis and Specification

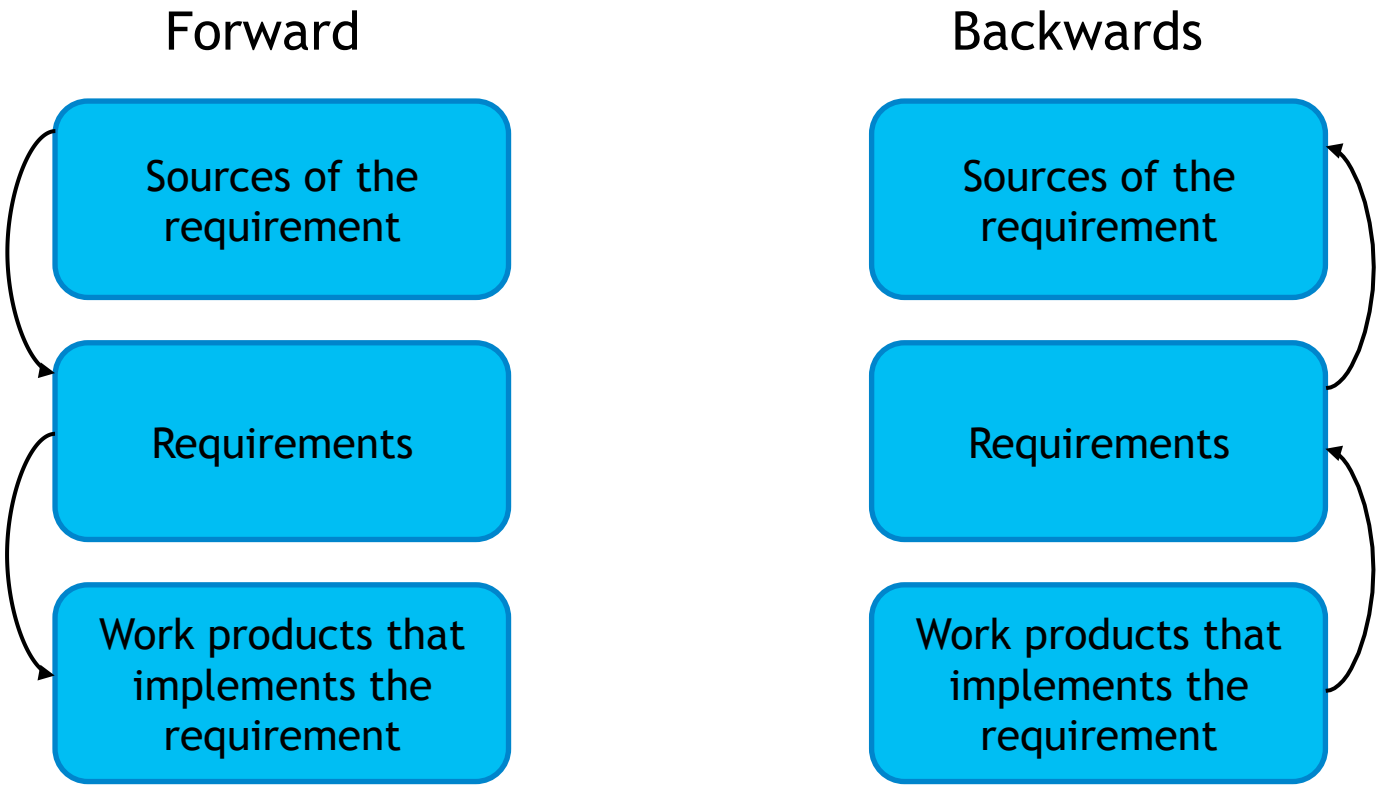


Examples of Typical Inputs Include

- New or updated customer needs, requirements, and objectives in terms of missions, measures of effectiveness, technical performance, utilization environments, and constraints.
- Technology base data including identification of key technologies, performance, maturity, cost, and risk.
- The outputs from a preceding acquisition phase.
- Requirements from contractually cited documents for the system and its configuration items.
- Technical objectives.
- Records of meeting and conversation with the customer.

- Source Requirements from the above are only a portion of the total system requirements
- Breakdown broad requirements statements will reveal need for additional clarification
- Concept of operation definition function may reveal need for additional clarification

Directions of Requirements Tracing



› The Requirements Verification Traceability Matrix (RVTM) is a useful tool to document the traceability aspects of a system

RVTM Example

- Forward

ID	User Requirements	System Reference
UF1	Add new customers	S1, S2
UBR4	Cannot add a user if they already exist	S1, S55
UD5	User surname is mandatory	S1
Etc.		

- Backward

System Reference	Functional Requirement	ID
S1	The system shall enable customers to be added	UF1, UBR4, UD5

In this example, there are a number of IDs

› UF is “User Functionality”

› UBR is User “Business Rule”

› UD is “User Data”

Requirement Verification and Traceability Matrix (RVTM)

Project name:		Business Area:	
Project manager:		Business Analysts lead:	
QA lead:		Target implementation date:	

Req. id.	Category or functional activity	Requirement description	Use case reference	Design document reference	Code or module reference	Test case reference	User acceptance validation	Comments

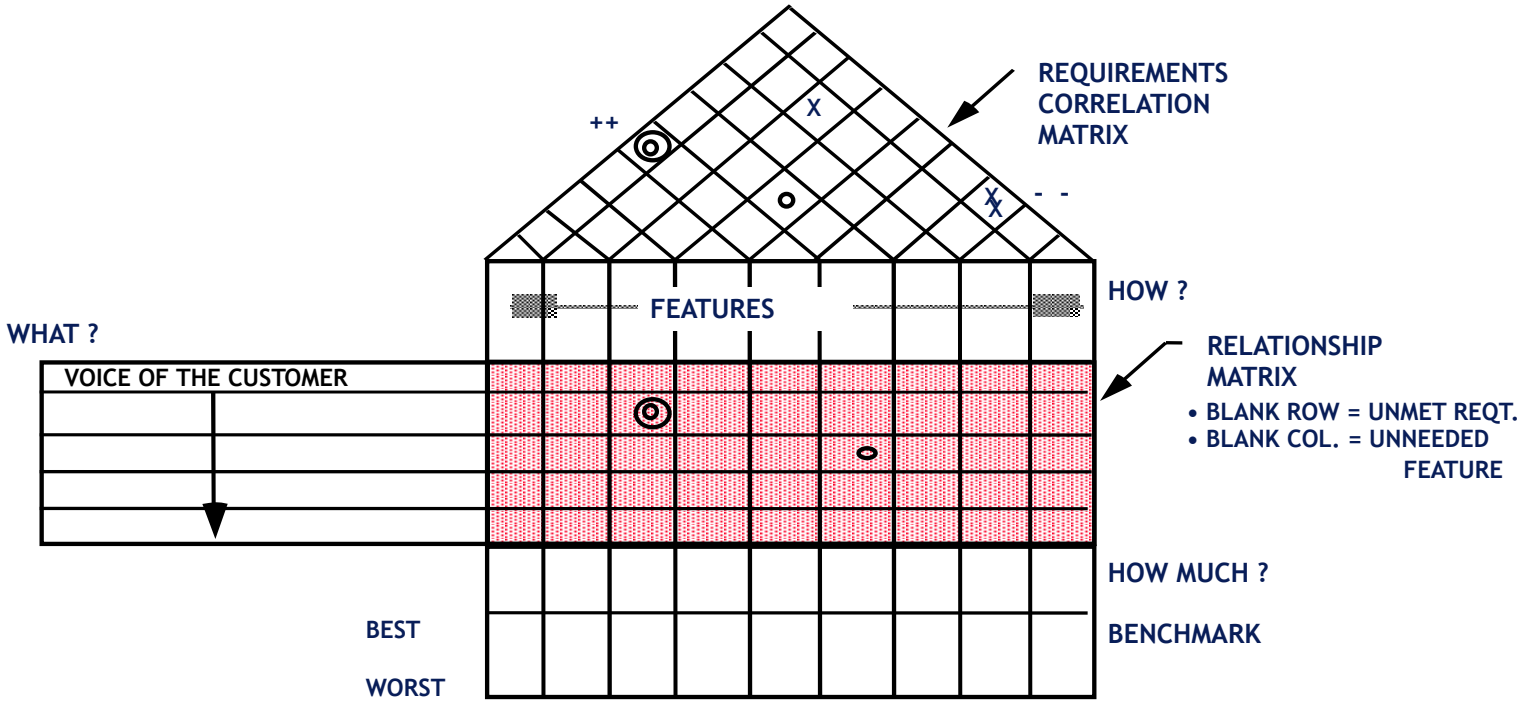
The System Requirement Specification(SRS)

- The output of this function will be a baseline set of complete, accurate non-ambiguous system requirements recorded in the decision database, accessible to all parties

- Non-ambiguous requirements must be broken down into constituent parts in a traceable hierarchy such that each individual requirement is:
  - Clear, unique, consistent, stand-alone and verifiable
  - Traceable to an identified source requirement
  - Not redundant, not in conflict with any other known requirement
  - Not biased by any particular implementation

Specification Satisfies Requirements

› Quality Function Deployment



› The “house of quality” is an effective way of arguing that a specification satisfies a given set of requirements

Towards the Implementation

› Identify possible system architecture and sub-systems

› Verify that performance requirements and characteristics are satisfied

› Choose the architecture to make this as easy as possible

› Evaluate the performance measures

› Vary the sub-systems and architecture

› Choose the best-performing configuration as blueprint for implementation

› But do not try to implement (yet)

Writing Better Requirements

Good requirements:

- Use complete sentences
- State subject and predicate
  - Subject is a stakeholder type or the system under discussion
  - Predicate is a condition, action or intended result.
- Uses of language consistently
- Specifies:
  - Desired goal or result (Stakeholder requirement)
  - Function (System requirement)
  - Constraint (either)
- Contains a success criterion or other measurable indication of the quality

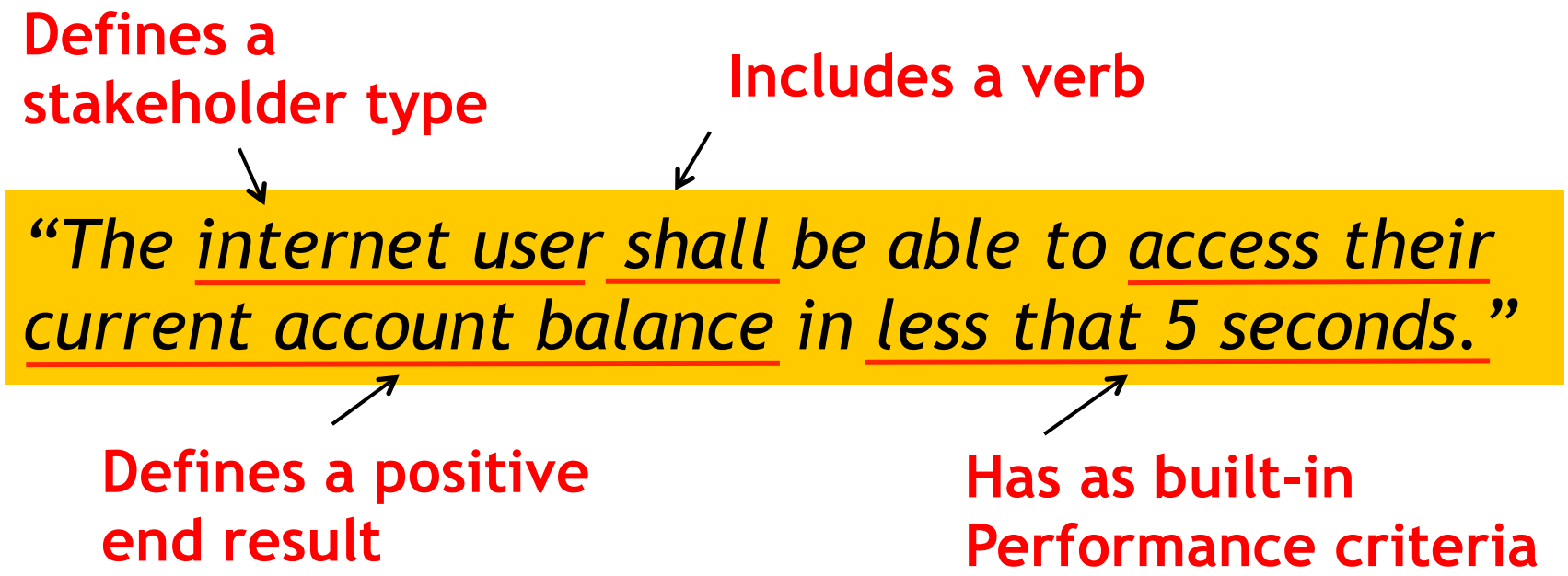
Essential Characteristics of Good Requirements

Each individual requirement should be:	
Necessary	Reflecting a real need
Clear	Unambiguous and not confusing
Consistent	Not in conflict with other requirements
Verifiable	It can be determined that the system meets the requirement
Traceable	Uniquely identified and can be tracked
Feasible	Can be accomplished within cost and schedule
Modular	Can be changed without excessive impact
Design-Free	Does not pose a specific solution on design (i.e., implementation free)
Positive	Written in the affirmative; not the negative, style

Essential Characteristics of Good Requirements

Each individual requirement should be:	
Correct	The requirement should be technically and legally achievable.
Complete	Express a whole idea or statement, i.e., stand alone and not being dependent on preceding or succeeding requirements.
Brief	Short, but not at the expense of clarity.
Prioritized	Needed to ensure that requirements of highest priority can get the proper attention.
Individual	Can be worked with as a self-contained entity
Owned	Allocate a “human” responsible.
Role	State the role of the responsible.
State/mode	Indicate progress: draft, approved, etc.

Anatomy of a Requirement



› The challenge is to seek out the stakeholder type, end result, and success measure **in every requirement you define**

Language Used in Requirements

- Use consistent language, for example:
  - "Shall" or "must" are mandatory
  - "Should" is optional, but omission must be justified
  - "May" is desirable

- Use consistent terminology
  - Define terms – use a glossary
  - Avoid using the same name for different things
  - Avoid using different names for the same thing

- *Compare:*
- *"I shall drown. No one will save me!"*
- *"I will drown. No one shall save me!"*

Examples of Using *shall*, *will*, *should*

› *Shall* – "The maximum speed of the treadmill shall be 10 mph." The treadmill must be built to enforce this limit.

› *Will* – "The treadmill speed will not exceed 10 mph." This is a statement of fact – no one is responsible for this happening.

› *Should* – "The treadmill speed should not exceed 10 mph." The treadmill can be built to exceed this limit if it has to, but it would be better if it did not.

Requirements: Ten Pitfalls to Avoid

1. Avoid ambiguity
2. Do not make multiple requirements
3. Never build in “let-out” or escape clauses
4. Do not ramble
5. Do not use vague indefinable terms
6. Do not speculate!
7. Do not mix up different kinds of requirements
8. Refrain from designing the system
9. Do not express suggestions or possibilities
10. Avoid wishful thinking

Pitfalls to Avoid (1/10)

- Avoid ambiguity
  - Write as clearly and explicitly as possible
  - Ambiguities can be caused by:
    - The word “or” to create a compound requirement
    - Poor definition (giving only examples or special cases)
    - Unclear definition (use of “etc.”, “… an do on”) – or –“shall include but not be limited to …”

- Example of ambiguity:

> *“The pilot and/or co-pilot shall also be able to hear or see a visible or audible caution/warning signal in case of emergency, hazard, etc…”*

Pitfalls to Avoid (2/10)

- Do not make multiple requirements
  - Keep each requirement as a single sentence
  - Requirements which contain conjunctions (words that join sentences) are “dangerous”
  - Dangerous conjunctions include “and, “or”, “with”, “also”

- Example of multiple requirements:

> *“The user shall be notified with a low battery warning lamp light when the voltage drops below 3,6 Volts and the current workspace or input shall be saved.”*

Pitfalls to Avoid (3/10)

- Never build in “let-out” or escape clauses!
  - Requirements with let-outs or escapes are dangerous
  - Do not ask for something definite, but later back down and allow for other options
  - Problem will arise in testing
  - Dangerous let-outs include: if, but, except, unless, although

- Example:

> *“The homeowner shall always hear the smoke detector alarm when smoke is detected unless the alarm is being tested or suppressed.”*

Pitfalls to Avoid (4/10)

› Do not ramble
  › Long sentences with arcane language
  › References to unreachable documents

› Example:

"Provided that the designated input signals from the specified devices are received by the user in the correct order where the system is able to differentiate the designators, the output signal shall comply this the required framework of section 3.1.5 to indicate the desired input state."

Pitfalls to Avoid (5/10)

- Do not use vague indefinable terms!
  - Many words used informally to indicate system quality are too vague to be verified
  - Vague terms include: user-friendly, highly versatile, flexible, to the maximum extent, approximately, as much as possible, minimal impact

- Example:

> *"The user shall be provided with a user-friendly front-end."*

Pitfalls to Avoid (6/10)

› Do not speculate!
  › There is no room for "wish list" – general terms about things that somebody probably wants.
  › Danger signs include vagueness about which type of stakeholder is speaking, and generalization words: usually, generally, often, normally, typically

› Example:

> *"The alarm system will probably have to operate over normal phone lines."*

Pitfalls to Avoid (7/10)

- Do not mix up different kinds of requirements!
  - Avoid mixing up requirements for stakeholders, system, an how the system should be designed, tested, or installed
  - Danger signs are very high level requirements mixed with database design, software terms, or very technical terms words

- Example:

> *"The user shall be able to view the currently selected channel number which shall be displayed in 14pt Swiss type on an LCD panel tested to Federal Regulation Standard 567-89 and mounted with shockproof rubber washers"*

Pitfalls to Avoid (8/10)

› Refrain from designing the system!
  › Requirements should specify the design envelope for the level required. If you supply too much detail you design the system (and increase the cost of systems)
  › Danger signs include names of components, materials, software objects, fields & records in the stakeholder or system requirements

› Example:

"The antenna shall be capable of receiving FM signals, using a copper core with nylon covering and a waterproof hardened rubber shield."

Pitfalls to Avoid (9/10)

- Do not express suggestions or possibilities!
  - Suggestions that are not explicitly stated as requirements are invariably ignored by developers.
  - Possible options are indicated with terms such as: may, might, should, ought, could, perhaps, probably

- Example:

> *"The network manager may be provided with possible network contentions point and should instantaneously re-route the traffic"*

Pitfalls to Avoid (10/10)

- Avoid wishful thinking!
  - Wishful thinking means asking for the impossible
  - Wishful terms include: 100% reliable, safe, handle all failures, fully upgradeable, run on all platforms

- Example:

> *"The network manager shall handle all unexpected error without crashing the system and be fully capable of managing future network configurations."*

Rate These Requirements

Are these good requirements? If not, recommend an improvement	
R1	The Order Entry system provides for quick, user-friendly and efficient entry and processing of all orders.
R2	Invoices, acknowledgements, and shipping notices shall be automatically faxed during the night, so customers can get them first thing in the morning,
R3	Changing report layouts, invoices, labels, and form letter shall be accomplished.
R4	The system OS shall be upgraded in one whack.
R5	The Easy Entry Navigator module converges the elements of Order Entry and communications, order processing, results processing, and reporting. The Order Entry module is fully integrated with the organization Intranet system and results are stored in the group's electronic customer record.
R6	Easy Entry shall be easy to use and require a minimum of training.
R7	The system has a goal that as much of the IS data as possible be pulled directly from the T&M estimate.

Automation