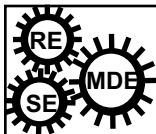


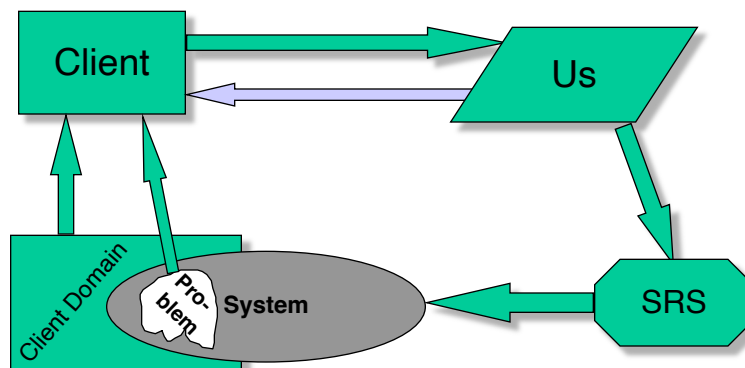
Requirements Analysis aka Requirements Engineering

Defining the WHAT

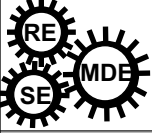
1



Requirements Elicitation Process



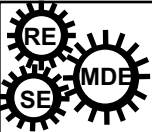
2



Requirements

- Specify functionality
 - model objects and resources
 - model behavior
- Specify data interfaces
 - type, quantity, frequency, reliability
 - providers, receivers
 - operational profile (expected scenarios)
 - stress profile (worst case scenarios)

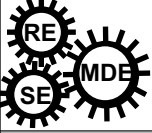
3



Requirements

- Specify interfaces
 - Control interfaces (APIs)
 - User interfaces - functionality and style
 - Hardware interfaces
- Specify error handling
- Identify potential modifications

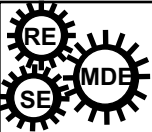
4



Requirements

- Identify necessary constraints
 - performance, security, reliability
- Identify areas of risk
 - alternatives to be explored
- Specify validation plans
- Specify documentation to be provided

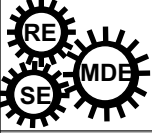
5



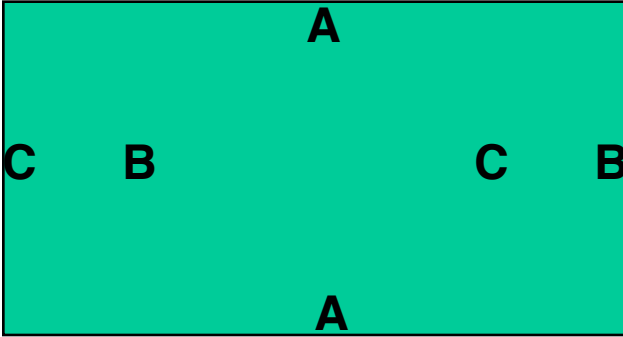
Analysis Principles

- Document reason for specific requirements
- Prioritize requirements
 - High, medium, low
- Ignore implementation details
 - Need to know feasible solutions can be developed
 - If feasibility is a concern, then propose alternatives to be explored
- Be prepared to change

6




Perspective and Early Binding of Constraints

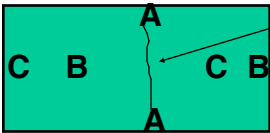


Connect like letters without crossing lines or leaving box.

7




Early Focus on Constraints



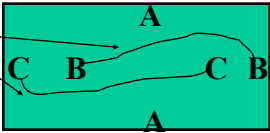
- A-A line seems to be only solution
 - But, is it really?
 - Need to examine domain and constraints more?
- What transforms or shifts would yield easier problem?

8




Focus Change

These choices still leave a path between A's



9




The Requirements Process

- A **requirement** is an expression of desired behavior
- A requirement deals with
 - objects or entities
 - the state in which they can be
 - functions that are performed to change states or object characteristics
- Requirements focus on the customer needs, not on the solution or implementation
 - designate **what behavior**, without saying how that behavior will be realized

Pfleeger and Atlee, Software Engineering: Theory and Practice, edited by B. Cheng,

Chapter 4.

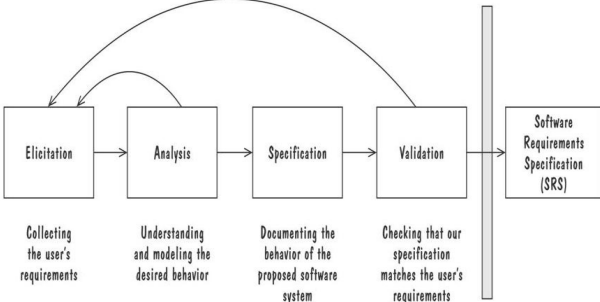
10



The Requirements Process

Process for Capturing Requirements


- Performed by the req. analyst or system analyst
- The final outcome is a Software Requirements Specification (SRS) document



Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

11




Requirements Elicitation

- Customers do not always understand what their needs and problems are
- It is important to discuss the requirements with everyone who has a stake in the system
- Come up with agreement on what the requirements are
 - If we can not agree on what the requirements are, then the project is doomed to fail

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

12



Requirements Elicitation

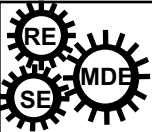
Stakeholders

- **Clients**: pay for the software to be developed
- **Customers**: buy the software after it is developed
- **Users**: use the system
- **Domain experts**: familiar with the problem that the software must automate
- **Market Researchers**: conduct surveys to determine future trends and potential customers
- **Lawyers or auditors**: familiar with government, safety, or legal requirements
- **Software engineers** or other technology experts

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

13



Requirements Elicitation

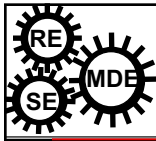
Means of Eliciting Requirements

- Interviewing stakeholders
- Reviewing available documentations
- Observing the current system (if one exists)
- Apprenticing with users to learn about user's task in more details
- Interviewing user or stakeholders in groups
- Using domain specific strategies, such as Joint Application Design, or PIECES
- Brainstorming with current and potential users

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

14



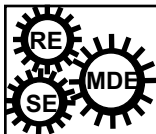
4.3 Types of Requirements

- **Functional requirement**: describes required behavior in terms of required activities
- **Quality requirement or nonfunctional requirement**: describes some quality characteristic that the software must possess
- **Design constraint**: a design decision such as choice of platform or interface components
- **Process constraint**: a restriction on the techniques or resources that can be used to build the system

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

16



Types of Requirements


Sidebar: Making Requirements Testable

- Fit criteria form objective standards for judging whether a proposed solution satisfies the requirements
 - It is easy to set fit criteria for quantifiable requirements
 - It is hard for subjective quality requirements
- Three ways to help make requirements testable
 - Specify a quantitative description for each adverb and adjective (e.g., “quickly,” “easy to use”) – how to measure?
 - Replace pronouns with specific names of entities
 - Make sure that every noun is defined in exactly one place in the requirements documents

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

17



Types of Requirements

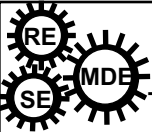
Resolving Conflicts

- Different stakeholder has different set of requirements
 - potential conflicting ideas
- Need to prioritize requirements
- Prioritization might separate requirements into three categories
 - essential: absolutely must be met
 - desirable: highly desirable but not necessary
 - optional: possible but could be eliminated

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

18



Types of Requirements


Two Kinds of Requirements Documents

- **Requirements definition:** a complete listing of everything the customer wants to achieve
 - Describing the entities in the environment where the system will be installed
- **Requirements specification:** restates the requirements as a specification of how the proposed system shall behave

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

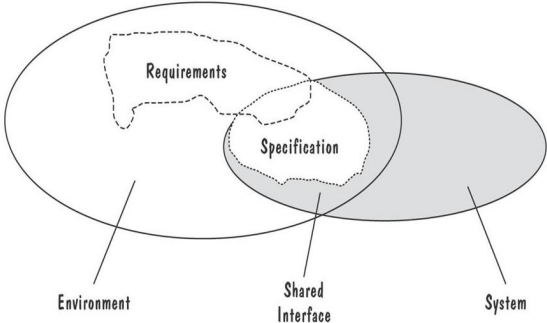
19



Types of Requirements

Two Kinds of Requirements Documents (continued)


- Requirements defined anywhere within the environment's domain, including the system's interface
- Specification restricted only to the intersection between environment and system domain



Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

20



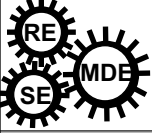
Characteristics of Requirements

- Correct
- Consistent
- Unambiguous
- Complete
- Feasible
- Relevant
- Testable
- Traceable

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

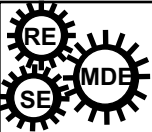
21



Non-Functional Requirements
(NFRs)

- Definitions
 - Quality criteria; metrics
 - Example NFRs
- Product-oriented Software Qualities
 - Making quality criteria specific
 - Catalogues of NFRs
 - Example: Reliability
- Process-oriented Software Qualities
 - Softgoal analysis for design tradeoffs

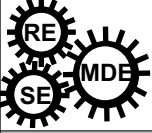
22



What are Non-functional Requirements?

- Functional vs. Non-Functional
 - Functional requirements describe what the system should do
 - functions that can be captured in use cases
 - behaviors that can be analyzed by drawing sequence diagrams, statecharts, etc.
 - ... and probably trace to individual chunks of a program
 - Non-functional requirements are global constraints on a software system
 - e.g. development costs, operational costs, performance, reliability, maintainability, portability, robustness etc.
 - Often known as software qualities, or just the “ilities”
 - Usually cannot be implemented in a single module of a program

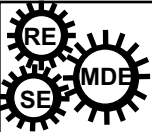
23



NFRs

- The challenge of NFRs
 - Hard to model
 - Usually stated informally, and so are:
 - often contradictory,
 - difficult to enforce during development
 - difficult to evaluate for the customer prior to delivery
 - Hard to make them measurable requirements
 - State them in a way that we can measure how well they have been met
 - Examples:
 - “Easy to use”
 - Reliable:
 - Secure:

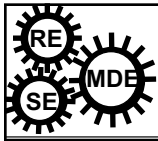
24



Example NFRs

<ul style="list-style-type: none">• Interface requirements<ul style="list-style-type: none">– how will the new system interface with its environment?<ul style="list-style-type: none">•User interfaces and “user-friendliness”•Interfaces with other systems• Performance requirements<ul style="list-style-type: none">– time/space bounds<ul style="list-style-type: none">•workloads, response time, throughput and available storage space•e.g. “the system must handle 1,000 transactions per second”– reliability<ul style="list-style-type: none">•the availability of components•integrity of information maintained and supplied to the system•e.g. “system must have less than 1hr downtime per three months”– security<ul style="list-style-type: none">•E.g. permissible information flows, or who can do what– survivability<ul style="list-style-type: none">•E.g. system will need to survive fire, natural catastrophes, etc	<ul style="list-style-type: none">• Operating requirements<ul style="list-style-type: none">– physical constraints (size, weight),– personnel availability & skill level– accessibility for maintenance– environmental conditions– etc• Lifecycle requirements<ul style="list-style-type: none">– “Future-proofing”<ul style="list-style-type: none">•Maintainability•Enhanceability•Portability•expected market or product lifespan– limits on development<ul style="list-style-type: none">• e.g., development time limitations,• resource availability• methodological standards• etc.• Economic requirements<ul style="list-style-type: none">– e.g. restrictions on immediate and/or long-term costs.
---	---

25



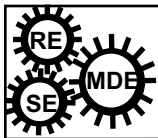
Invariant Requirements

- ***Invariant***: something that is constant, unchanging; always true
- ***Invariant Requirement***: observable functionality/property that must always be satisfied
- Examples:
 - Therac-25?
 - Cruise control?

CSE870: Advanced Software Engineering (Cheng): Intro to Software Engineering

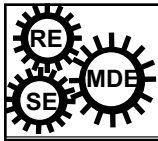
27

27



Elicitation techniques

28



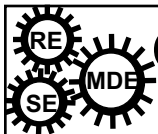
Technique: Initial client interview

Goal: Discover as many requirements as you can in a limited amount of time

Implications:

- Essentially an information-extraction process
- Ask open-ended questions
 - ask them in more than one way
- Your analysis should be very limited
 - OK to ask follow up questions, but don't get bogged down analyzing one requirement, or you will run out of time
 - **Never (during this interview):**
 - suggest a "better way to think about it"
 - express opinions on answers

29



Question Structure is Critical

What is the client's problem?

- what, precisely, is the problem to be solved?

When does the problem occur?

- what generates the problem?
- situations, are they new or old? Transient?

Where does the problem occur?

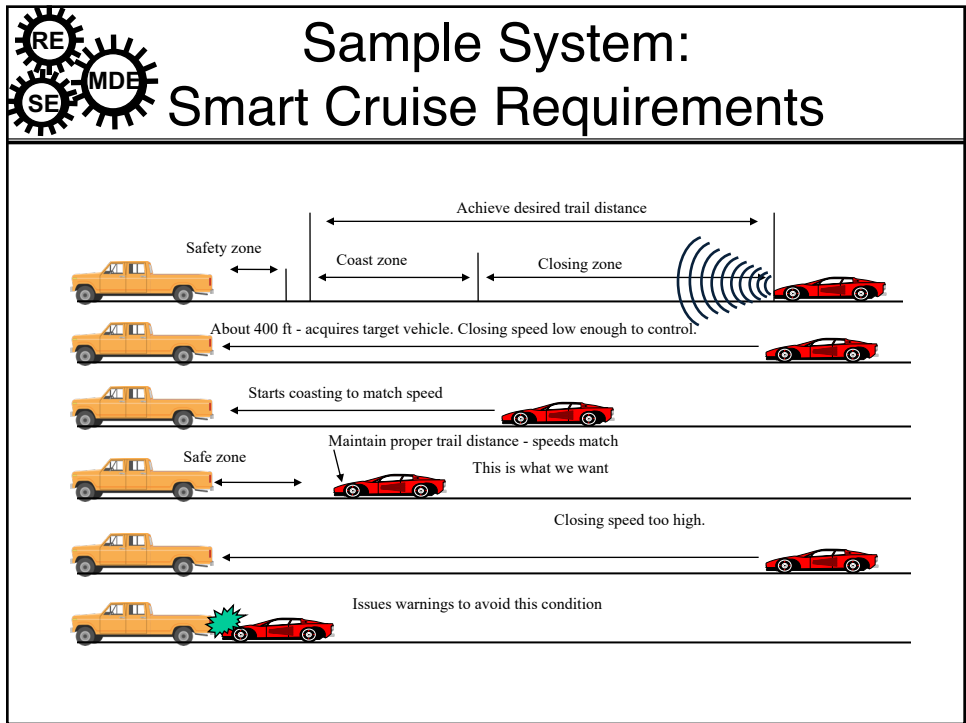
- what are the problem domain boundaries?

How is the problem handled now?

Why does the problem exist?

Remember, this is a diagnosis / information extraction process

30



31

RE

SE

MDE


Closed-ended questions

Q: When a vehicle cuts in front of the car,
you have to slow down quickly and not
hit it, right?

A: Yes

You learned absolutely nothing.

32



Open-ended questions

New reqt!


Q: What happens when a car cuts in front of you?

A: Well, if the lead car is too close, the driver has to intervene or else a crash results. I guess we need a warning light in this case. If the car is moving faster, you don't have to do anything. He's pulling away. I guess the only time brakes are used is when the closing speed is too high for the distance and yet within the capabilities of the system to slow down. But I guess if a collision is imminent, we should max out the braking.

Clarification

Now, we learned something...

33



Dialogue with different responses

Q: Tell me what should happen if a car cuts in front of our car too close to avoid a collision?

A: I guess since there is nothing the system can do, turn off the controller and hope the driver brakes in time.

Q: What? Are you nuts? We should at least try to stop. Shouldn't we?

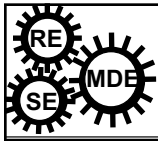
A: Perhaps...

Q: We have quite a bit of braking power in the system. What would happen if we used it here?

A: Well, I guess it could avoid a collision and at least get the car slowed down but the attorneys tell me we don't want the system active when a collision occurs.

Ah ha! Non-technical constraint

34



From elicitation to analysis...

Your interview should result in a large volume of facts which must be analyzed to derive requirements

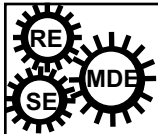
- Here “analysis” involves both analysis and synthesis
- Synthesis: attempt to compose a coherent “model” of the problem requirements

A model can be analyzed to:

- identify potentially inconsistent facts, and
- infer facts that should be true

Both of these issues must be clarified, often via a second client interview

35



Putative questions

Asks about a situation in a way that tests your model of the domain

SWE:


“If a lead vehicle turns, or otherwise is not in front of the car anymore, the car can resume the previous speed, correct?”

CLIENT:

“Yes, exactly.”

Very specific question that tests the idea of cruise plus collision avoidance

36



Sample Interview I

SWE:
Could you tell me about the cruise control system?

CLIENT
Yes, normal cruise control holds a fixed speed. What we want is to make the car "smart" so that it slows down when there is a vehicle in front of it.

SWE:
What does a driver currently do in this situation?


CLIENT
Currently, the driver can step on the brakes to disengage the cruise, or turn the cruise off completely. Or, not use the cruise.

SWE:
Why is turning off the cruise this way a problem?

Establish facts, but open ended. It's seemingly obvious what happens "now"

i.e., Why do you need "smart" cruise?
Try to get at the motivation for the problem

37



Sample Interview II

CLIENT
In an urban environment, say I-75 in Detroit, using the cruise becomes irritating, but really we are more interested in avoiding collisions.

SWE:
Tell me more about the collision avoidance aspect, please.

CLIENT
If we limit how close a lead vehicle can get, and control the speed while the car is in trail, the chances of a collision can be greatly reduced.

SWE:
How would a system avoid a collision in a typical scenario?

CLIENT
Suppose the driver is following a truck, but at a higher speed than the truck. As the car closes, the system could alter the speed to match the speed of the truck.

SWE:
What does the slowdown profile look like?


open, info gathering

The system is mis-named. This is good info

Looking for process/behavior information

Specific request for facts

38



Sample Interview III

CLIENT
Well, we have discovered that slowing down linearly over a long distance can lead to other cars cutting in front of you. This is also not what a human driver does. Instead, we continue at our current speed and start a coast when we compute that we will get too close.

SWE:
What is "too close"

CLIENT
Oh, within 2 seconds of trail distance

SWE:
Does that mean at 60 mph, 88 ft/sec, too close is 176 ft?

CLIENT
Yes, closer than 176 ft is too close.

Great insight


Very specific, to resolve ambiguity in domain terms

Ok, we can infer what this means

Time for a putative theorem to verify current model that resolves ambiguity

verification

39



Sample Interview IV

SWE:
What if a car cuts in front of you within the "safe" 2 second distance?

CLIENT
I guess since there is nothing the system can do. Turn off the controller and hope the driver brakes in time.

SWE:
The specs indicate we have a fair amount of braking power available. What would be the problem with using it here?

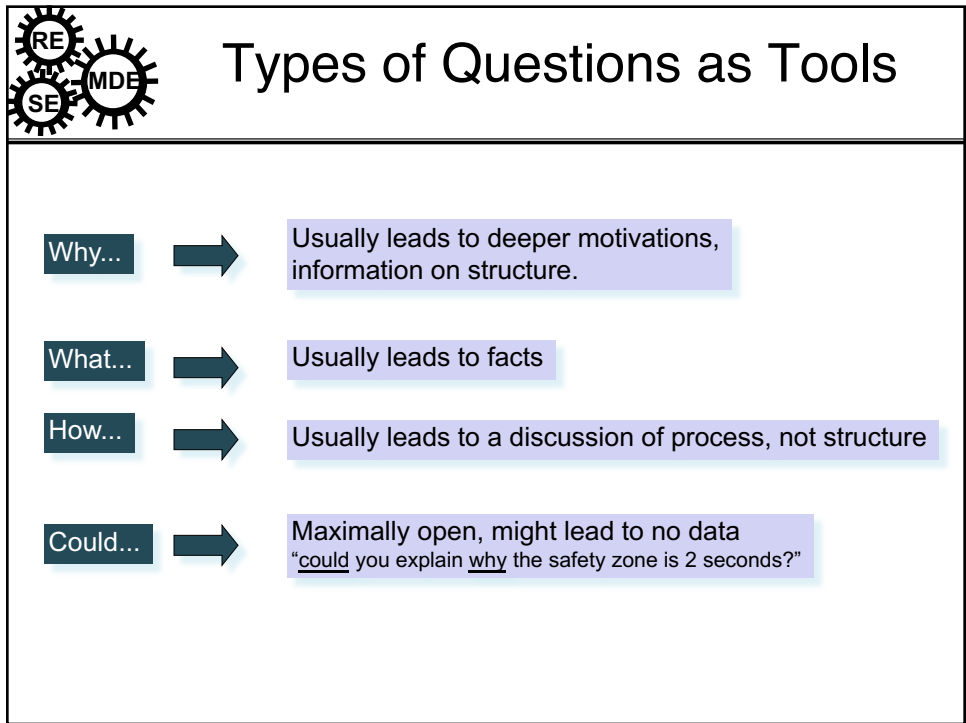
CLIENT
The system does have access to the brakes, which are anti-lock. Technically, we could apply the brakes, but at the moment, our attorneys tell us we'd rather not have the system active if a collision is imminent.

filling in model

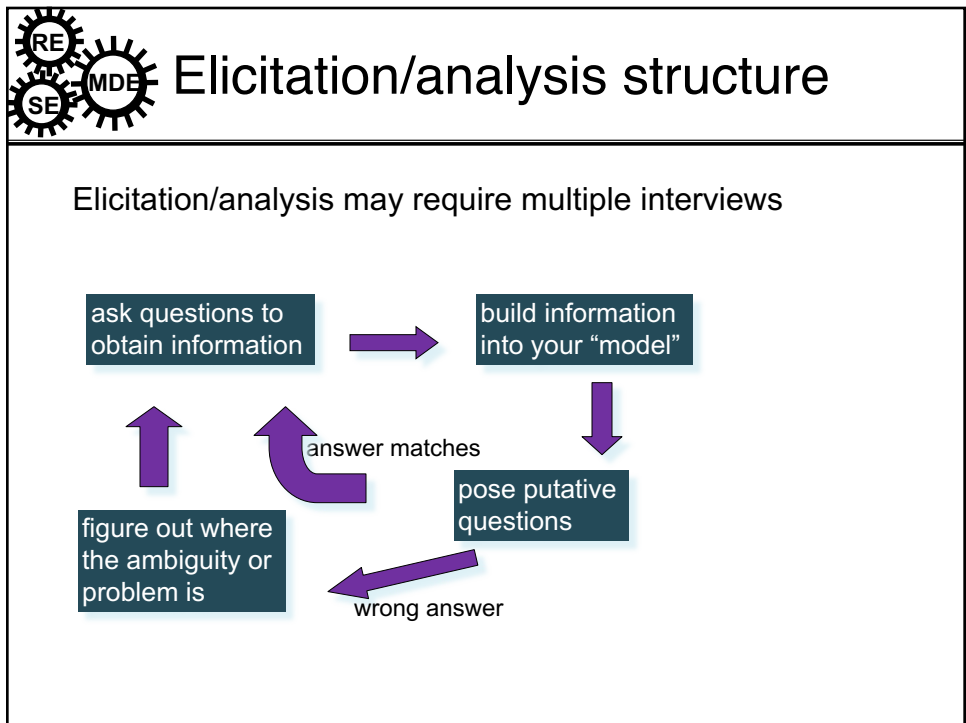
vs. "can't we use..."

A non-technical issue arises

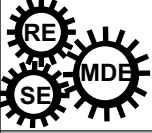
40



41



42



Summary

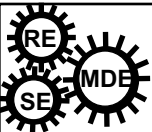
Elicitation is critical to:

- address the requirements-completeness problem
- support analysis, which aims to address the requirements-consistency problem

Client interviews are a useful tool, but:

- Must be carefully planned and orchestrated
 - Meetings should focus on a primary goal (e.g., information extraction vs. clarification)
- Big mistake to fail to plan for some iteration here

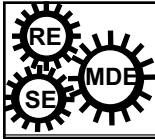
43



Reviewing a requirements document

<ul style="list-style-type: none">• Is it ambiguous?<ul style="list-style-type: none">– Carefully define terms and use these terms• Is it consistent?• Is it complete?<ul style="list-style-type: none">– Vague requirements– Omitted requirements• Is it verifiable?• Is it realistic?• Does it plan for change?	<ul style="list-style-type: none">• Does it not overly constrain the problem?• Have alternatives been considered and explored?• Is it clearly presented?<ul style="list-style-type: none">– Precise, concise, clear– diagram complex objects and behaviors• Is it what the customer wants?
---	--

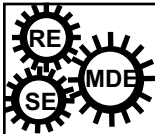
45



Why is requirements analysis difficult?

- Communication: misunderstandings between the customer and the analyst
 - Analyst does not understand the domain
 - Customer does not understand alternatives and trade-offs
- Problem complexity
 - Inconsistencies in problem statement
 - Omissions/incompleteness in problem statement
 - Inappropriate detail in problem statement

46



Escalator System Requirements

Two Signs on Escalator:

Shoes Must
Be Worn

Dogs Must
Be Carried

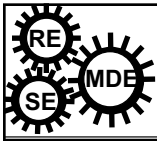
Consistent Conclusions:

You must have on shoes, and you must be carrying a dog.

If you have a dog, you have to carry it,
so you have to wear all the shoes you are carrying.

If you don't have a dog, you don't need to carry it, so you don't
have to wear shoes unless you are carrying some.

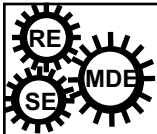
47



Why is requirements analysis difficult?

- Need to accommodate change
 - Hard to predict change
 - Hard to plan for change
 - Hard to foresee the impact of change


48



First Law of Software Engineering

“No matter where you are in the system lifecycle, the system will change, and the desire to change it will persist throughout the lifecycle.”

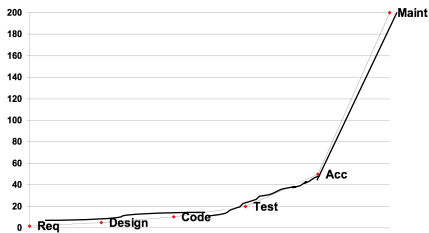
49



Reasons for changing requirements


- Poor communication
- Inaccurate requirements analysis
- Failure to consider alternatives
- New users
- New customer goals

- New customer environment
- New technology
- Competition
- Software is seen as malleable



Changes made after requirements are approved increase cost and schedule


50



Requirements Products

- Specification document
 - Agreement between customer and developer
 - Validation criteria for software
 - Problem statement in domain language
 - external behavior
 - constraints on system
- Preliminary users manual
- Prototype (*do not deliver the prototype!*)
 - If user interaction is important
 - If resources are available
- Review by customer and developer
 - Iteration is almost always required

51



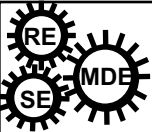
Modeling Notations

- It is important to have standard notations for Modeling, documenting, and communicating decisions
- Modeling helps us to understand requirements thoroughly
 - Holes in the models reveal unknown or ambiguous behavior
 - Multiple, conflicting outputs to the same input reveal inconsistencies in the requirements

Pfleeger and Atlee, Software Engineering: Theory and Practice,
edited by B. Cheng,

Chapter 4.

52



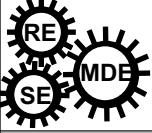
RE Exercise

- Customer:
- Scribe
- RE Stakeholders:
 - Business
 - Legal
 - Technical
 - User
- Google Doc:
 - Record problem stmt
 - List of requirements

CSE870: Advanced Software Engineering (Cheng): Intro to Software Engineering

53

53

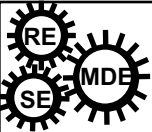


RE Exercise Debrief

- Testable requirements
- Define terms
- Explicitly identify and describe elements of system that will contribute to requirements
- Nonfunctional requirements: think about how to “assess” (i.e., test for satisfaction).

CSE870: Advanced Software Engineering (Cheng): Intro to Software Engineering54

54



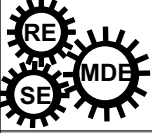
RE Exercise

LMS:

- 1. Should turn off the system when driver takes control How much angle? Experiment
- 2. System should only be active if its confident in lane
 - a) Status light for status of system
 - b) Gentle sound notification should alert status
- 3. The driver can ‘override’ the lane keeping system with enough turn force (a nudge) on the steering wheel, and a visual indicator will display the user is trying to override.
- 4. In the event of poor weather conditions, the lane keeping system will be temporarily disabled.

CSE870: Advanced Software Engineering (Cheng): Intro to Software Engineering55

55



RE Exercise

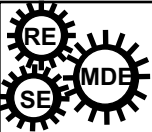
ACC:

1. If an emergency situation occurs, the system must activate full brakes
2. The car should maintain the speed that the user set it at, until there is a car in front going a slower speed
3. Subjective (Non-functional):
 - a) The system should be easy to enable and should have an intuitive design
 - b) The display should be easy to understand and should show the speed, and the distance of the car in front
4. Auto stop car if there is a slower car in front
 - a) Distance to car: 3 seconds distance
 - b) If too close, stop, notification to driver (sound, flashing light) to indicate slow down

CSE870: Advanced Software Engineering (Cheng): Intro to Software Engineering

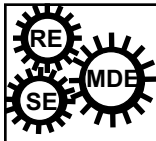
56

56



DETOUR TO UML MODELING

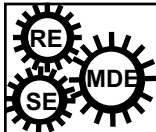
57



Approaching a Problem

Where do we start?
How do we proceed?

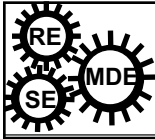
58



Where Do We Start?

- Start with the requirements
 - Capture your goals and possible constraints
 - Environmental assumptions
- Use-case analysis to better understand your requirements
 - Find actors and a first round of use-cases
- Start conceptual modeling
 - Conceptual class diagram
 - Interaction diagrams to clarify use-cases
 - Activity diagrams to understand major processing

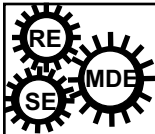
59



How Do We Continue?

- Refine use-cases
 - Possibly some “real” use-cases
 - Using interface mockups
- Refine (or restructure) your class diagram
 - Based on your hardware architecture
 - For instance, client server
- Refine and expand your dynamic model
 - Until you are comfortable that you understand the required behavior
- Identify most operations and attributes

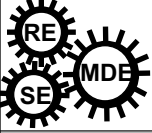
60



How Do We Wrap Up?

- Refine the class diagram based on platform and language properties
 - Navigability, public, private, etc
 - Class libraries
- Identify most operations
 - Not the trivial get, set, etc.
- Write a contract for each operation
- Define a collection of invariants for each class
- Implement

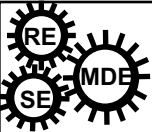
61



Putting It Together

- **Principles**
 - Rigor and Formality
 - Separation of Concerns
 - Modularity
 - Abstraction
 - Anticipation of Change
 - Generality
 - Incrementality
- ***Notion of Process***

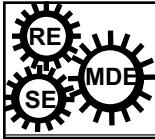
62



Overview: Steps to Follow

- Map out environment as-is
- Map out environment as required
- Decide on systems boundaries / goals
- List actions with types
- Define terms
- Construct model
- Challenge model
- Modify as required

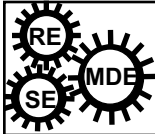
67



Analysis: Steps to follow

- Obtain a problem statement
- Develop use cases (depict scenarios of use)
- Build an object model and data dictionary
- Develop a dynamic model
 - state and sequence diagrams
- Verify, iterate, and refine the models
- Produce analysis document

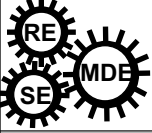
68



Use Cases

- High-level overview of system use
- Identify scenarios of usage
- Identify actors of the system:
 - External entities (e.g., users, systems, etc.)
- Identify system activities
- Draw connections between actors and activities
- Identify dependencies between activities (i.e., extends, uses)

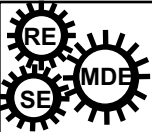
69



Analysis: Conceptual (Domain) Model

- Organization of system into classes connected by associations
 - Shows the static structure
 - Organizes and decomposes system into more manageable subsystems
 - Describes real world classes and relationships

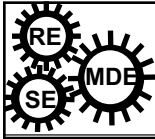
70



Analysis: Conceptual Model

- Object model precedes the dynamic model because
 - static structure is usually better defined
 - less dependent on details
 - more stable as the system evolves

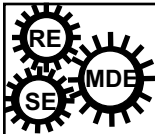
71



Analysis: Domain Model

- Information comes from
 - The problem statement and use cases
 - Expert knowledge of the application domain
 - Interviews with customer
 - Consultation with experts
 - Outside research performed by analyst
 - General knowledge of the real world

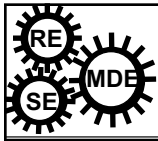
72



Client View of Domain

- Clients can't be expected to have rigorous or formal view of domain
- Hence, can't be expected to be completely aware of domain-problem relationship
- Some knowledge is *explicit*
 - Easier to get at
- Some knowledge is *implicit* ("everybody knows...")
 - Many constraints are implicit
 - Hard to get at

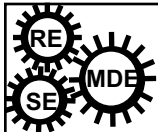
73



Object Model: Steps to follow

- Identify classes and associations
 - nouns and verbs in a problem description
- Create data dictionary entry for each
- Add attributes
- Combine and organize classes using inheritance

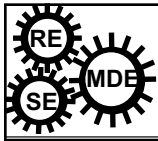
74



Analysis: Dynamic model

- Shows the time dependent behavior of the system and the objects in it
- Expressed in terms of
 - states of objects and activities in states
 - events and actions
- State diagram summarizes permissible event sequences for objects with important dynamic behavior

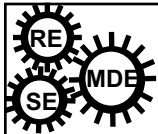
75



Dynamic Model: Steps to follow

- Use cases provide scenarios of typical interaction sequences
- Identify events between objects (Sequence Diagram)
- Prepare an event trace for each scenario
- Build state diagrams
- **Match events between objects to verify consistency**

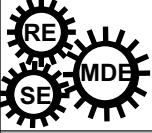
76



Analysis: Iteration

- Analysis model will require multiple passes to complete
- Look for inconsistencies and revise
- Look for omissions/vagueness and revise
- Validate the final model with the customer
 - Pose scenarios from various perspectives
 - Look for consistency in responses

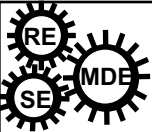
77



Object Model: main embedded system
objects or classes

- Controller object
 - might be made up of several controllers
 - is the brains of the system.
 - Takes input from the sensors and gives instructions to the actuators.
- Sensor object
 - environmental objects that gives information to controller.
 - Can be passive (thermometer) or active (button).
- Actuator object
 - Mechanical device to realize an effect

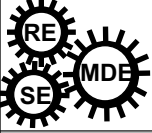
78



Meeting Purposes

- Disseminate information (including stating a problem)
- Gathering opinions
- Confirming consensus
- Social requirements
 - team building
 - approval

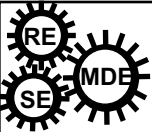
79



Meeting Requirements

- Agenda
- Leader
- Action list
 - With assignments so we know who is doing what.
 - Timelines so we know when it's to get done.
- Summary
 - Something happened or there would not have been a meeting. Record it briefly.

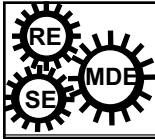
80



Project Issue List

- Every issue goes on the list
 - Date and brief description
- Make assignment to get it resolved
- Put resolution on list.
 - “Close” issue.
- 1st version usually generated on 1st read of problem statement.
 - And then, back to the customer...

81



Interviewing

- Have a list of things you want to know.
- Listen.
- Listen.
- Ask open-ended questions.
- Don't express (show, say) opinions on answers. Just record, and think.
- Listen.
- Ask questions more than one way.