THE UNIVERSITY OF WESTERN AUSTRALIA

Requirements Engineering: Specification & Validation

Software Requirements and Design
CITS4401

Week 5

Key ideas (this week)

1. Testing requirements

    Convince me!

2. Prototyping requirements

    Show me!

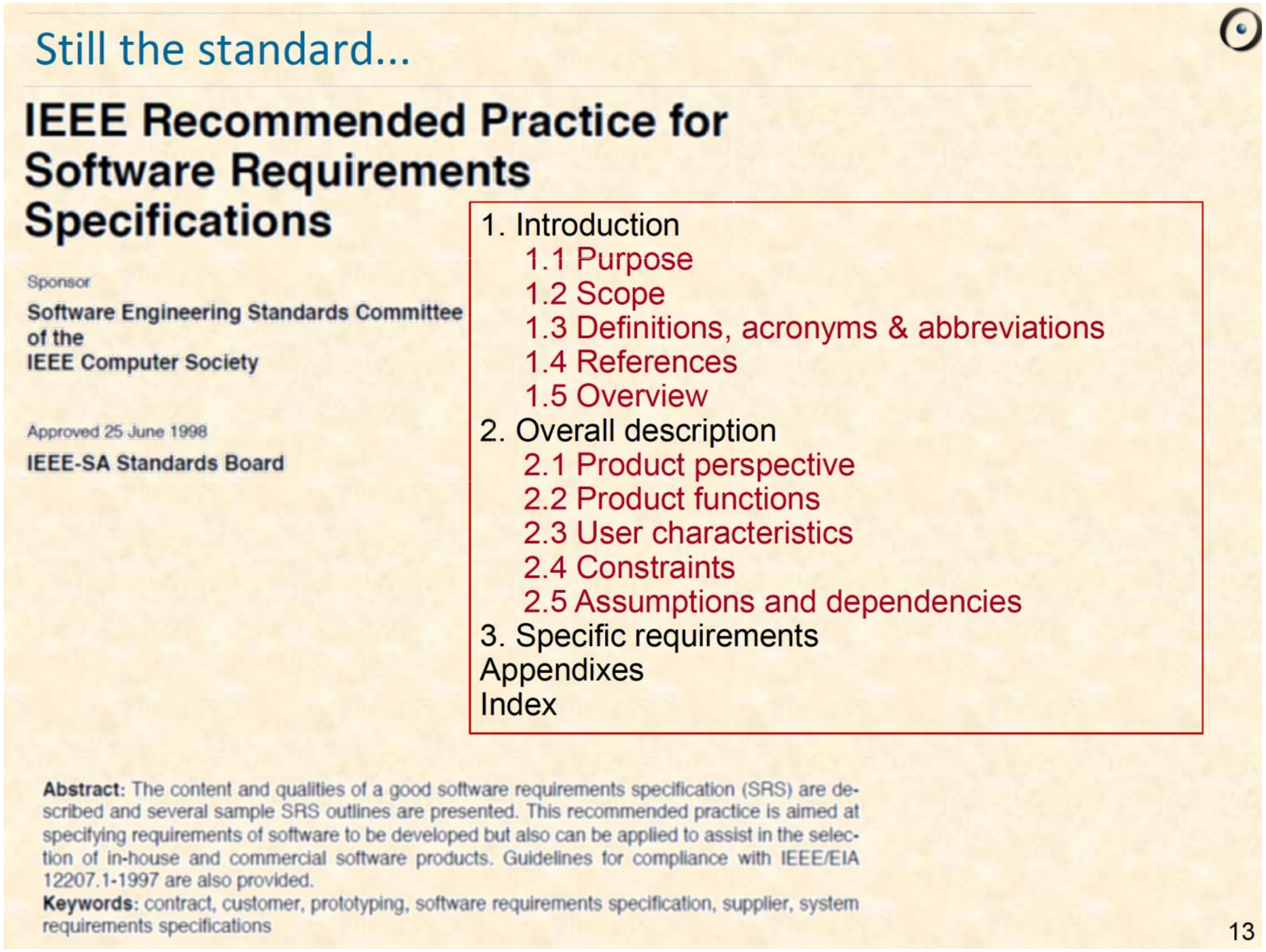
3. Requirements Specification Document

    Tell me!

4. Managing requirements

    Maintain me!

IEEE Standard



Still the standard...

IEEE Recommended Practice for Software Requirements Specifications

Sponsor

Software Engineering Standards Committee of the IEEE Computer Society

Approved 25 June 1998

IEEE-SA Standards Board

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
    - 2.2 Product functions
    - 2.3 User characteristics
    - 2.4 Constraints
    - 2.5 Assumptions and dependencies
3. Specific requirements

Appendixes

Index

Abstract: The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.

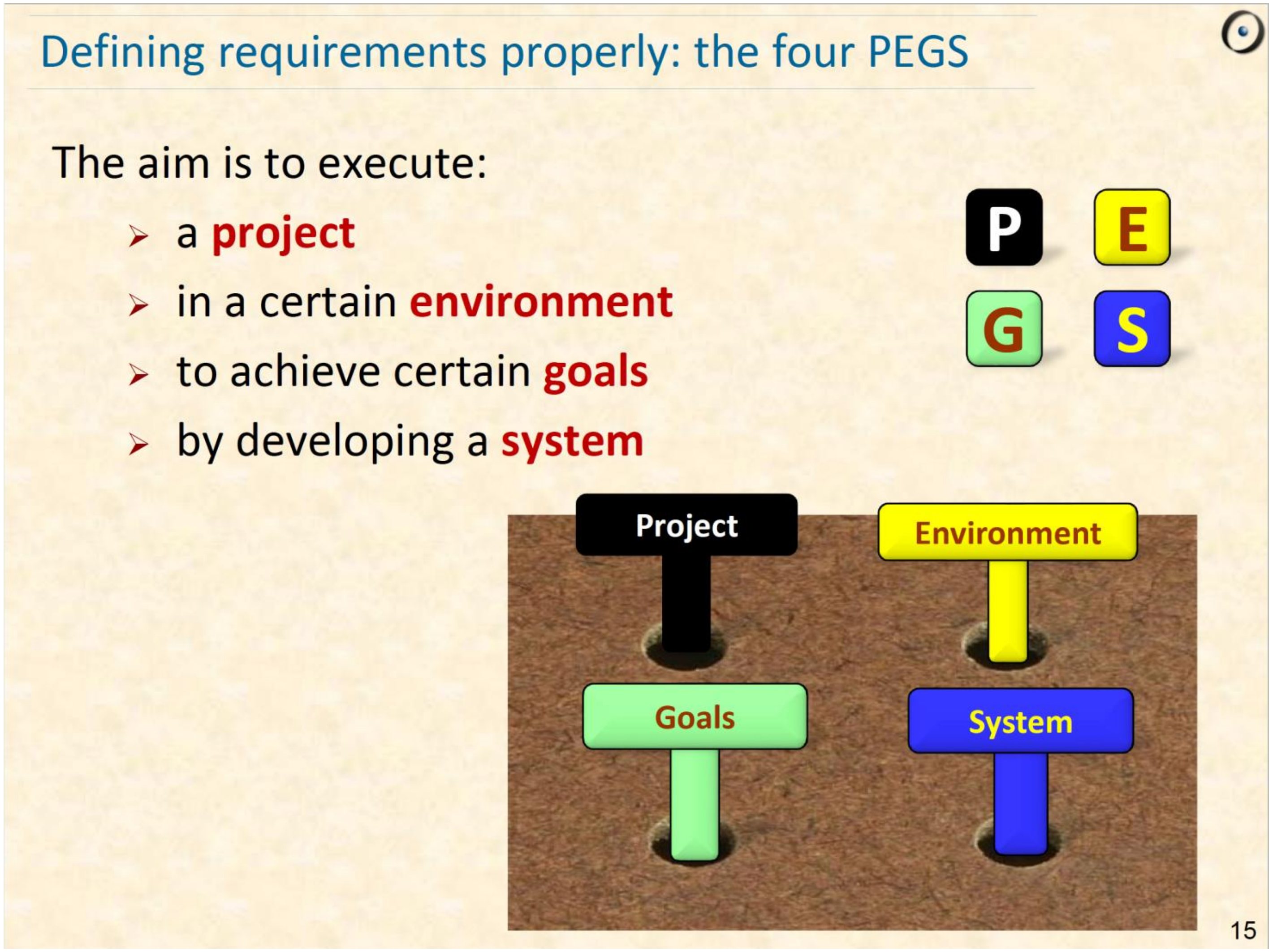
Keywords: contract, customer, prototyping, software requirements specification, supplier, system requirements specifications

13

Requirements Specification Document Template (IEEE)

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Scope of the system
   - 1.3 Objectives and success criteria of the project
   - 1.4 Definitions, acronyms, and abbreviations
   - 1.5 References
   - 1.6 Overview
2. Current system
3. Proposed system
   - 3.1 Overview
   - 3.2 Functional requirements
   - 3.3 Non-functional requirements
     - 3.3.1 Usability
     - 3.3.2 Reliability
     - 3.3.3 Performance
     - 3.3.4 Supportability
     - 3.3.5 Implementation
     - 3.3.6 Interface
     - 3.3.7 Packaging
     - 3.3.8 Legal
   - 3.4 System models
     - 3.4.1 Scenarios
     - 3.4.2 UML use case models
     - 3.4.3 Object model
     - 3.4.4 Dynamic models
     - 3.4.5 User interface—navigation paths and screen mock-ups
4. Glossary

PEGS



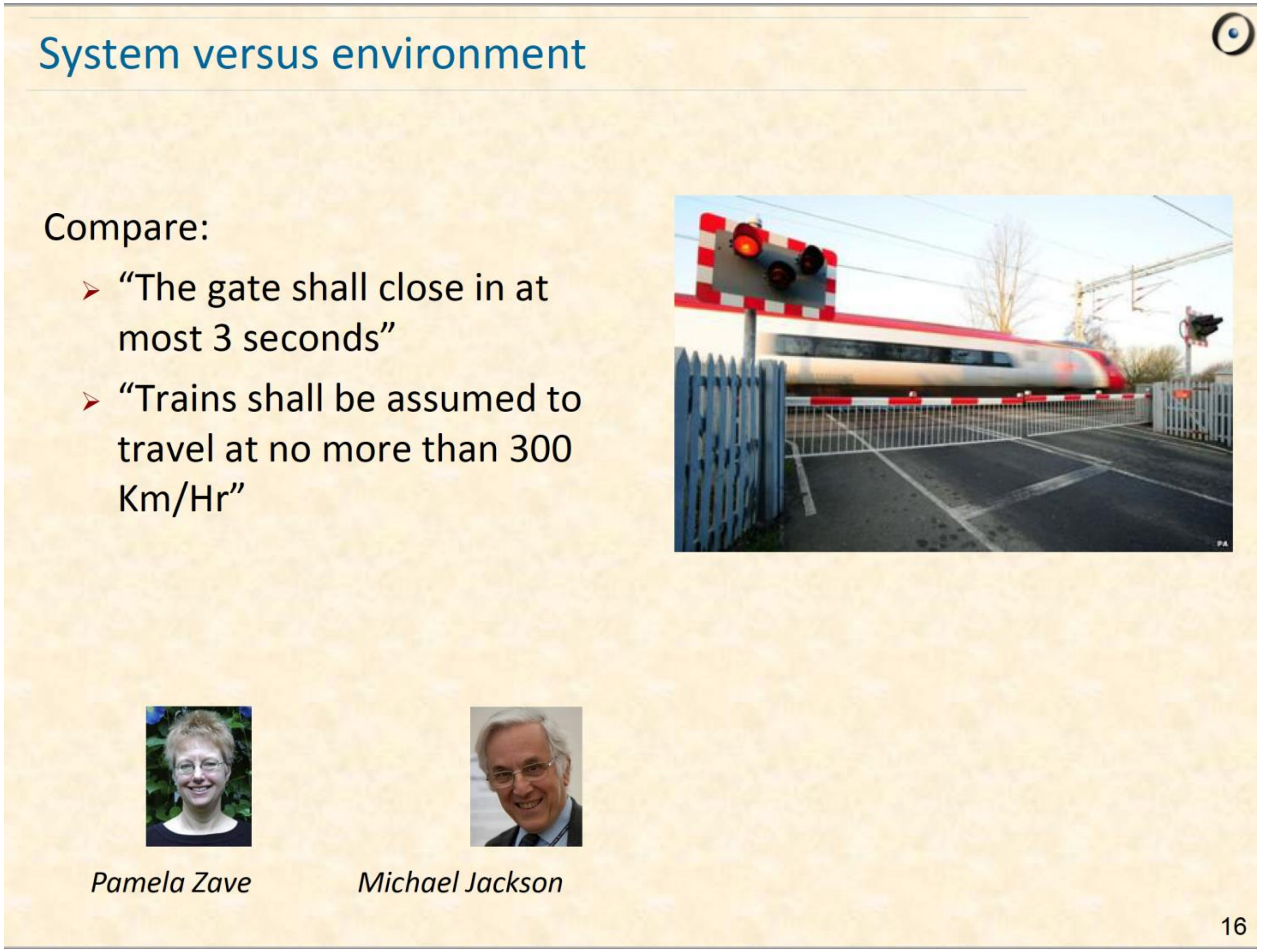
Defining requirements properly: the four PEGS

The aim is to execute:

- a **project**
- in a certain **environment**
- to achieve certain **goals**
- by developing a **system**

P
E
G
S
Project
Environment
Goals
System
15

System vs Environment



System versus environment

Compare:

- "The gate shall close in at most 3 seconds"
- "Trains shall be assumed to travel at no more than 300 Km/Hr"

Pamela Zave

Michael Jackson

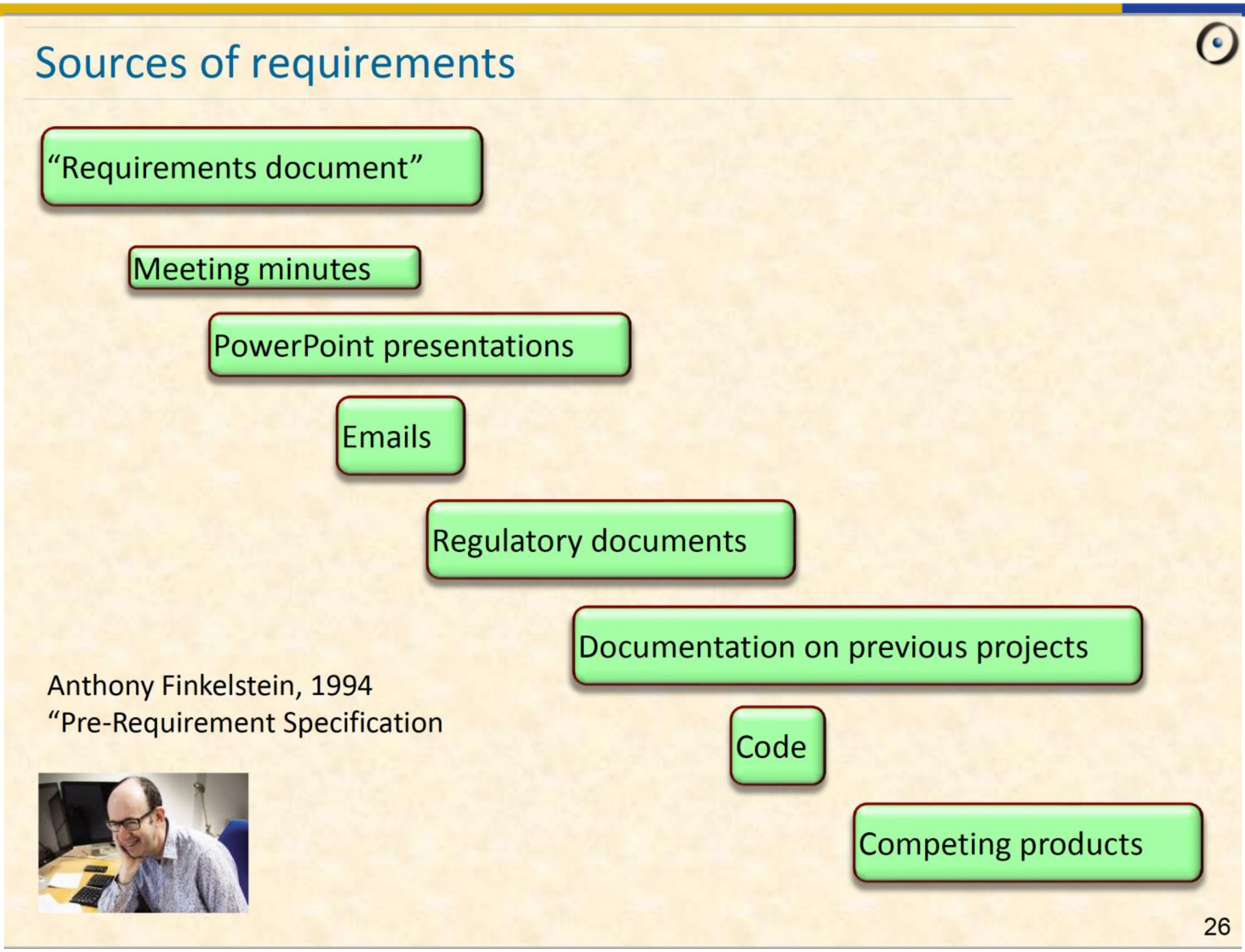
16

Project vs Goals

- Project Requirements concern how the SW development will be carried out
- Example: The system will be implemented using React and Express
- Example: The system will be delivered in 3 sprints of 1 month each with final deliverable on 31 May 2021

- Goals Requirements capture the business goals of the system – why we are building it
- Example: The system will enable admin staff to process 4 student applications per hour instead of 2 hours per application now

Requirements Sources



Sources of requirements

"Requirements document"

Meeting minutes

PowerPoint presentations

Emails

Regulatory documents

Documentation on previous projects

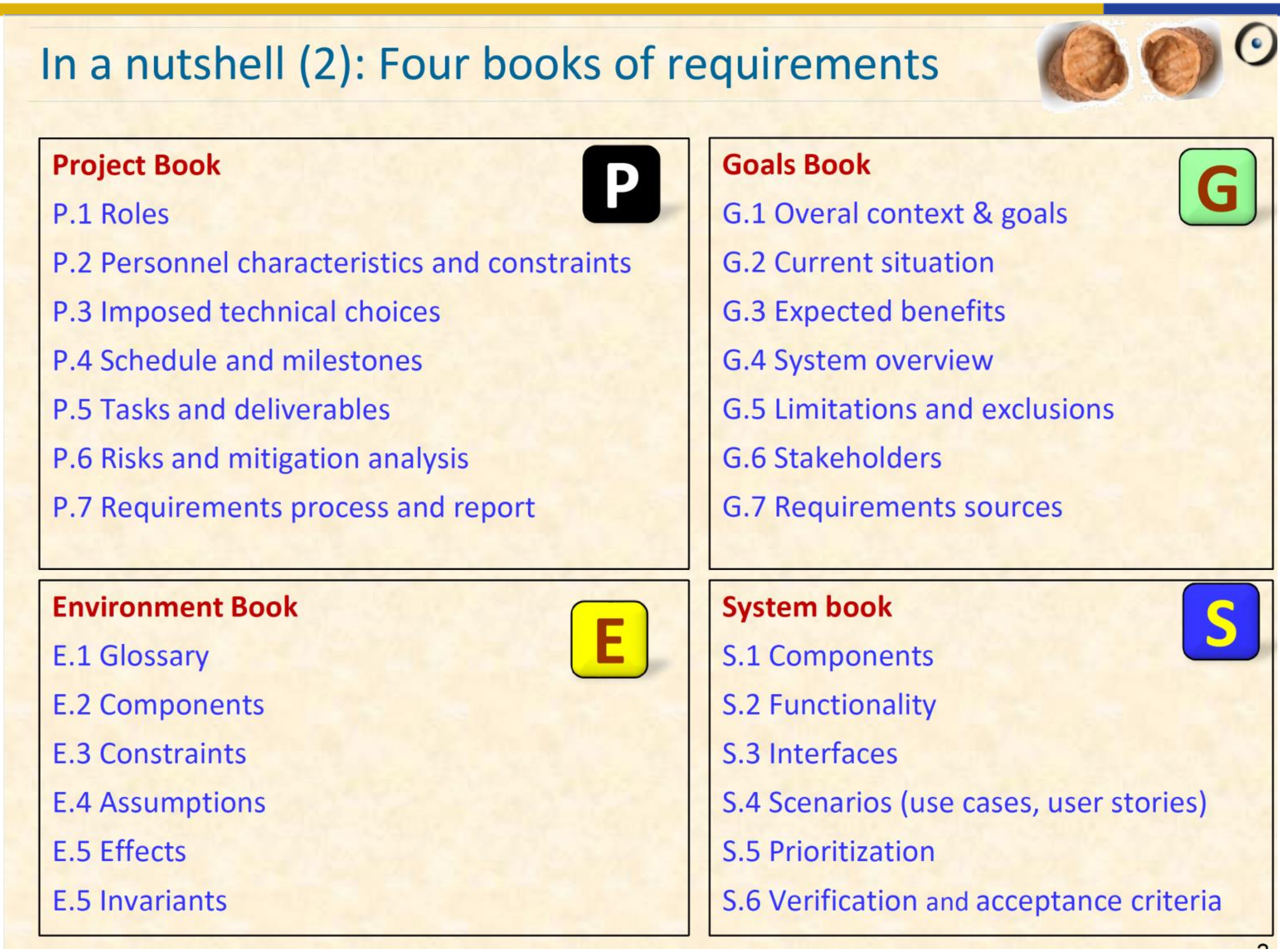
Code

Competing products

Anthony Finkelstein, 1994
"Pre-Requirement Specification

26

PEGS requirements repository



In a nutshell (2): Four books of requirements

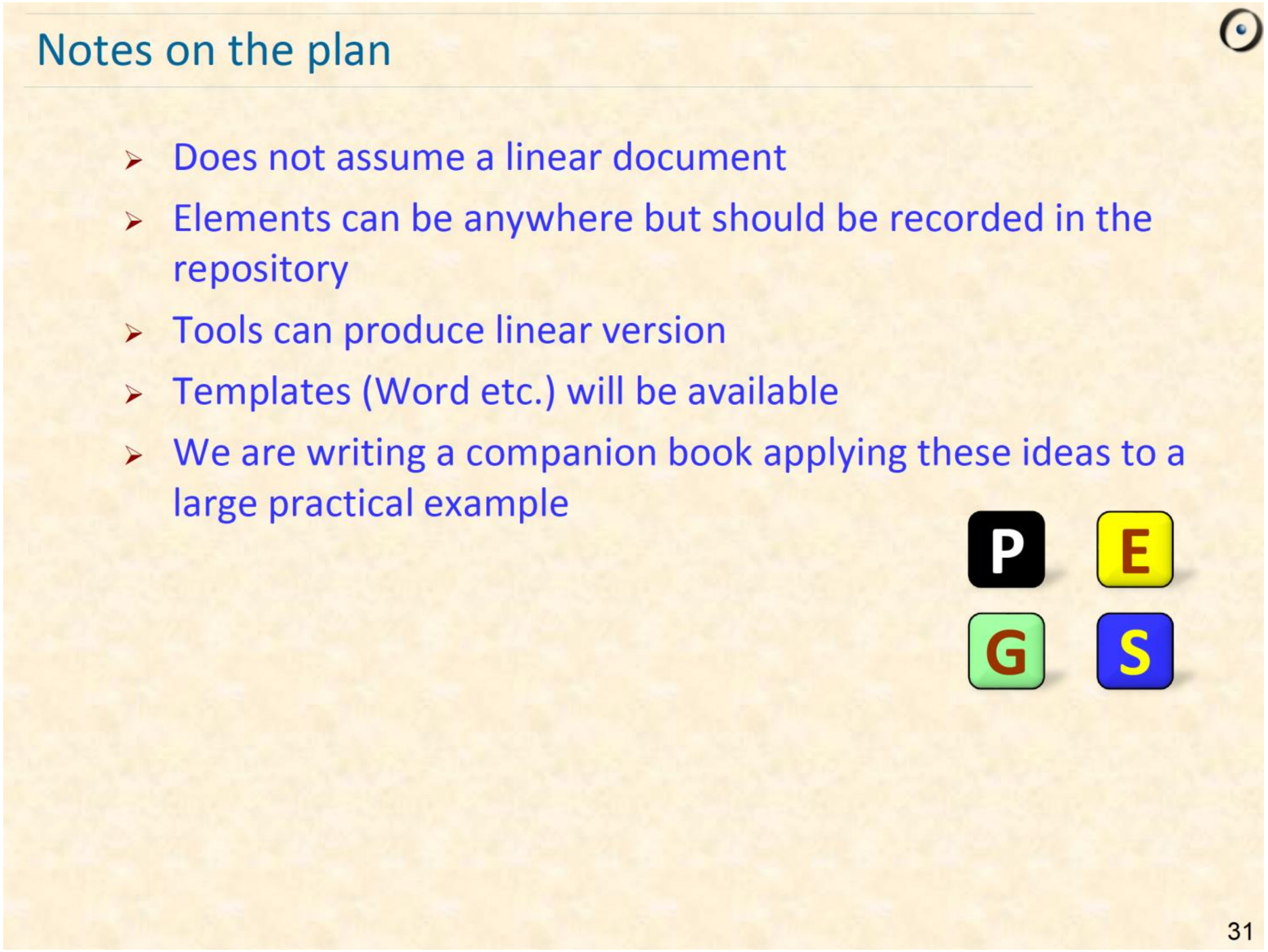
Project Book
- P.1 Roles
- P.2 Personnel characteristics and constraints
- P.3 Imposed technical choices
- P.4 Schedule and milestones
- P.5 Tasks and deliverables
- P.6 Risks and mitigation analysis
- P.7 Requirements process and report

Goals Book
- G.1 Overal context & goals
- G.2 Current situation
- G.3 Expected benefits
- G.4 System overview
- G.5 Limitations and exclusions
- G.6 Stakeholders
- G.7 Requirements sources

Environment Book
- E.1 Glossary
- E.2 Components
- E.3 Constraints
- E.4 Assumptions
- E.5 Effects
- E.5 Invariants

System book
- S.1 Components
- S.2 Functionality
- S.3 Interfaces
- S.4 Scenarios (use cases, user stories)
- S.5 Prioritization
- S.6 Verification and acceptance criteria

PEGS books



Notes on the plan

- Does not assume a linear document
- Elements can be anywhere but should be recorded in the repository
- Tools can produce linear version
- Templates (Word etc.) will be available
- We are writing a companion book applying these ideas to a large practical example

Attributes of a Good Requirements Specification

- Concise
- Complete
- Unambiguous
- Testable
- Consistent
- Feasible
- Modifiable
- Traceable

- Specifies external system behaviour only
- Specifies constraints on the implementation
- Easy to change
- Reference Tool for maintainers
- Records forethought about the life cycle
- Characterises acceptable responses to undesired events

More is not usually better

Aim to be concise!

YOUR REQUIREMENTS DOCUMENT IS THE BIGGEST I'VE EVER SEEN.
IT'S TOO BIG TO READ, BUT I CAN GUESS FROM ITS WEIGHT WHAT MUST BE IN THERE.
YOU KNOW IT'S A MULTI-USER, GLOBAL SYSTEM, RIGHT?
NO, I'M NOT GETTING THAT.
scottadams@aol.com
www.dilbert.com
8/9/99 © 1999 United Feature Syndicate, Inc.

Dilbert
© Scott Adams
At http://dilbert.com/strip/1999-08-09/
(Educational/Classroom usage permission is granted by Universal Uclick. All Rights Reserved)

Requirements: Analysis vs Validation

- **Analysis** works with raw requirements as elicited from the system stakeholders
  - "Have we got the right requirements" is the key question to be answered at this stage

- **Validation** works with a final draft of the requirements document i.e. with negotiated and agreed requirements
  - "Have we got the requirements right" is the key question to be answered at this stage

Summary

- **Requirements Specification**
  - Document or Repository
  - Basis for client-developer contract
- Desirable attributes
  - Concise, complete, consistent, unambiguous, testable, modifiable, traceable, feasible
- Choose an appropriate method for your project
  - Agile: User stories
  - In between: PEGS
  - Waterfall: IEEE Standard

UML Review – Requirement Analysis Framework

- **Requirement analysis generates an analysis model with three parts:**

  - Functional model: use cases & scenarios

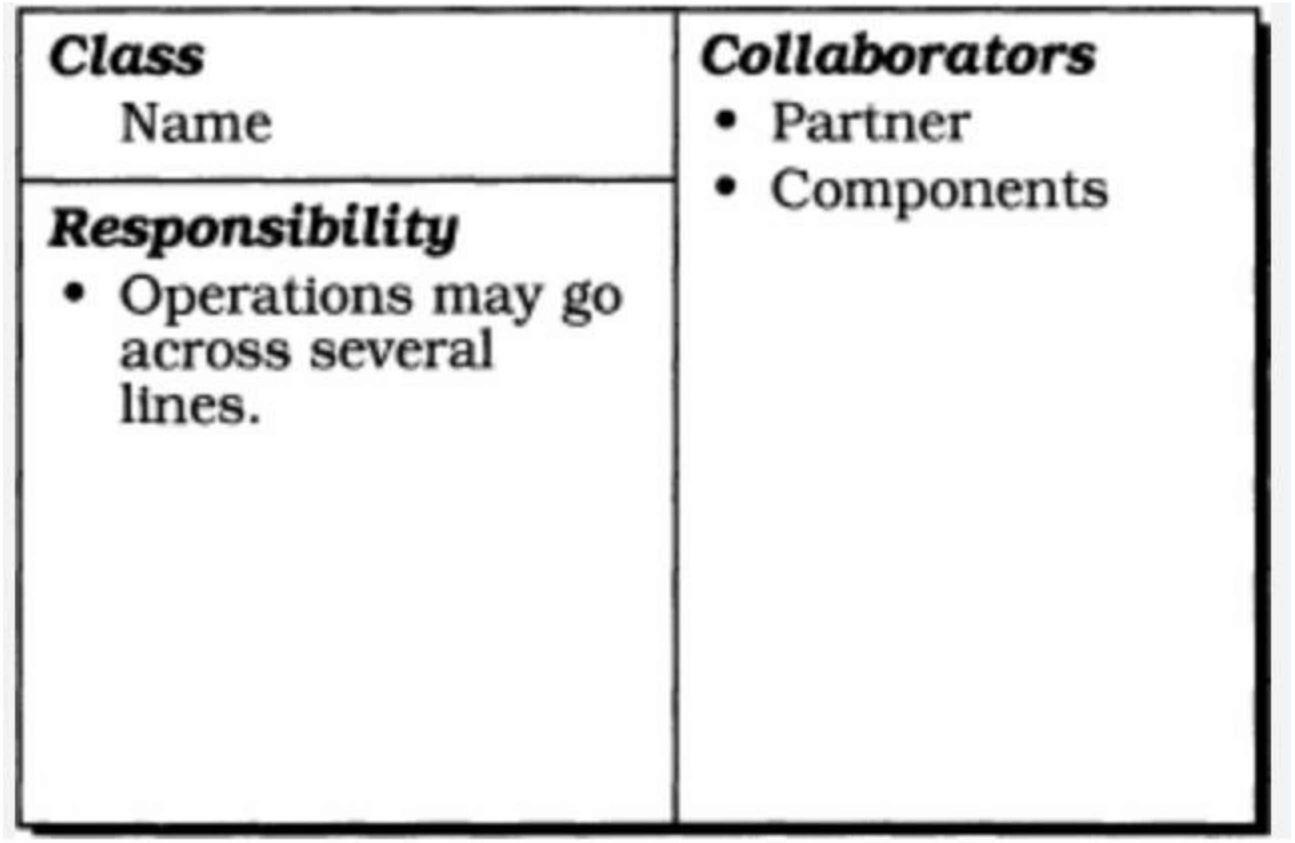
  - Static analysis object model: class & object diagrams

  - Dynamic model: sequence diagrams

Goal is to investigate the problem domain as far as possible before moving to the solution domain (i.e., design & implementation)

Requirements Analysis

- Class Responsible Collaboration



Class Name	**Collaborators**
Responsibility • Operations may go across several lines.	• Partner • Components

UML Class Diagram

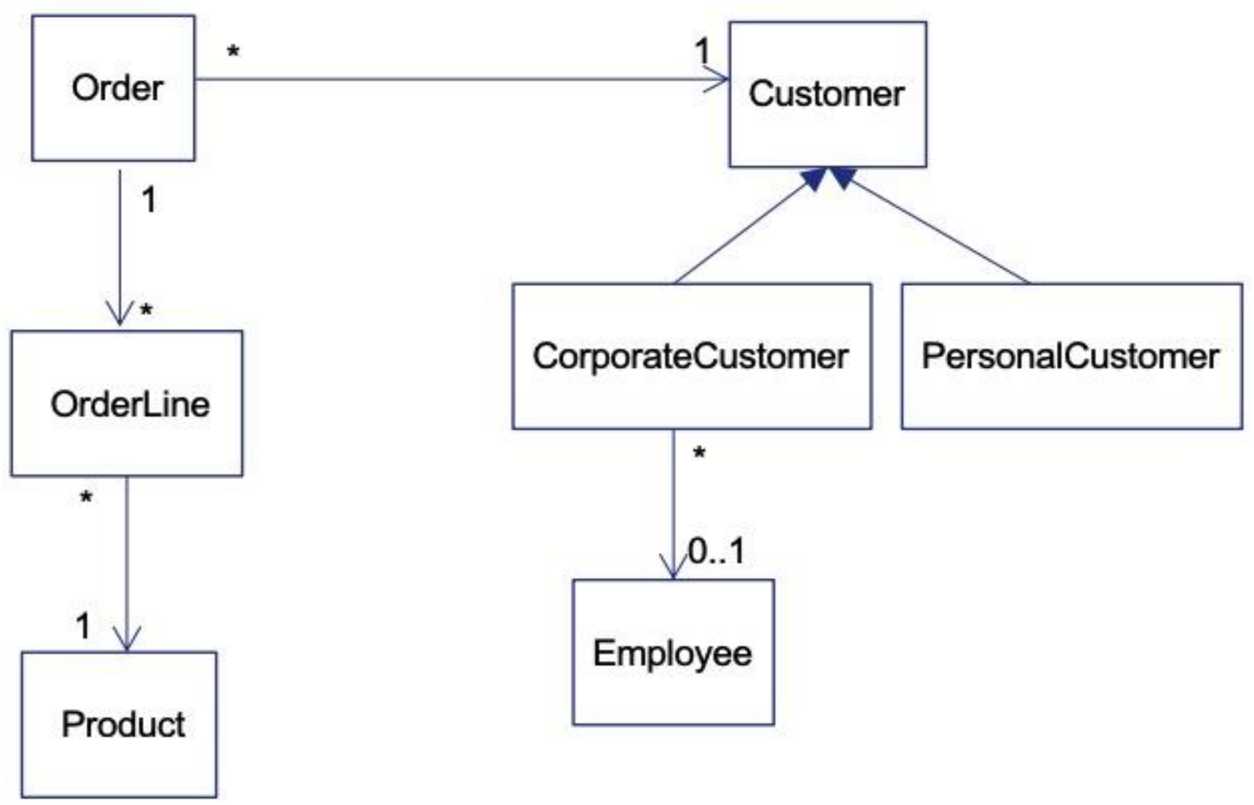
- **What is a UML class diagram?**

- A class diagram describes the **types of objects** in the system and the various kind of **static relationships** that exist among them.

- Class diagram also show the **properties and operations** of a class and the constraints that apply to the way objects are connected.

UML Class Diagram

- **Read this class diagram**



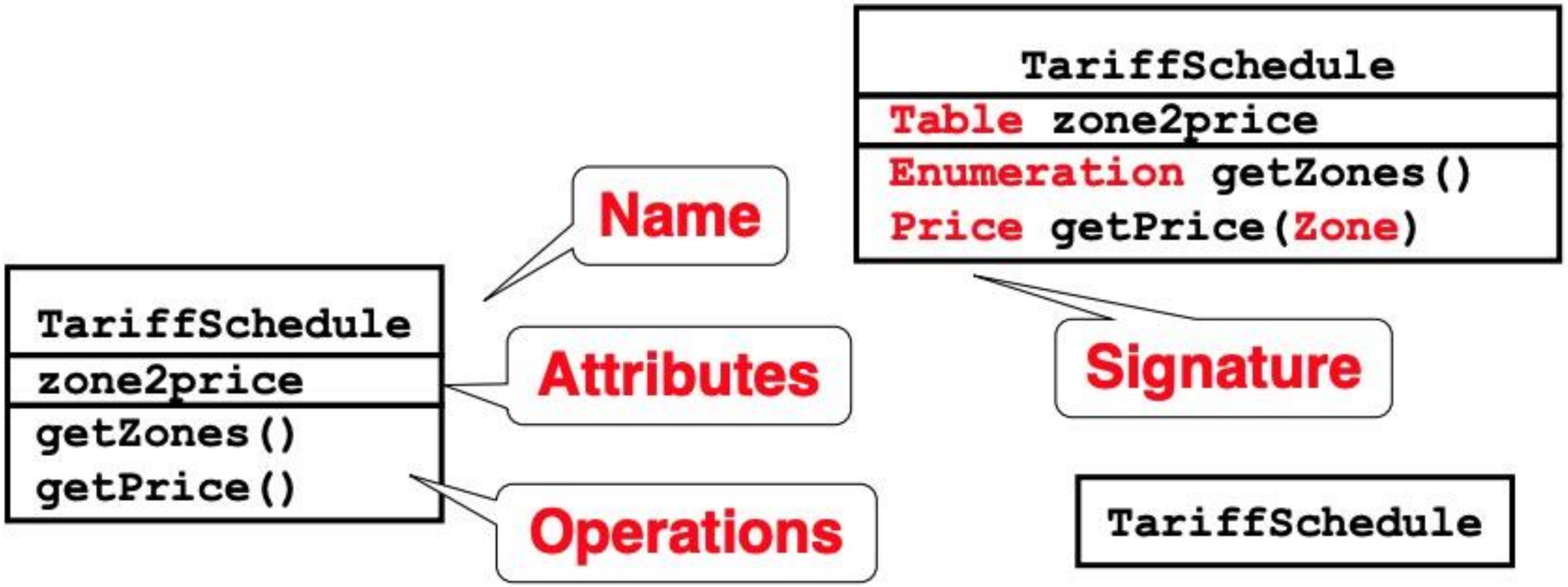
UML Class Diagram

- **Interpretation**:

- 1 customer has 0 or more orders, but each order has a single customer
- An order comprises one or more order lines
- Many order lines refer to a product

- Corporate customer and personal customer are special types of customer
- A corporate customer is supported by 0 or 1 sales reps who are order company employees – that is they may have a sale rep or may not

UML Class Diagram



- A class represent a concept (**template**)
- A class encapsulates state (**attributes**) and behavior (operations)
- Each attribute has a ***type***
- Each operation has a ***signature***
- The class name is the ***only mandatory*** information

UML Class Diagram

- **Multiplicity**

- The multiplicity of a property is an indication of how many objects may fill the property.

- The most common multiplicity you will see are

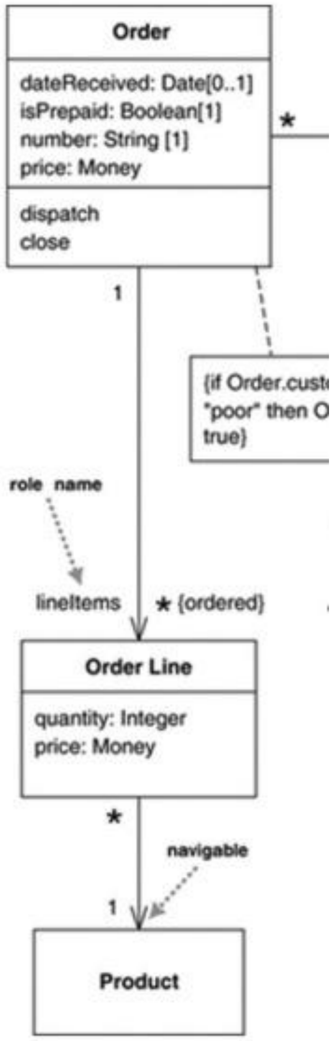
1	(An order must have exactly one customer)
0..1	(A corporate customer may or may not have a single sales rep.)
*****	(A customer need not place an order; there is no upper limit to the number of orders. That is, Zero or more orders)

UML Class Diagram

- As with anything else in the UML, there is ***no one way*** to interpret properties in code.
- So, the Order Line class would correspond to something like the following in Java:

```
public class OrderLine...
private int quantity;
private Money price;
private Order order;
private Product product
```

If an attribute is multivalued, this implies that the data concerned is a collection (ArryList ect in Java)



UML Class Diagram - Summary

- Our focus: class diagrams are the backbone of the UML, you will find yourself using them.
- The trouble with class diagrams is that they are so rich, they can be overwhelming to use.
- Don't try to use all the notation available to you.
- Start with the simple stuff, classes, associations, attributes, generation... Introduce other notations only when you need them.

Articles

- Unhelkar, Bhuvan. *Software engineering with uml*. Auerbach Publications, 2017.
- Ohst, Dirk, Michael Welle, and Udo Kelter. "Differences between versions of UML diagrams." Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering. 2003.