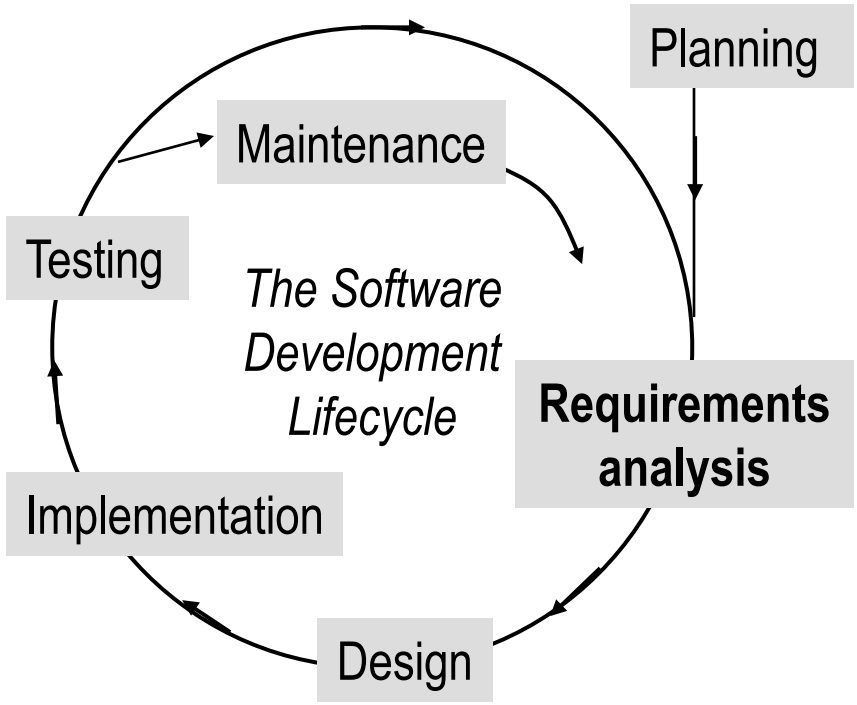
SOFTWARE REQUIREMENTS

Chapter 10



Phase most relevant to this chapter is shown in bold

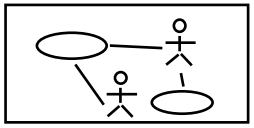
Learning goals of this chapter

- Why the term requirements *analysis*?
- What is the value of writing down requirements?
- Where do requirements come from?
- What is the difference between high-level and detailed requirements?
- What is the difference between functional and non-functional requirements?
- How do you document requirements?
- What does traceability mean?
- How do agile methods handle requirements?
- What are good tips for student project requirements analysis?

A Typical Example of Software Lifecycle Activities

Software Lifecycle Activities

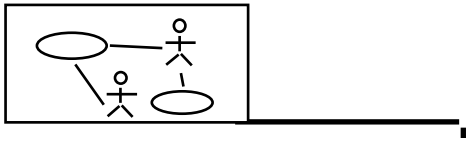
Requirements Elicitation	Requirements Analysis	System Design	Detailed Design	Implemen-tation	Testing



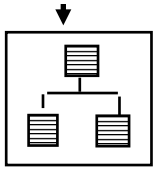
Use Case Model

Software Lifecycle Activities

Requirements Elicitation	Analysis	System Design	Detailed Design	Implementation	Testing



Expressed in terms of

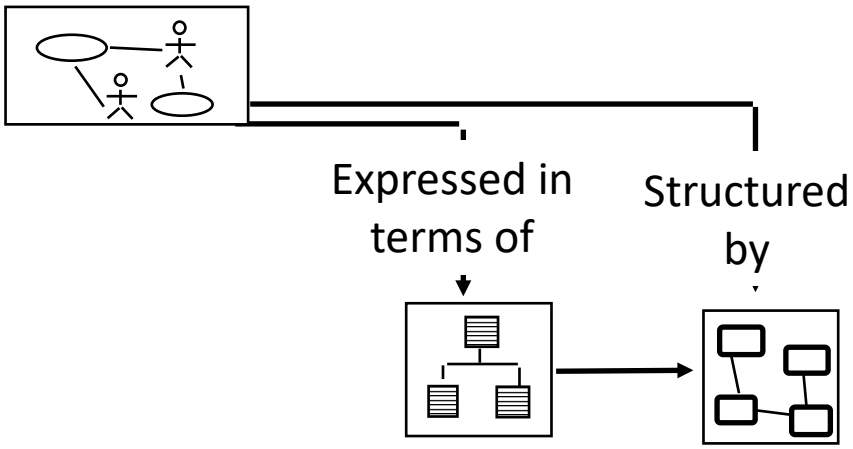


Use Case Model

Application Domain Objects/ Entities/ Terms

Software Lifecycle Activities

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



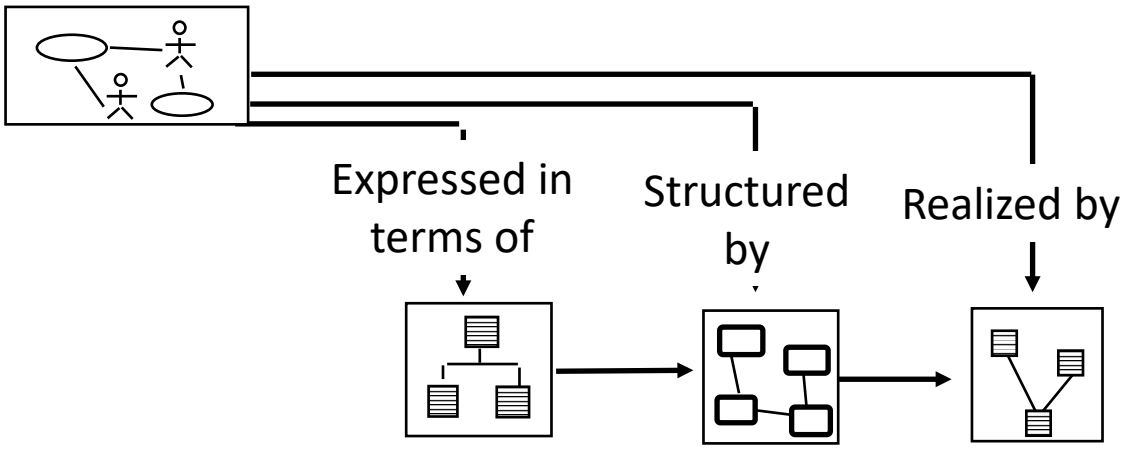
Use Case Model

Application Domain Objects

Sub-systems

Software Lifecycle Activities

Requirements Elicitation	Analysis	System Design	Detailed Design	Implemen-tation	Testing



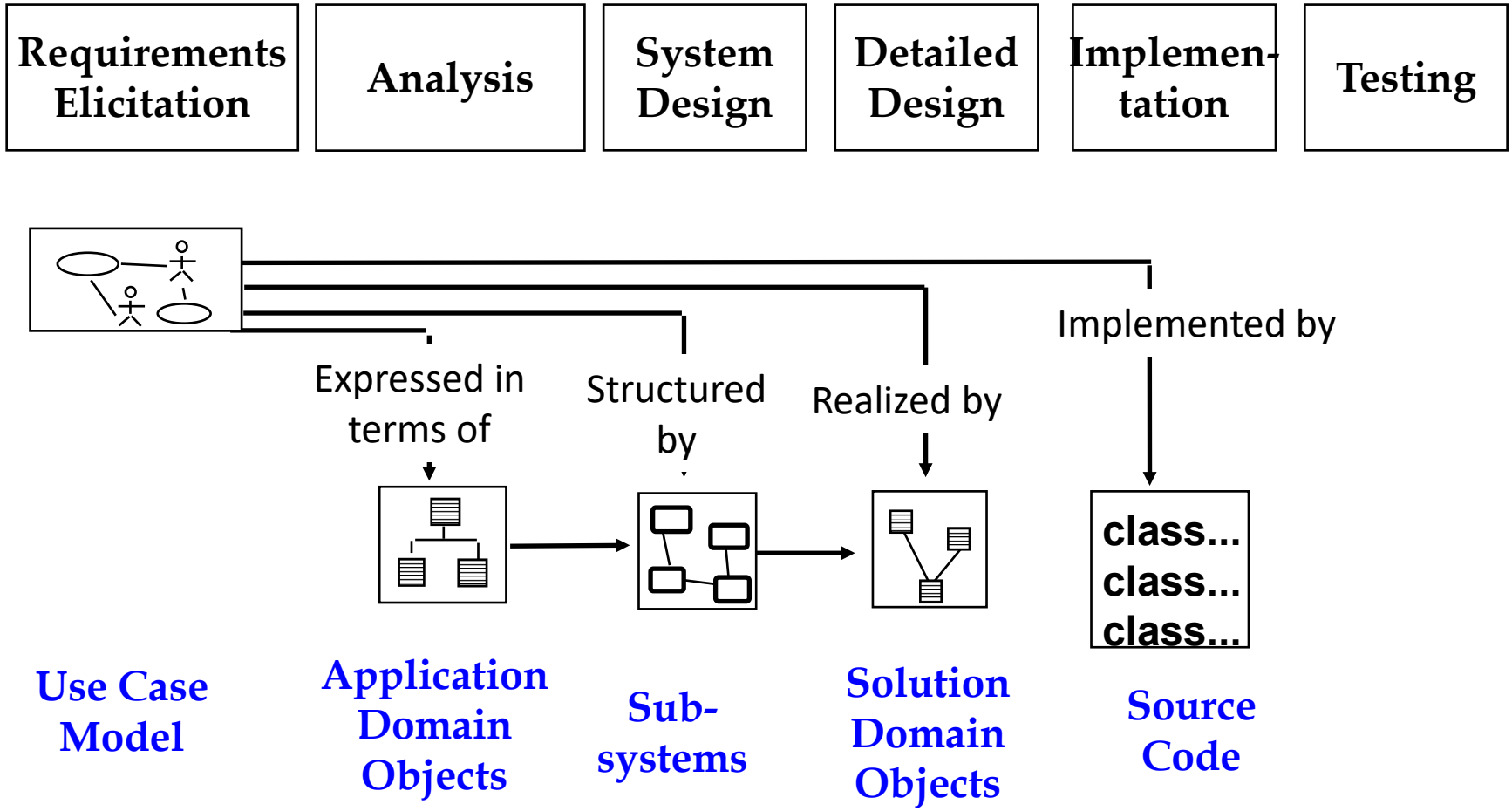
Use Case Model

Application Domain Objects

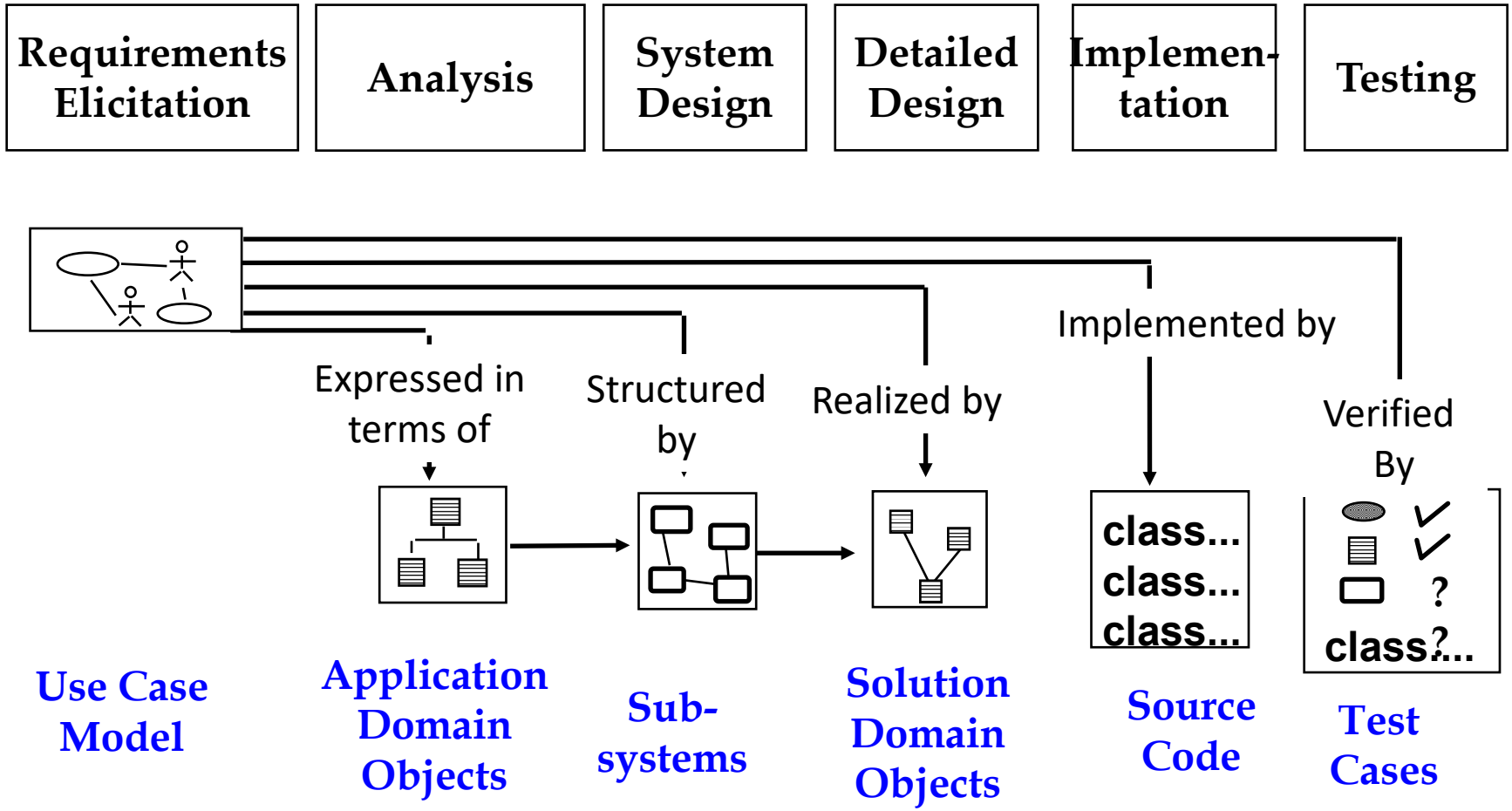
Sub-systems

Solution Domain Objects

Software Lifecycle Activities



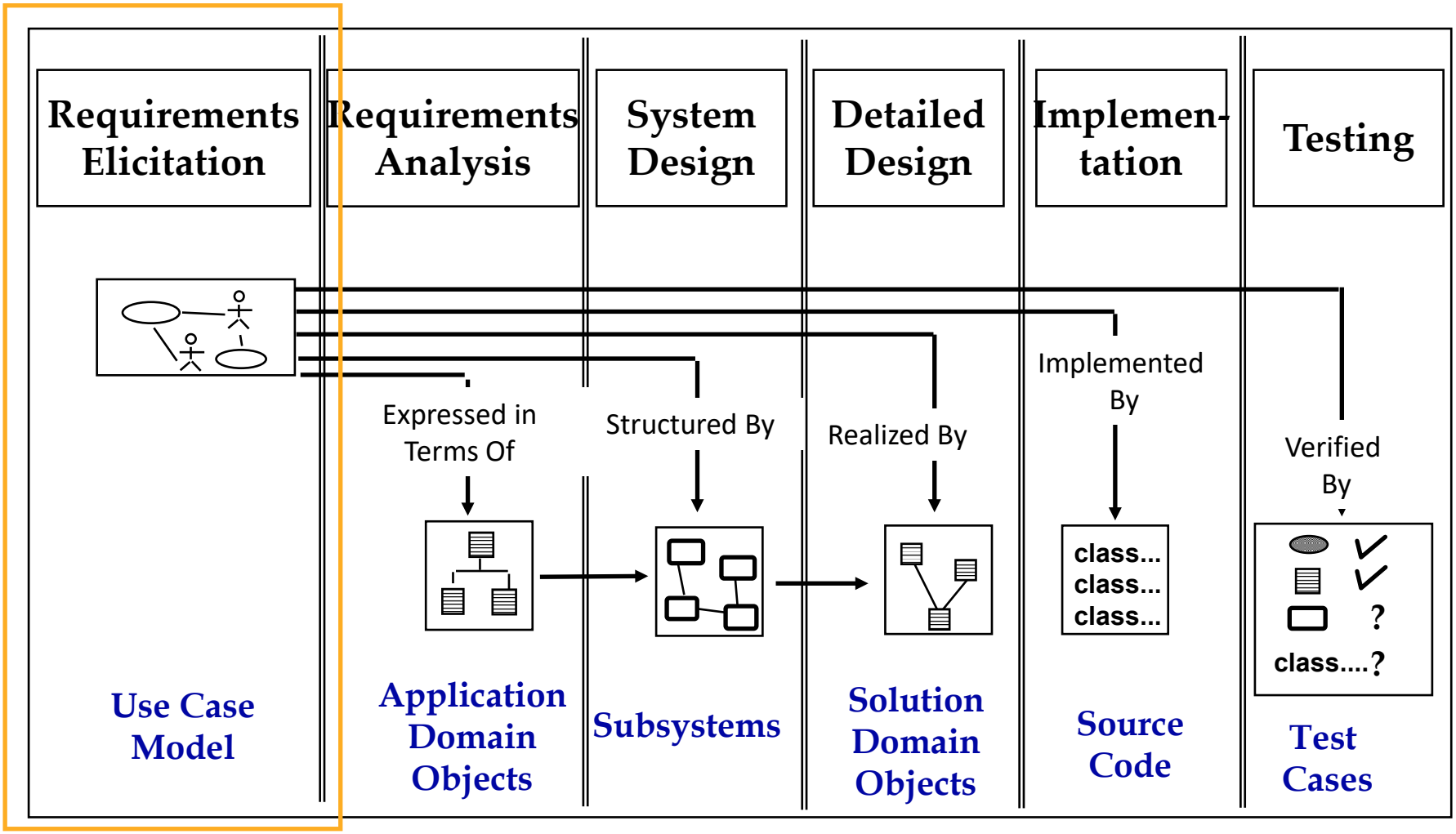
Software Lifecycle Activities



What is the best SDLC?

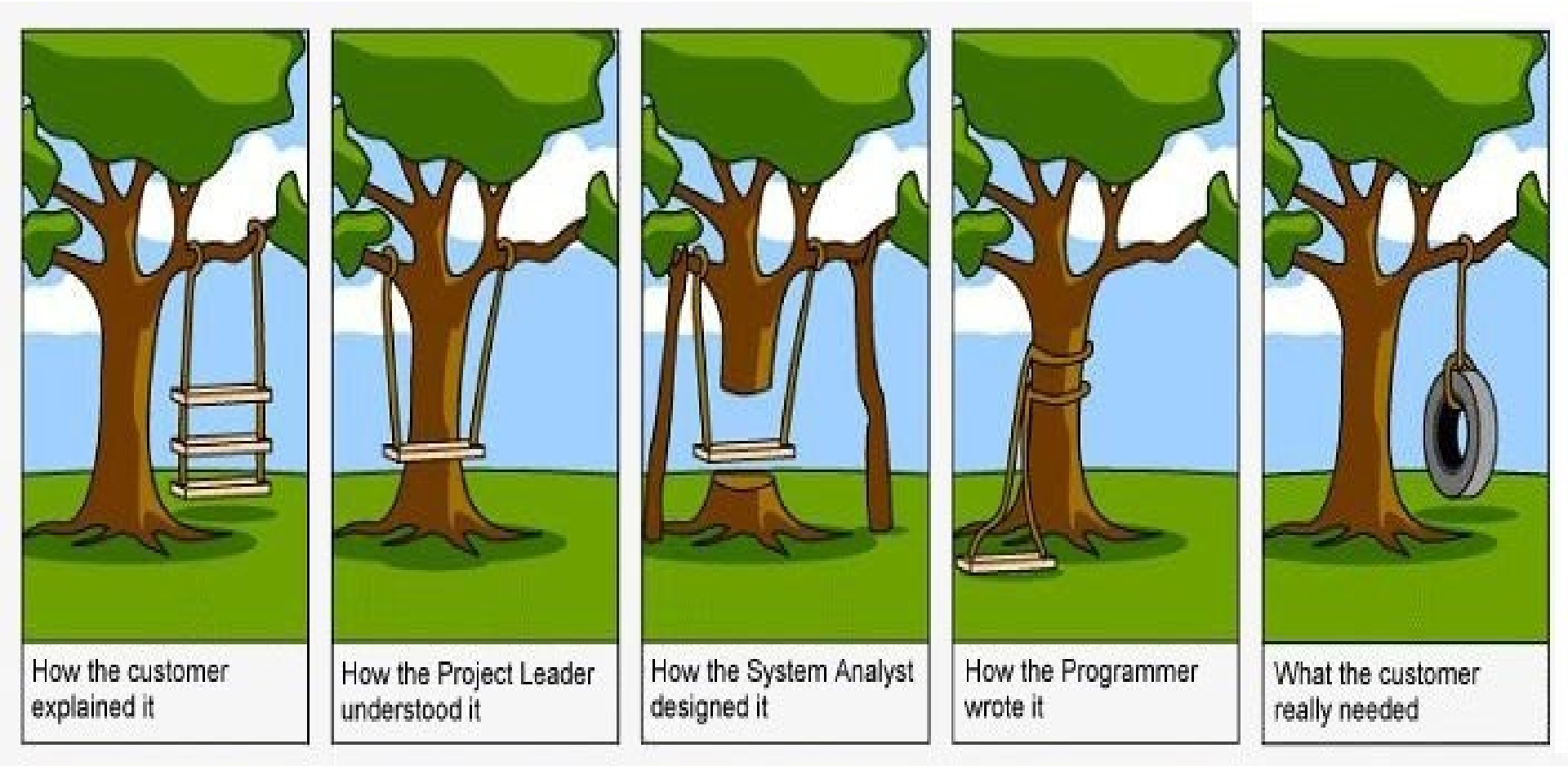
- Answering this question was the topics of the lectures on software processes
- We know that there is a set of tasks/activities/phases that are common to software development.
- Today, we focus on the activity Requirements Engineering
  - Requirements Elicitation
  - Requirements Analysis.

Software Lifecycle Activities



What does the Customer say?

Why is Software Requirements Important?



Meeting Users' Needs

- Software must do what its users want;
- It seems self-evident, but…
- First step should be: find out user needs
- Called requirements engineering:
  - Requirements Elicitation
  - Requirements Analysis
  - Requirements Specification
  - Requirements Validation.

Give Products to Customers Early

- Involve the customer early in the software life cycle
- No matter how hard you try to learn user's needs during the requirements phase, the most effective way to determine real needs is to give users a product and let them play with it.

Determine the Problem before Writing the Requirements

- When faced with what they believe is a problem, most engineers rush to offer a solution
- Before you try to solve a problem, be sure to explore all the alternatives and don’t be blinded by the “obvious” solution.

The Meaning of *Requirements Engineering*

- The process of understanding what's wanted and needed in an application. For example, you may know that you *want* a colonial house in New England, but you may not know that you will probably *need* a basement for it.
- We express requirements in writing to complete our understanding and to create a contract between developer and customer.

Sources of Requirements: People vs. Other

Methods to Elicit Requirements

- Bridging the gap between users and developers:
  - Interviewing customers and domain experts
  - Questionnaires
  - Task Analysis (Observation/Ethnography)
  - Study of documents and software systems
  - JAD, Joint Application Development

Interviews (Commonly Used)
Five Basic Steps

1. Selecting interviewees
   - Often good to get different perspectives
     - Managers, Users, Ideally all key stakeholders
2. Designing interview questions
   - Closed-Ended Questions
     - How many email orders are received per day?
     - How do customers place orders?
   - Open-Ended Questions
     - What do you think about the current order system?
     - What are some of the problems you face on a daily basis?
3. Preparing for the interview
   - Prepare general interview plan: list of question and anticipated answers and follow-ups
   - Inform the interviewee about the reason for interview and the areas of discussion.

Interviews - Five Basic Steps, *cont.*

4. Conducting the interview
   - Record, if consent is given.
   - Appear professional and unbiased
   - Record all information
   - Give interviewee time to ask questions
   - Be sure to thank the interviewee
   - End on time
5. Post-interview follow-up
   - Prepare interview notes
   - Prepare interview report
   - Look for gaps and new questions.

Questionnaires

- In addition to interviews
- Questionnaire is a passive technique
  - advantage – time to consider the answers
  - disadvantage – no possibility to clarify questions and answers

- Close-ended questions
  - Multiple-choice questions
    - additional comments may be allowed
  - Rating questions (Likert scale, e.g. strongly agree, agree, ...)
    - when seeking opinions
  - Ranking questions
    - ranked with numbers, percent, values, etc.

Observation/Ethnography (Task Analysis)

- In addition to interviews (and possibly questionnaires)
- When the user cannot convey sufficient information and/or has only fragmented knowledge
- Three forms
  - Passive
    - no interruption or direct involvement
    - video camera may be used
  - Active
  - Explanatory
    - explaining what is done when observed
- Should be carried for a long time, at different times and at different workloads
- Note that, people tend to behave differently when watched.

Study of documents and software systems

- Provides clues about existing “as-is” system
- Typical documents
  - Organizational documents
    - including procedures, policies, descriptions, plans, charts, internal and external correspondence
  - System forms and reports (if prior computer system exists)
    - record of change requests (defects and enhancements)
- Domain knowledge requirements
  - domain journals and reference books
  - using Internet searches.

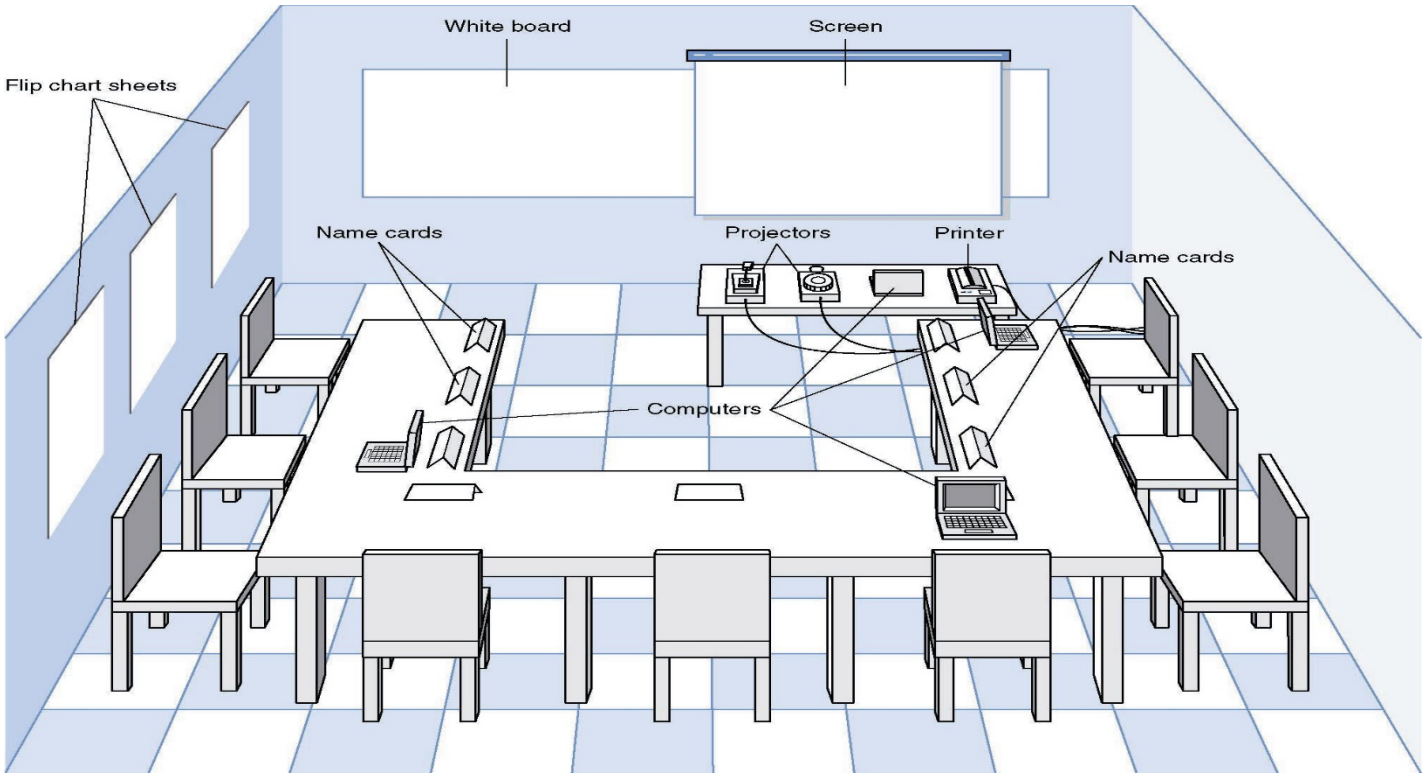
Joint Application Development (JAD)

- It is what the name implies — allows project managers, users, and developers to work together
- Avoids requirements being too specific or too vague.

Joint Application Development (JAD)

- The membership and desirable skills:
  - Leader (communication skills, knowledge of business domain)
  - Scribe (touch typing, software development knowledge, software tool skills)
  - Customers/Users
    - Users
    - Managers
  - Software Developers

Joint Application Design (JAD) Setting



- U-Shaped seating
- Away from distractions
- Useful for any process model
- Whiteboard/flip chart
- Prototyping tools.
- Particularly Agile

The JAD Session

- Tend to last 5 to 10 days over a three-week period
- Prepare questions as with interviews
- Formal agenda
- Leader (Facilitator) activities
  - Keep session on track
  - Help with technical terms
  - Record group input
  - Help resolve issues
- Post-session follow-up.

Techniques to Elicit Requirements

- Bridging the gap between end user and developer:
  - **Questionnaires:** Asking the end user a list of pre-selected questions – not a good idea.
  - **Task Analysis:** Observing end users in their operational environment
  - **Scenarios:** Describe the use of the system as a series of interactions between a concrete end user and the system
  - **Use cases:** Abstractions that describe a class of scenarios.

Scenarios

- Scenario (Italian: that which is pinned to the scenery)
  - A synthetic description of an event or series of actions and events.
  - A textual description of the usage of a system. The description is written from an end user's point of view.
  - A scenario can include text, video, pictures and story boards.
  - It usually also contains details about the work place, social situations and resource constraints.

More Definitions

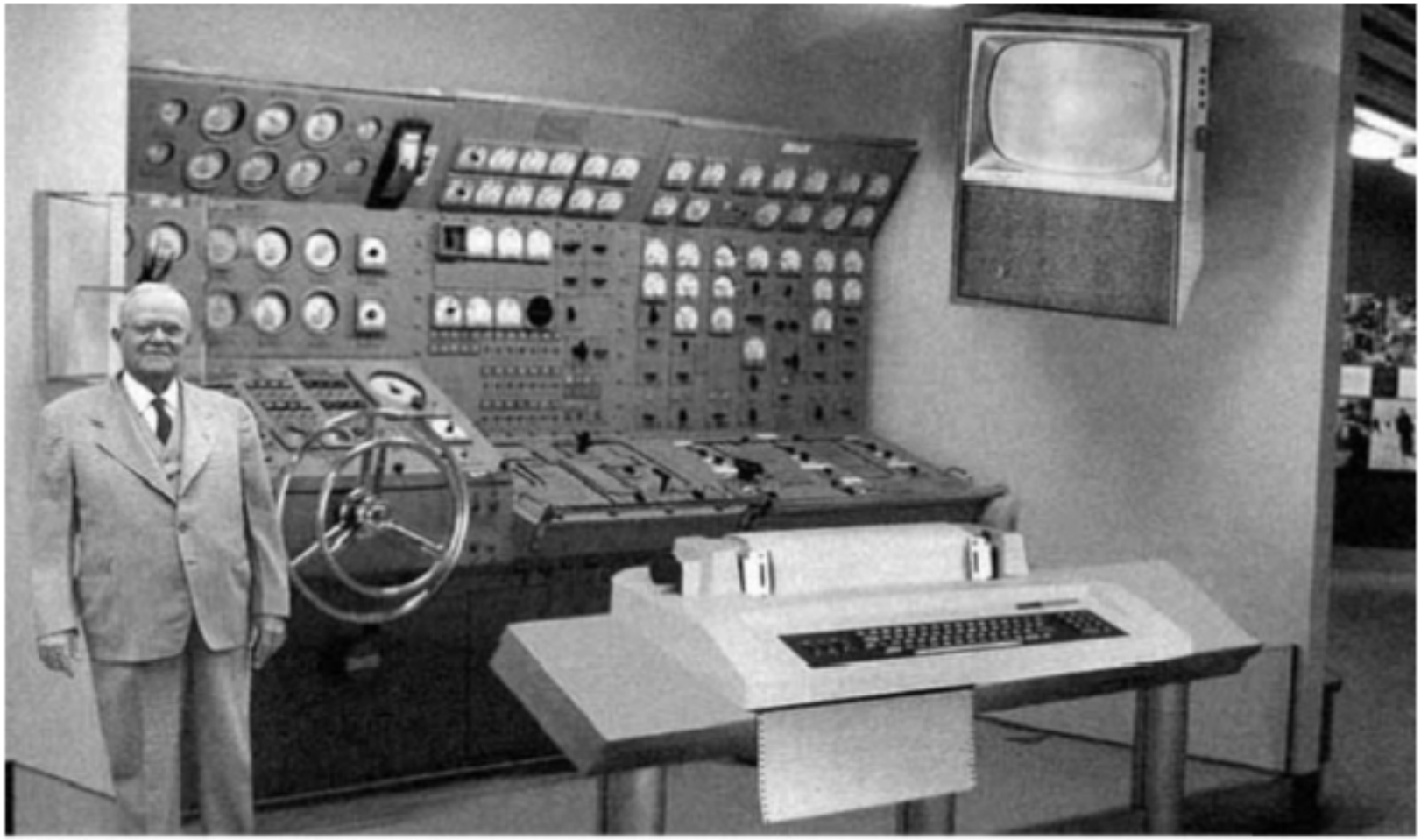
- Scenario: “A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carroll, Scenario-Based Design]

- A concrete, focused, informal description of a single feature of the system used by a single actor. {I like this one!}

Types of Scenarios

- As-is scenario:
  - Describes a current situation. Usually used in re-engineering projects. The user describes the system
    - Example: Description of chess game
- Visionary scenario:
  - Describes a future system. Usually used in greenfield engineering
  - Can often not be done by the user or developer alone
    - Example: Description of an interactive internet-based Tic-Tac-Toe game tournament
    - Example: Description - in the year 1954 - of the Home Computer of the Future.

A Visionary Scenario (1954): The Home Computer in 2004



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

Difficult for Anyone to Predict the Future

- "64k of memory ought to be enough for anyone", Bill Gates of Microsoft, 1981
- "Computer in the future will weight no more than 1.5 tons", Popular Mechanics, 1949
- "I think there is a world market for may be five computers", Thomas Watson, chairman of IBM, 1943
- "But what…. is it good for?", Engineer at IBM commenting on the microchip, 1968
- "There is no reason why anyone would want to have a computer in their home", Ken Olson, president of Digital Equipment Corporation, 1977
- "I have traveled the length and breath of this country and talked with the best people, and I can assure you that data processing is a fad that will not last out the year", Editor of Prentice Hall, 1957

How do we find scenarios?

- Don’t expect the client to be verbal if the system does not exist
  - Client understands problem domain, not the solution domain.
- Don’t wait for information even if the system exists
  - “What is obvious needs to be said”
- Engage in a dialectic approach
  - You help the client to formulate the requirements
  - The client helps you to understand the requirements
  - The requirements evolve while the scenarios are being developed.

Heuristics for finding scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on questions *and* questionnaires alone
- Insist on task observation if the system already exists
  - Ask to speak to the end user, not just to the client
  - Expect resistance and try to overcome it.

Example: Warehouse on fire

- Bob, a police officer, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building into her wearable computer , a brief description of its location (i.e., north-west corner), and an emergency level.
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors
  - Bob, Alice and John

After the scenarios are formulated

- Find all the use cases in the scenario that specify all instances of how to report a fire:
  - Example: “Report Emergency“ in the first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the entry condition
  - Describe the flow of events
  - Describe the exit condition
  - Describe exceptions
  - Describe non-functional requirements.

Use case name	**Report Emergency –** Abstraction of warehouse on fire!
Participating actors	Initiated by FieldOfficer Communicated with Dispatcher
Entry condition	The FieldOfficer is logged into FRIEND System
Flow of events	1. The FieldOfficer activates the "Report Emergency" function of her terminal. 2. FRIEND responds by presenting a form to the FieldOfficer. 3. The FieldOfficer fills the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form. 4. FRIEND receives the form and notifies the Dispatcher. 5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the report. 6. FRIEND displays the acknowledgement and selected response to the FieldOfficer.
Exit condition	• The FieldOfficer receives an acknowledgment and the selected response from the Dispatcher, OR • The FieldOfficer has received an explanation indicating why the transaction could not be processed.
Non-Funtional requirements	• The FieldOfficer's report is acknowledged within 30 seconds. • The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

Requirements Elicitation: Difficulties and Challenges

- Communicate accurately about the domain and the system
  - People with different backgrounds must collaborate to bridge the gap between end users and developers
    - Client and end users have application domain knowledge
    - Developers have solution domain knowledge
- Identify an appropriate system (Defining the system boundary)
- Provide an unambiguous specification
  - (I saw a banner with you in a MRT train).
- Leave out unintended features

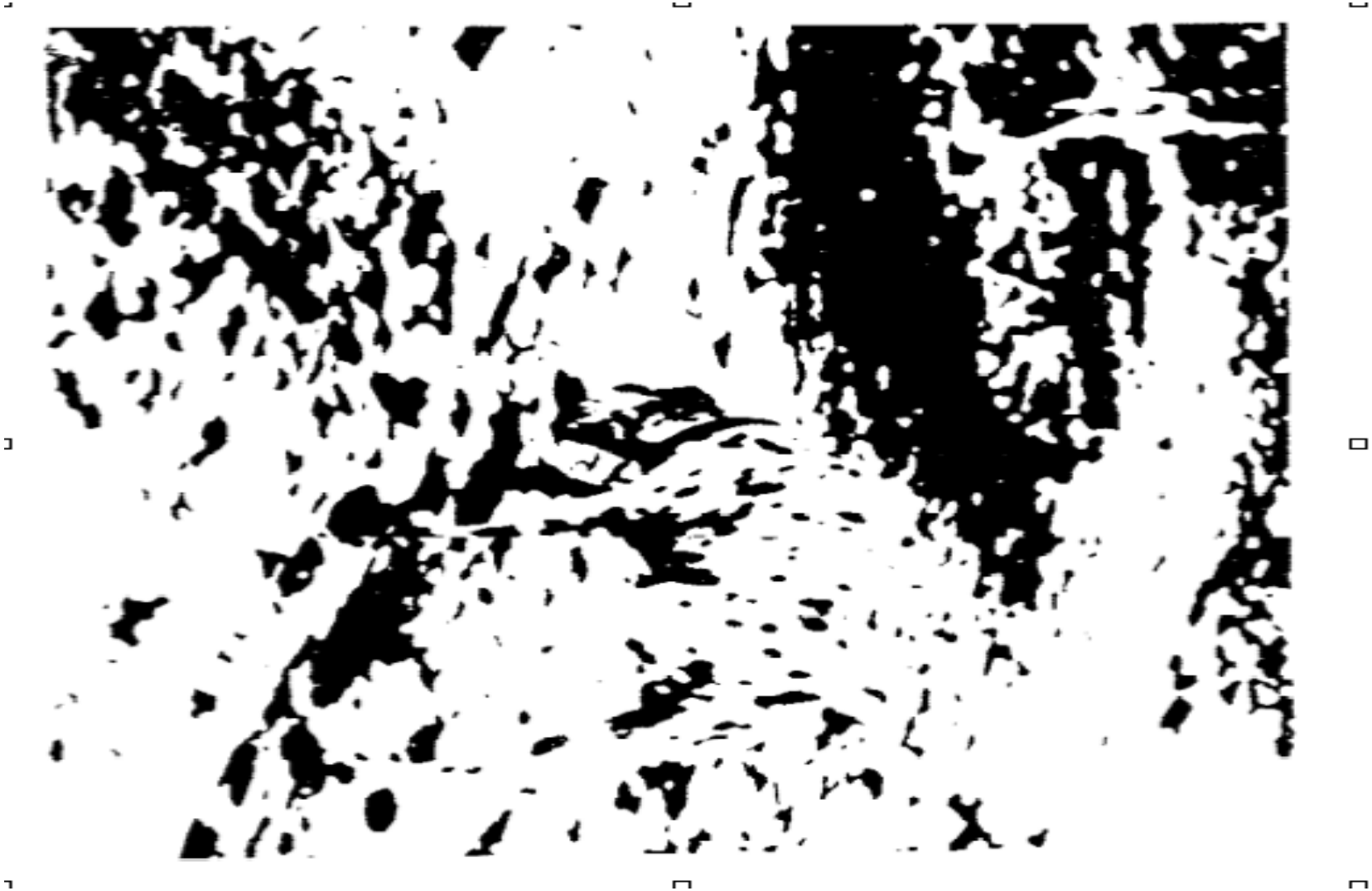
=> 3 Examples.

Defining the System Boundary is difficult

What do you see here?

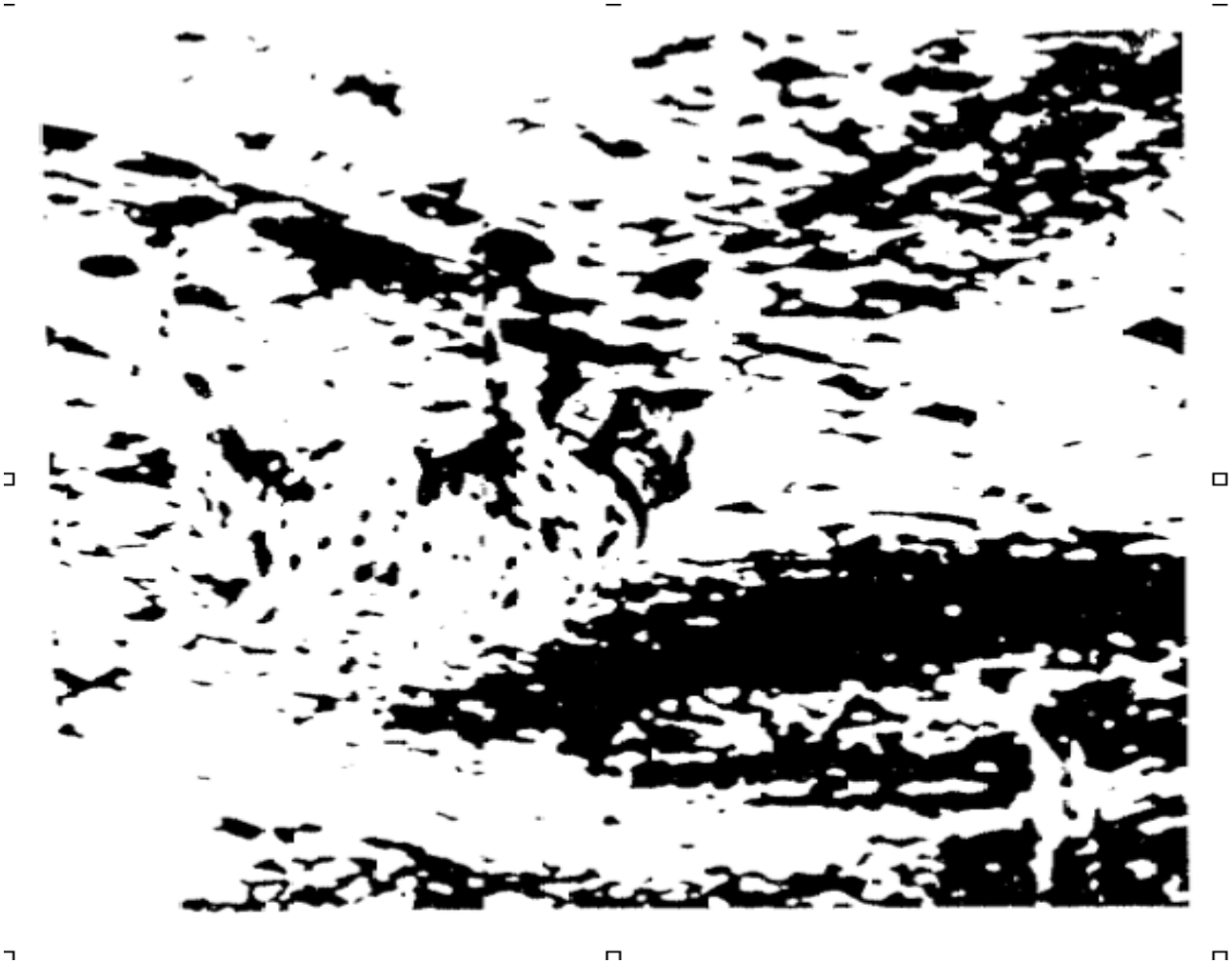
Defining the System Boundary is difficult

What do you see now?



Defining the System Boundary is difficult

What do you see now?



Ambiguity is a **Big** Problem

An Ambiguous Specification

During a laser experiment, a laser beam was directed from earth to a mirror on the Space Shuttle Discovery

The laser beam was supposed to be reflected back towards a mountain top 10,023 feet high

The operator entered the elevation as “10023”

The light beam never hit the mountain top
What was the problem?

The computer interpreted the number in miles...

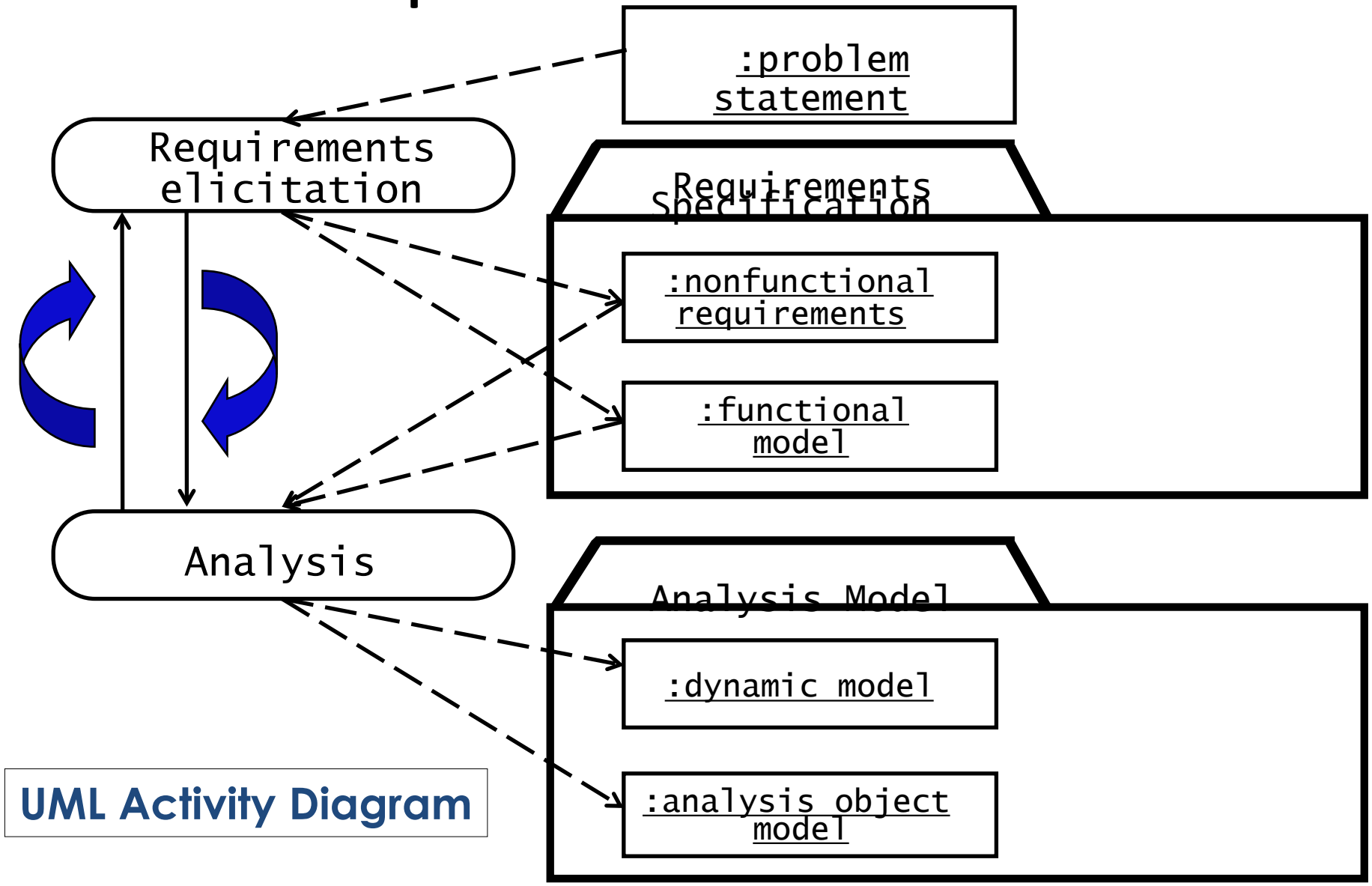
Example of an Unintended Feature

From the News: London underground train leaves station without driver!

What happened?

- A passenger door was stuck and did not close
- The driver left his train to close the passenger door
  - He left the driver door open
  - He relied on the specification that said the train does not move if at least one door is open
- When he shut the passenger door, the train left the station without him
  - The driver door was not treated as a door in the source code!

Requirements Process



UML Activity Diagram

Requirements Specification vs Analysis Model

Both focus on the requirements from the user's view of the system

- The requirements specification uses natural language (derived from the problem statement)
- The analysis model uses a formal or semi-formal notation
  - We use Unified Modeling Language (UML).

Types of Requirements

- Functional requirements
  - Describe the interactions between the system and its environment independent from the implementation

    "An operator must be able to add a new item in the catalogue."
- Non-functional requirements
  - Aspects not directly related to functional services.

    "The response time must be less than 1 second"
  - Constraints Imposed by the client or the environment
    - "The implementation language must be in Java"

Requirements Elicitation Tasks

- During requirements elicitation, software engineers access many different sources of information using some or all the methods we have covered.
- These information will be represented in structured forms by applying the following tasks:
  - Identifying actors: software engineers identify the different types of users the future system will support.
  - Identifying scenarios: software engineers observe users and develop a set of detailed scenarios for typical functionality provided by the future system.
  - Identifying non-functional requirements: developers, users, and clients agree on aspects that are visible to the user, but not directly related to functionality.

Clear Functional Requirements

- R1: The rental application shall enable clerks to check bikes in and out.
- R2: The daily late charge on a bike shall be computed at half the regular two-day rental rate, up to US$20.
- R3: When the “Submit” button is pressed on GUI 37, the GUI shall disappear and GUI 15 shall appear with a superimposed green border, and the name and address fields filled with the customer data.

Vagueness, Ambiguities and Omissions

- R1: The drone, a quad chopper, will be very useful in search and recovery operations, especially in remote areas or in extreme weather conditions.
- R2: It will give high-resolution images.
- R3: It will fly according to a path pre-set by a ground operator, but will be able to avoid obstacles on its own, returning to its original path whenever possible.

Non-Functional Requirements

- **Quality attributes**
  - **Reliability and Availability** (observed faults, average up time)
  - **Performance** (speed, throughput, storage)
  - **Security** (malicious and non-malicious compromise of data or functionality)
  - **Maintainability** (cost to maintain)
  - **Portability** (move to a different operating environment)
- **Constraints** on the application or its development
- **External interfaces** that the application "talks to"
  - **Hardware**
  - **Other software**
  - **Communication with external agents**
- **User interfaces**
- **Error handling.**

Examples of Constraints

- **Platform**
  - e.g., the application must execute on any 4GHz Linux computer
- **Development Media**
  - e.g., the application must be implemented in Java
  - e.g., Rational Rose CASE must be used for the design.

External Interface Requirements

- **Hardware**
  - e.g., “the application must interface with a model 1234 bar code reader and a QR coder reader”
- **Software**
  - e.g., “the application shall use the company’s payroll system to retrieve salary information”
  - e.g., “the application shall use version 1.1 of the Apache server”
- **Communications**
  - e.g., “the application shall communicate with human resources applications via the company intranet”
  - e.g., “the format used to transmit “article expected” messages to cooperating shipping companies shall use XML standard 183.34 published at http://…”

A Non-Functional Requirement

R10: The bike rental system shall not crash more than 5 times per year.

R11: The system shall recover from each crash ASAP. Uhhmmm!

Or better….

R11: The system shall recover from each crash in 5 minute or less to avoid down time.

Functional vs. Non-Functional Requirements

Functional Requirements

- Describe user tasks that the system needs to support
- Phrased as actions
  - "Advertise a new bike"
  - "Notify a customer in the waiting list"

Non-functional Requirements

- Describe properties of the system or the domain
- Phrased as constraints or negative assertions
  - "All user inputs should be acknowledged within 1 second"
  - "A system crash should not result in data loss".

IEEE System Requirements Specification

1. Introduction

1.1. Purpose

1.2. Scope

1.3. Definitions, acronyms & abbreviations

1.4. References

1.5. Overview

2. Overall description

2.1. Product perspective

2.1.1. System interfaces

2.1.2. User interfaces

2.1.3. Hardware interfaces

2.1.4. Software interfaces

2.1.5. Communications interfaces

2.1.6. Memory constraints

2.1.7. Operations

2.1.8. Site adaptation requirements

2.2. Product functions

2.3. User characteristics

2.4. Constraints

2.5. Assumptions and dependencies

2.6 Apportioning of requirements

3. Specific/Detailed requirements

-- see Figure 9

Appendixes

IEEE SRS - Detailed Requirements

3.1 External interfaces
3.2 Functional requirements
-- organized by feature, object, user class, etc.
3.3 Performance requirements
3.4 Logical database requirements
3.5 Design constraints
3.5.1 Standards compliance
3.6 Software system attributes
3.6.1 Reliability
3.6.2 Availability
3.6.3 Security
3.6.4 Maintainability
3.6.5 Portability
3.7 Organizing the specific requirements
3.7.1 System mode -- or
3.7.2 User class -- or
3.7.3 Objects (see right) -- or
3.7.4 Feature -- or
3.7.5 Stimulus -- or
3.7.6 Response -- or
3.7.7 Functional hierarchy -- or
3.8 Additional comments

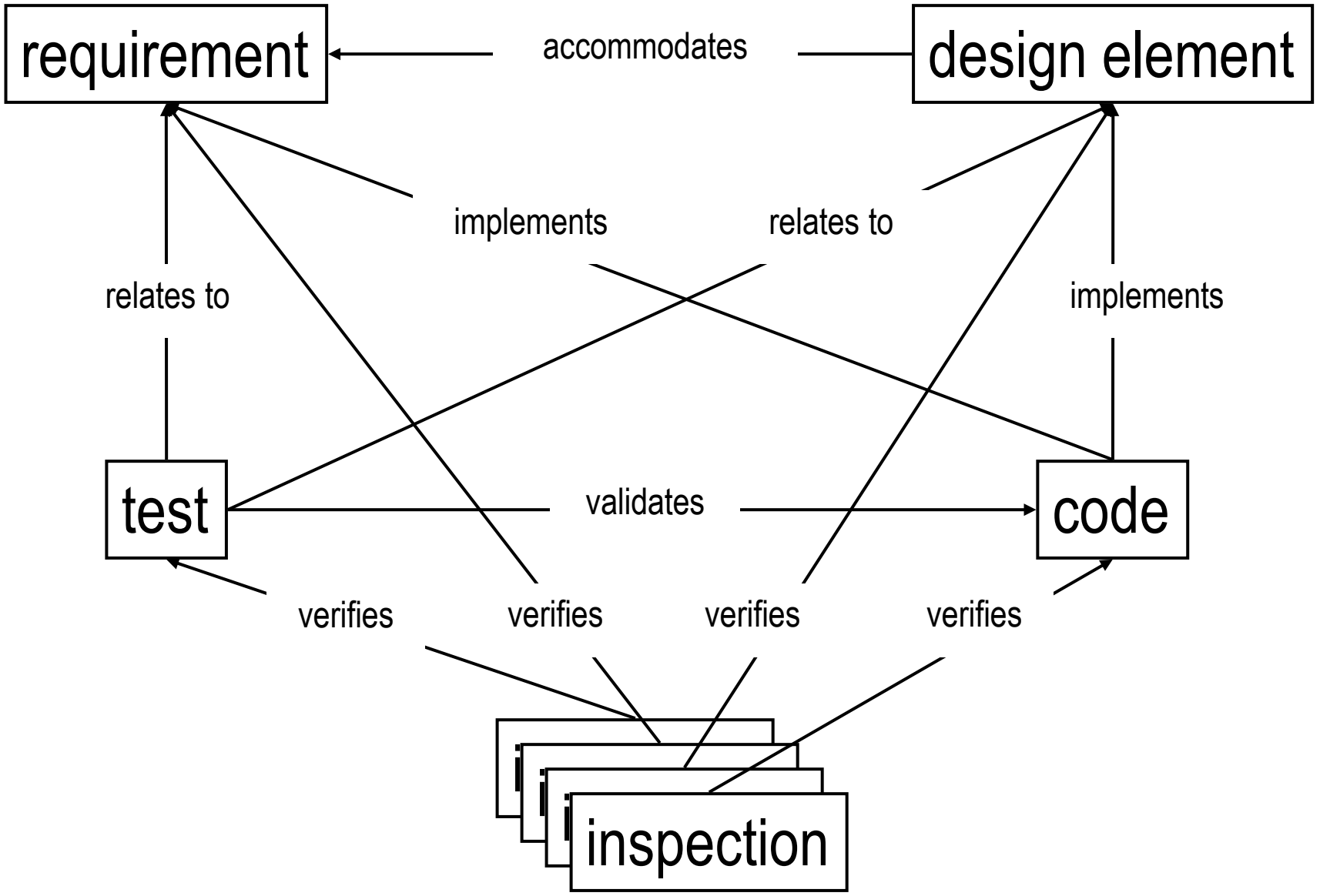
Prioritizing Detailed Requirements

[essential] *Every game character has the same set of qualities.*

[desirable] *Each area has a set of* preferred *qualities.*

[optional] *The player's character shall age with every encounter. The age rate can be set at setup time. Its default is one year per encounter.*

Traceability



A Change in a Requirement Causes Changes In Other Artifacts

Artifact	Original version	Revised version
Requirement	The title of a movie shall consist of between 1 and 15 English characters.	The title of a movie shall consist of between 1 and 15 characters, available in English, French and Chinese.
Design element		
Code	class Movie { String title …. }	class Movie { Title title …. } class Title…
Inspection report	Inspection # 672: 4 defects; Follow-up inspection #684.	Inspection # 935: 1 defect; no follow-up inspection required.
Test report	Test # 8920 …	Test # 15084 …

Tools for Requirements Management

DOORS (Telelogic)

- Multi-platform requirements management tool, for teams working in the same geographical location. DOORS XT for distributed teams

RequisitePro (IBM/Rational)

- Integration with MS Word
- Project-to-project comparisons via XML baselines

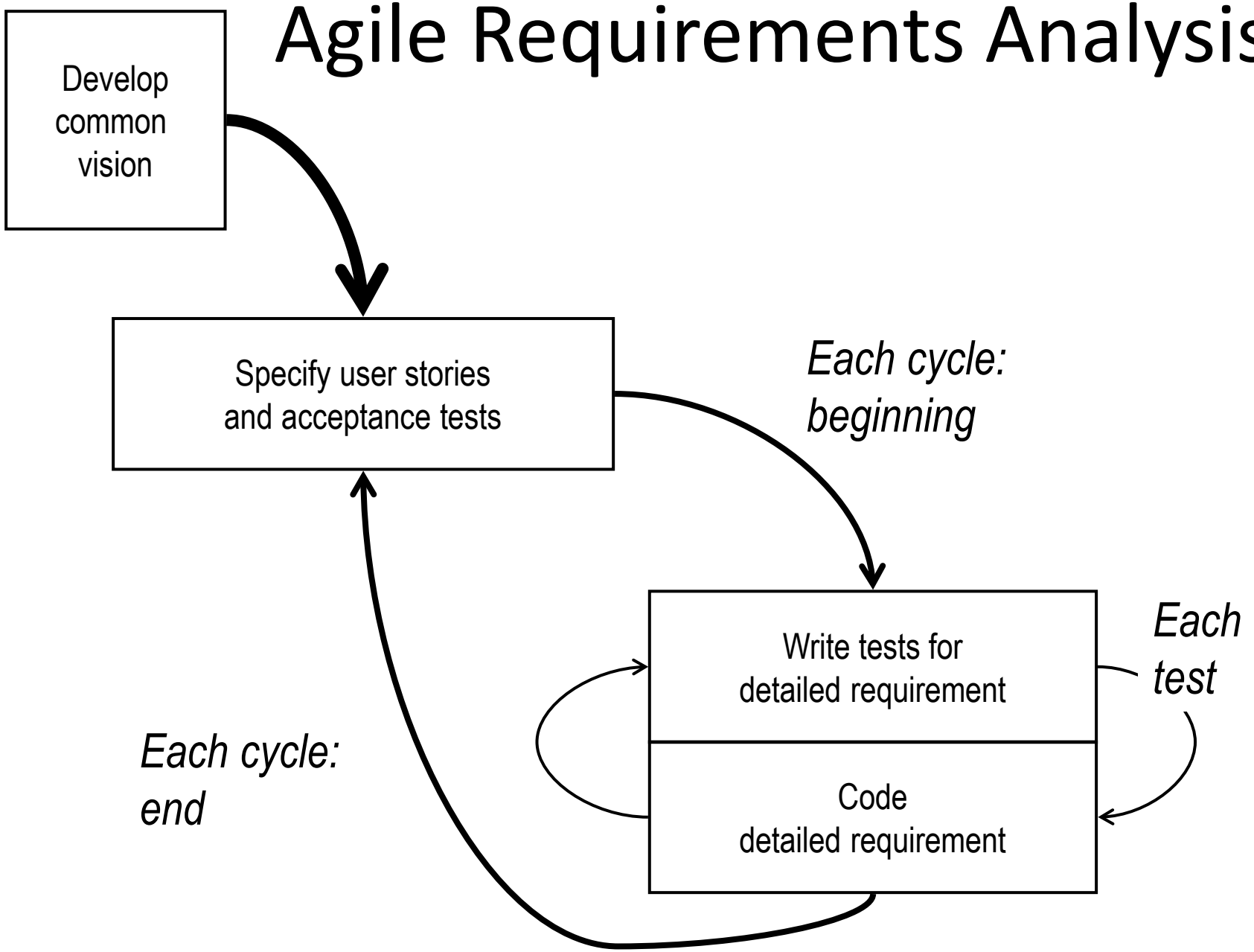
RD-Link (http://www.ring-zero.com)

- Provides traceability between RequisitePro & Telelogic DOORS

Unicase (http://unicase.org)

- Research tool for the collaborative development of system models
- Participants can be geographically distributed.

Agile Requirements Analysis



Recap: Why is software development difficult?

- The problem is usually ill-defined and ambiguous
- The problem domain (also called application domain) is large
- The solution domain is complex
- The development process is difficult to manage
- The requirements are usually unclear and changing when they become clearer
- Software offers extreme flexibility.

Assignment-2

- To be done individually, by Feb. 9th
- Goal: SRS for an e-Grocery e-commerce platform
- A description of an initial SRS is given: YSC3232-04-SRS-eGrocery-Assign-2
- Three SRS template samples
- An e-Grocery Algorithm
- Use YSC3232-04-SRS-IEEE-Template.