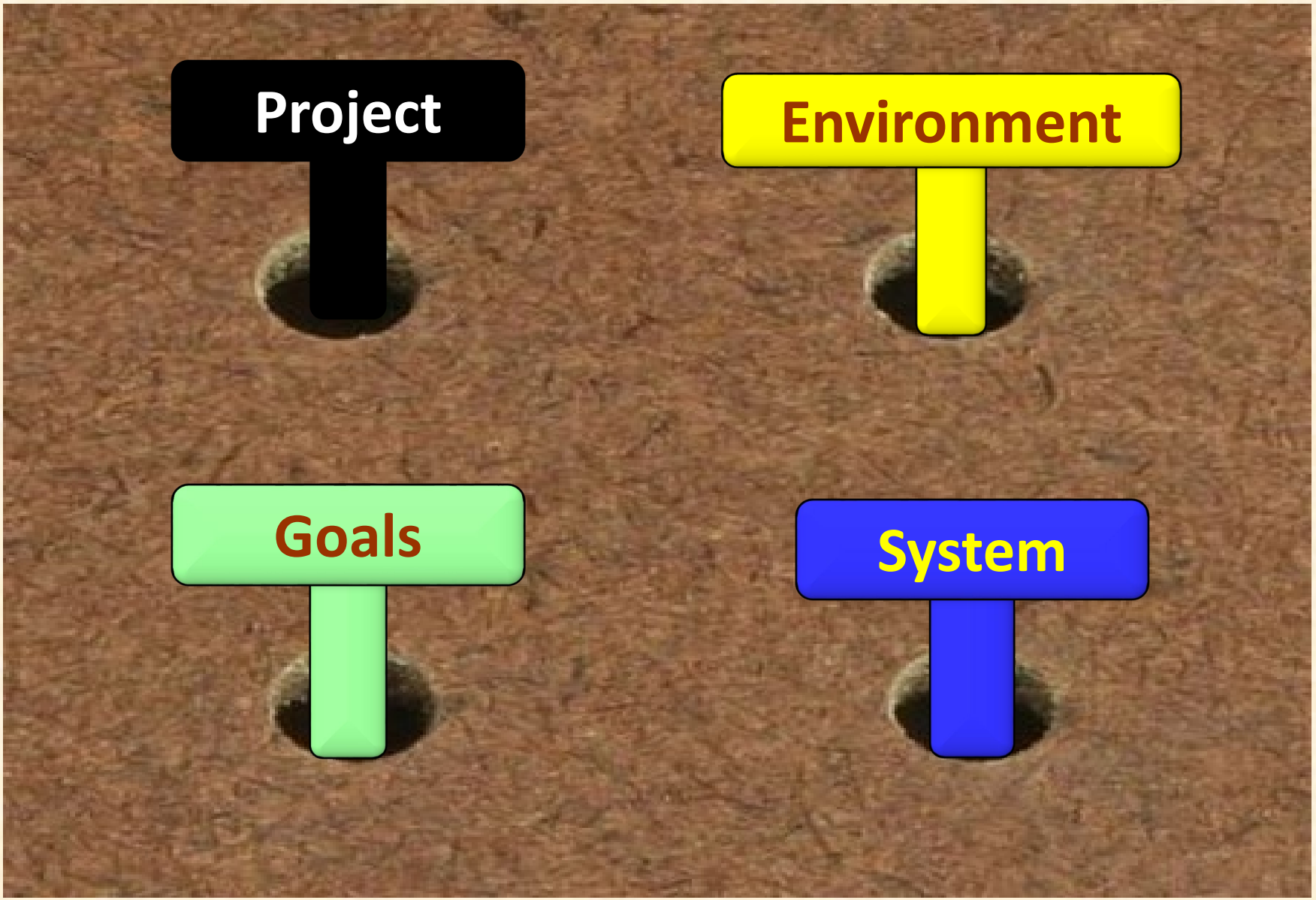
The four PEGS of requirements engineering

Bertrand Meyer

Schaffhausen Institute of Technology
& Eiffel Software

ACM Tech Talk, 4 March 2021

In a nutshell (1): four PEGS



In a nutshell (2): Four books of requirements

Project Book P

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

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

E.1 Glossary

E.2 Components

E.3 Constraints

E.4 Assumptions

E.5 Effects

E.5 Invariants

System book S

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

What's in this work

A comprehensive approach to requirements engineering, in line with modern views of software development

Four parts:

- 1. A precise definition of requirements concepts
- 2. Some requirements principles
- 3. A standard plan for requirements documents
- 4. Sketch of an effective software lifecycle model

Forthcoming book

Handbook of requirements and business analysis

Bertrand Meyer

Version: 2020-10-5
Draft, do not circulate
Change log is on next page

Acknowledgments

Jean-Michel Bruel (U. of Toulouse)

Sophie Ebersold (U. of Toulouse)

Florian Galinier (U. of Toulouse)

Manuel Mazzara (Innopolis University)

Alexander Naumchev (Innopolis University)

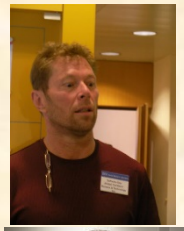
Maria Naumcheva (Innopolis University)

Some influences

Joanne Atlee

Daniel Berry

Alistair Cockburn



Anthony Finkelstein

Carlo Ghezzi

Martin Glinz

Daniel Jackson

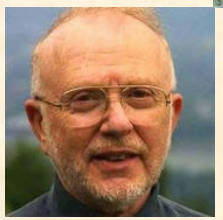
Michael Jackson

Ivar Jacobson

Jeff Kramer

Bashar Nuseibeh

David Parnas



Axel Van Lamsweerde

Karl Wiegers

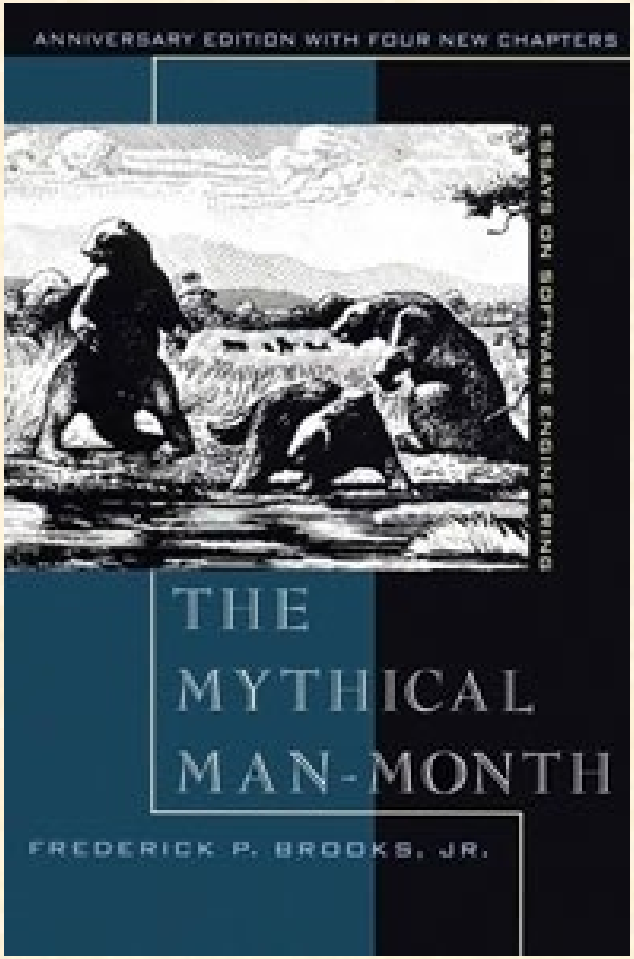
Pamela Zave

- 1 -

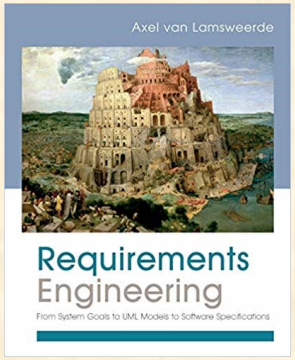
Key requirements concepts

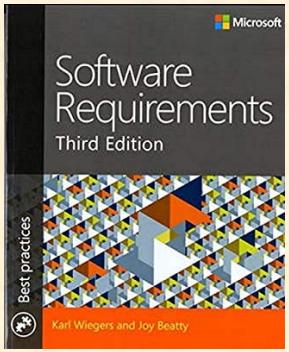
Requirements: Brooks (1975)

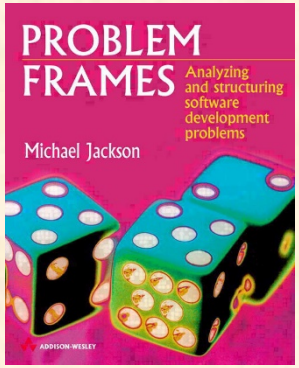
The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.

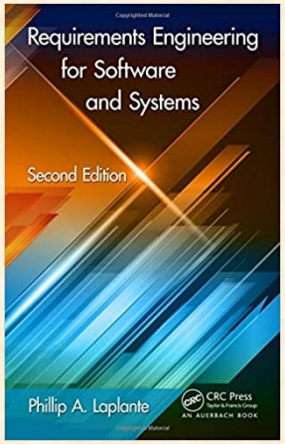


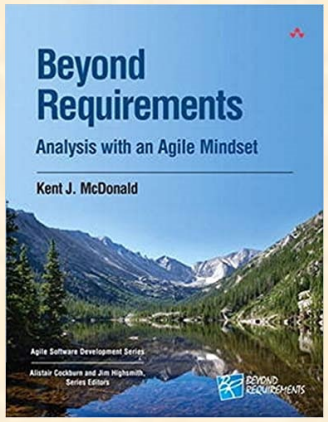
Chasm: theory vs practice

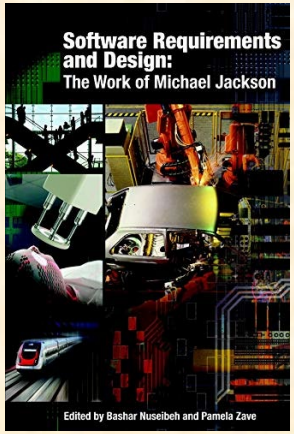




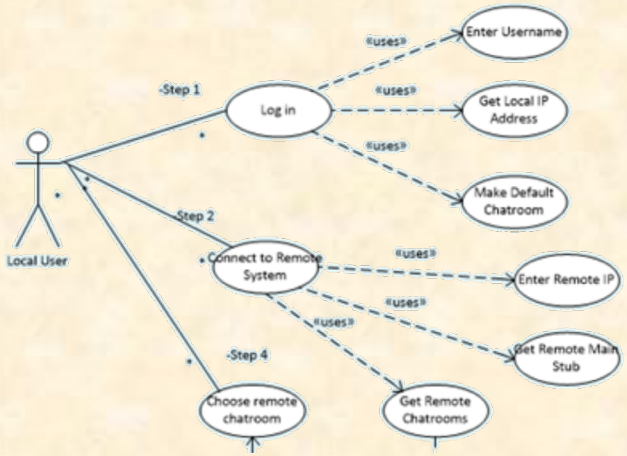








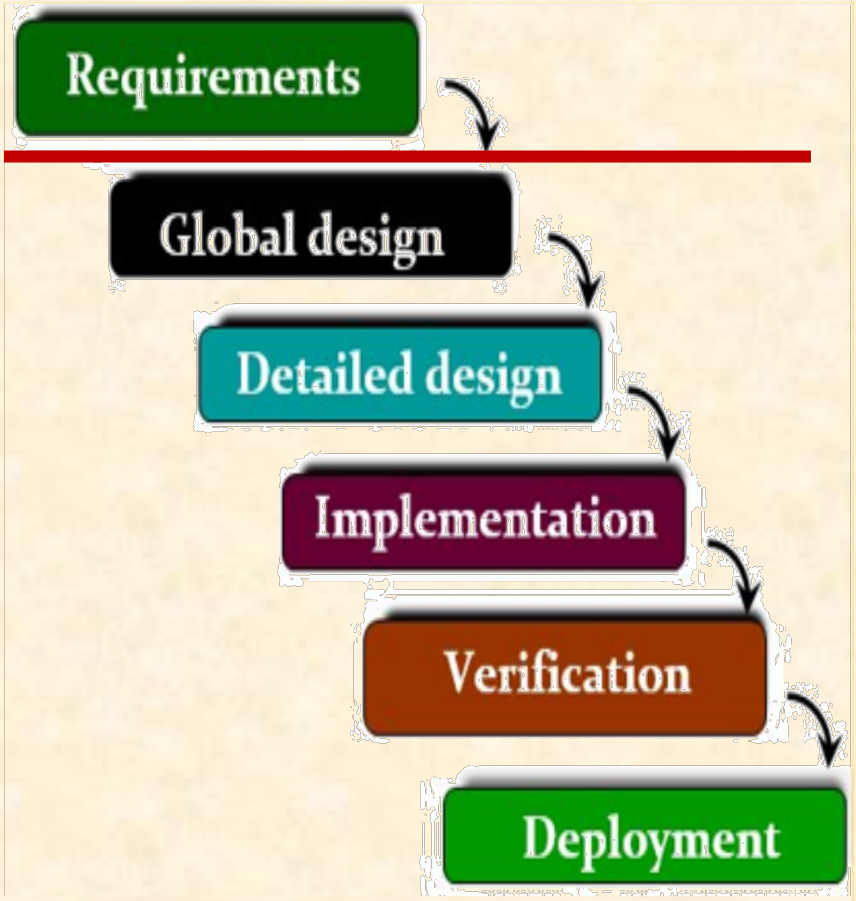
Use cases



User stories

As a *system administrator*, I want to *hide users' requests* so that I can *enjoy my holidays*

Chasm: traditional vs agile



Agile rejection of “big upfront anything”

And those things called requirements? They are really candidate solutions; separating requirements from implementation is just another form of handover

Mary Poppendieck, in “Lean Software”

Chasm: geek vs non-geek

Requirements engineers/
Business analysts

Subject-Matter Experts

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

More standards: “definitions”

(IEEE) ISO/IEC Standard for Application and Management of the Systems Engineering Process, 2005

Requirement

> *A statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).*

Confuses the prescriptive and the descriptive

IEEE Requirements Engineering standard (2018)

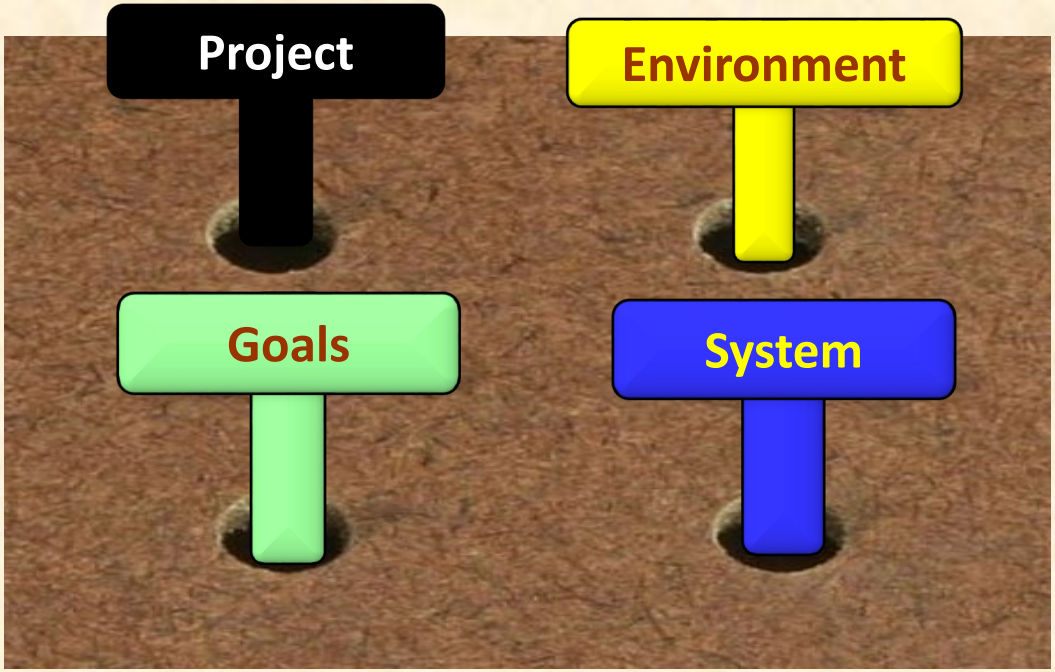
Requirement elicitation

> *Use of systematic techniques, such as prototyping and structured surveys, to proactively identify and document* ***customer*** *and* ***end user*** *needs*

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**



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

Reference concepts

Property: (boolean) predicate (on project, environment, goal or system)

All humans are mortal

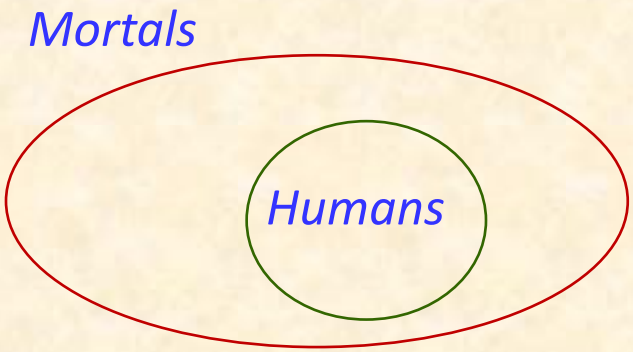
Relevant property: of interest to a stakeholder

Statement: human-readable expression of a property

Tout les hommes sont mortels

Alle Menschen sind sterblich

Все люди смертны



∀ x: HUMAN ¦ xìis_mortal

Requirement

A statement of a relevant property

of the

Project, Environment, Goals or System

Compare...

IEEE Systems Engineering standard

Requirement

A statement that identifies a product or process operational, functional, or design characteristic or constraint, which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).

IEEE Requirements Engineering standard

Requirement elicitation

*Use of systematic techniques, such as prototyping and structured surveys, to proactively identify and document **customer** and **end user** needs*

Kinds of requirement

Meyer, Bruel, Naumchev, Ebersold, Galinier,
"*A Taxonomy of Requirements*", TOOLS 2019




Goal: objective of the project or system, through effect on environment

Component: part that system, project or environment must include

Behavior: effects of system's operation

Task: project must include a certain an activity.

Product: material or virtual object used or produced by a task

Constraint: condition imposed by the environment on components, behaviors, tasks or products

Role: component carries some or all responsibility for behavior or task

Limit: project, system or environment does *not* include a requirement

More kinds of requirement element

Noise: property that is in the requirements but should not

Silence (lack): property that should be part of the requirements but isn't

Meta-requirement: property of requirements themselves (not the system, project or environment)

- 2 -

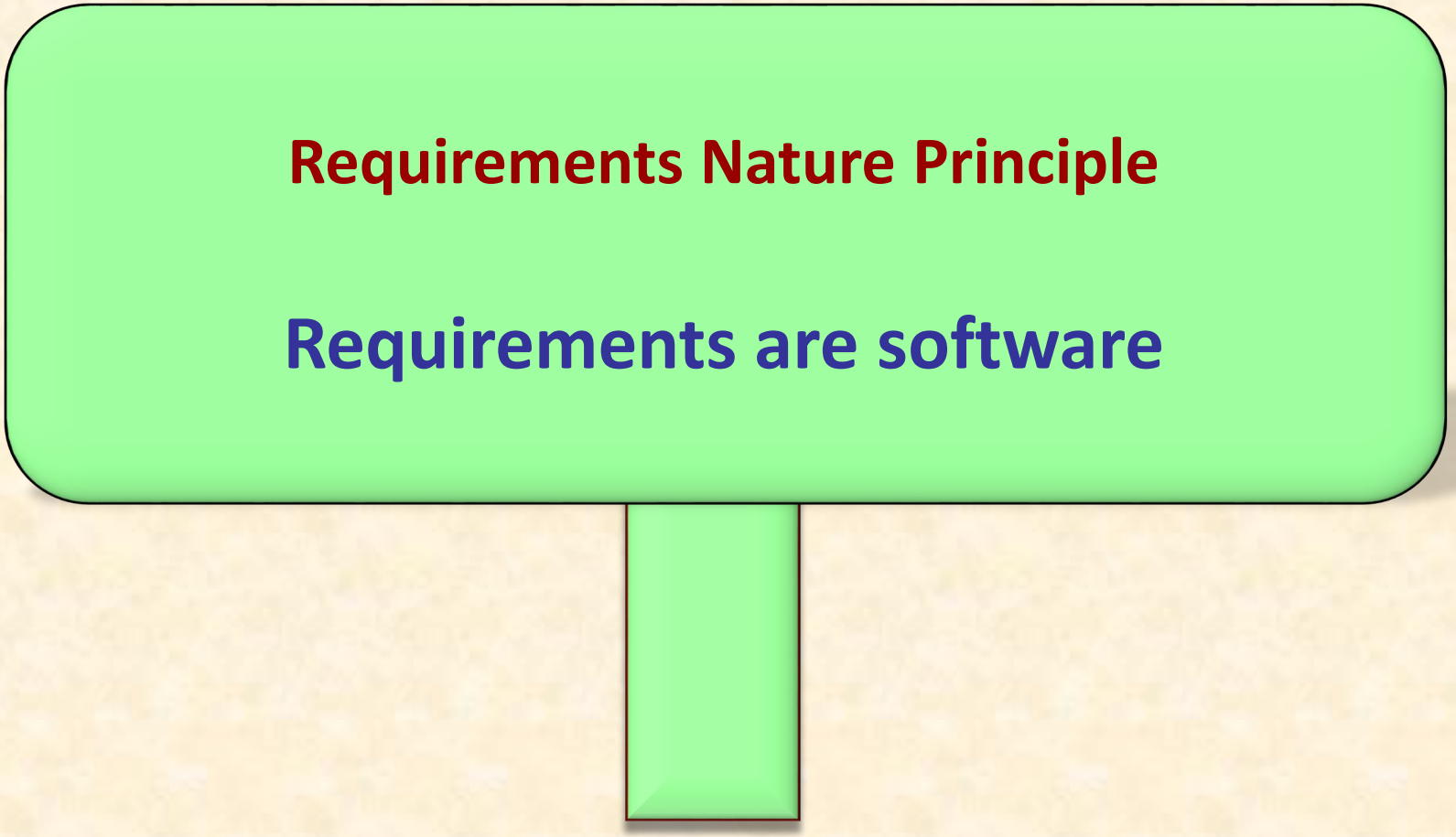
A few principles

Requirements quality: avoid “analysis paralysis”

Requirements Effort Principle

Devote enough effort to guarantee requirements quality, but not so much as to detract from other tasks of the software development process

The nature of requirements



The management of requirements

Requirements Management Principle

Make requirements and all elements that provide requirements-relevant information available in a repository, continuously maintained

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

Requirements change

Requirements Evolution Principle

Requirements are a living asset of any project, subject to evolution. They must be adapted & maintained throughout the project

Requirements in the lifecycle

Requirements Construction Principle

- Produce an initial version of requirements at start of project
- Update and extend them throughout project

- 3 -

The PEGS plan

Four books of requirements

Project Book

P

- 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

- G.1 Overall 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

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

System book

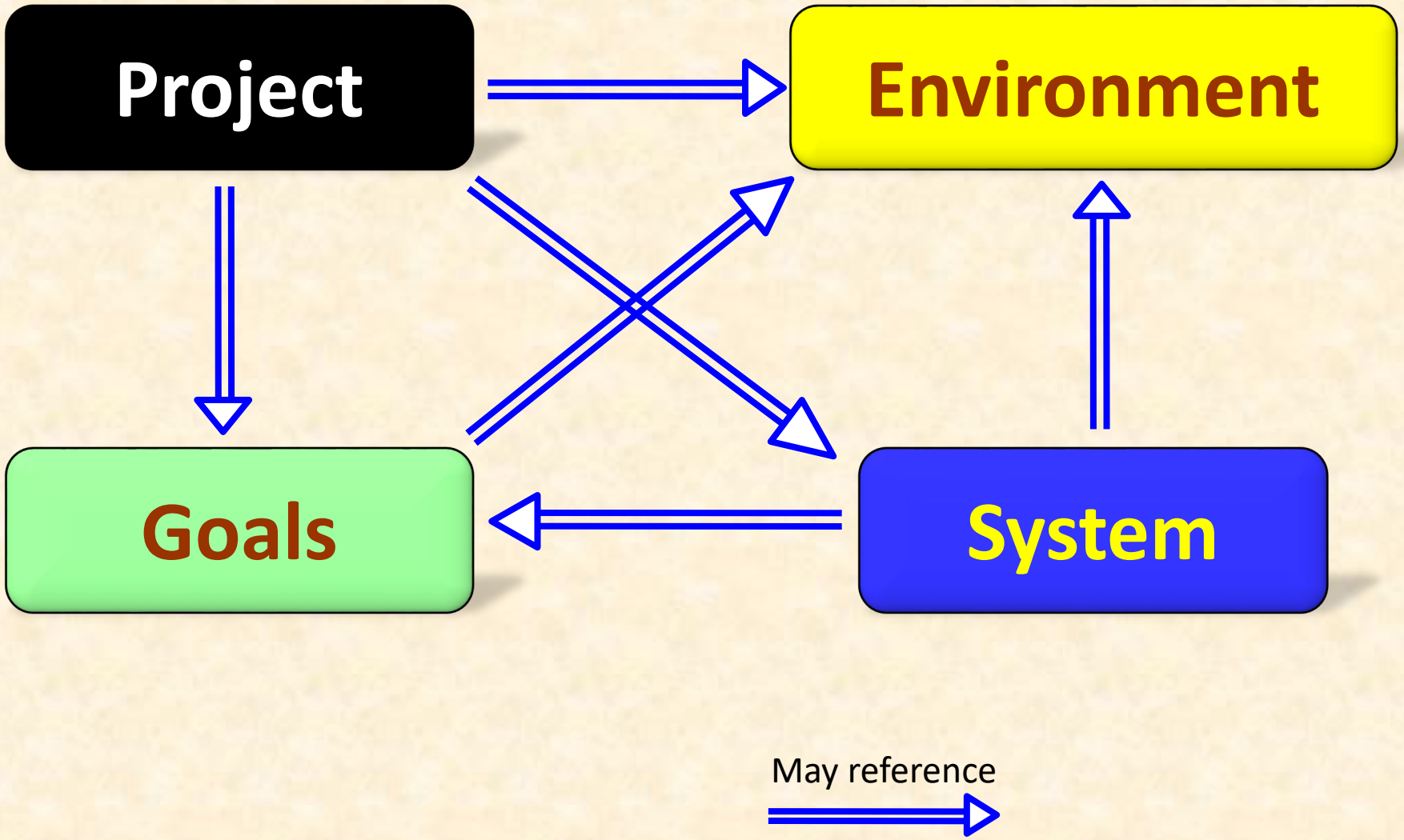
S

- 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

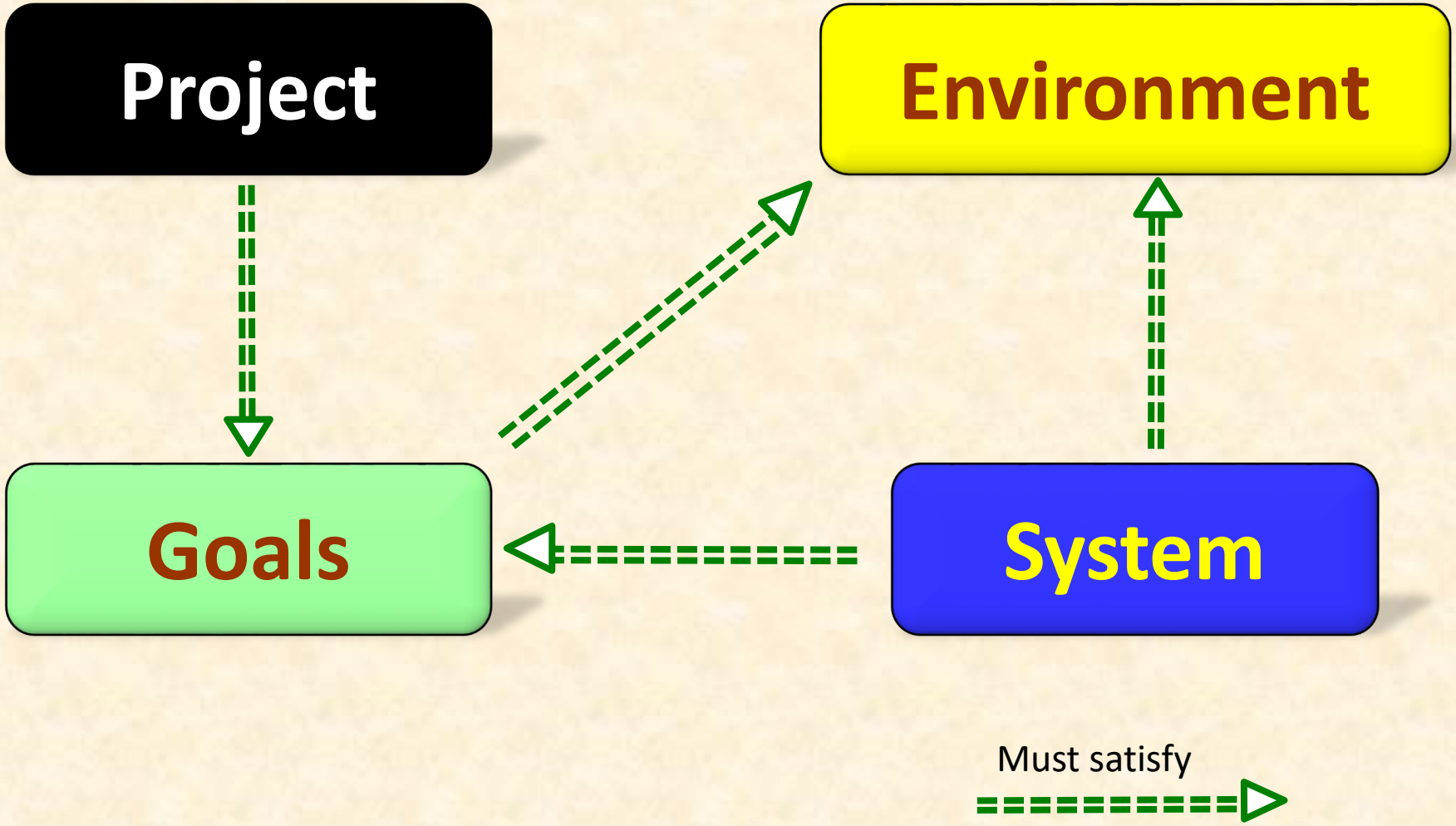
Notes on the plan

- 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

References between the four PEGS



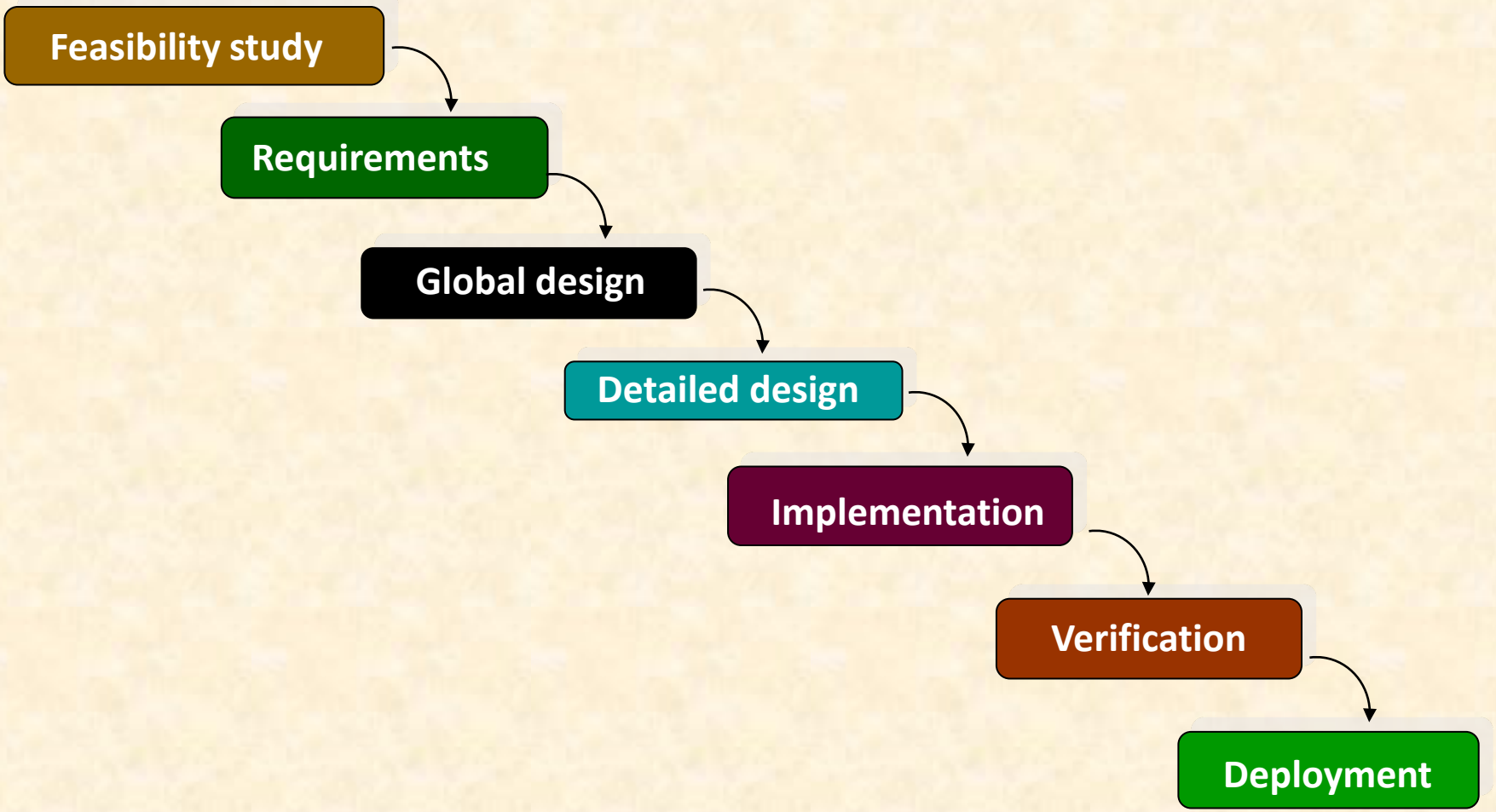
Verification obligations between the four PEGS



- 4 -

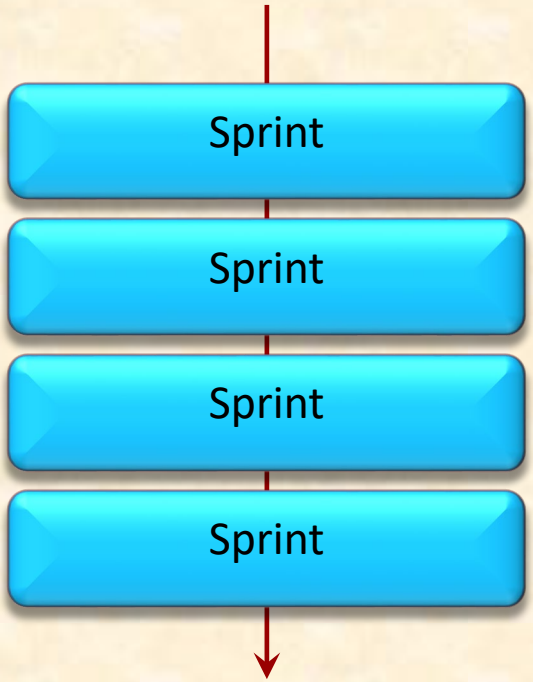
Requirements in the lifecycle

The waterfall view (*a pedagogical device*)



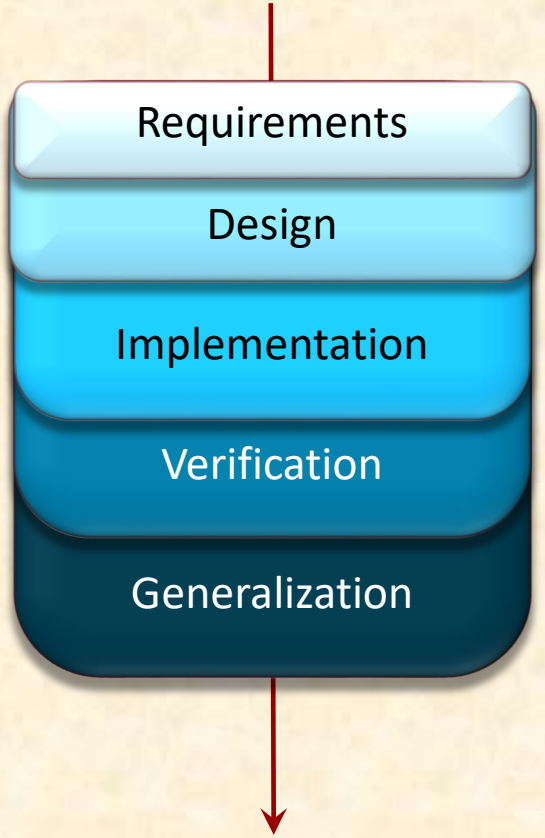
The agile model

- Iterations: sprints
- No upfront steps
- Each sprint includes requirements (user stories), test design, implementation, test run
- Every sprint produces executable system (prototype)



Seamless development

- Single concepts, principles, notation, tools
- Continuous, incremental development
- Keep model, implementation, documentation consistent



ACCOUNT, TRANSACTION…

STATE, COMMAND…

COUNTER…

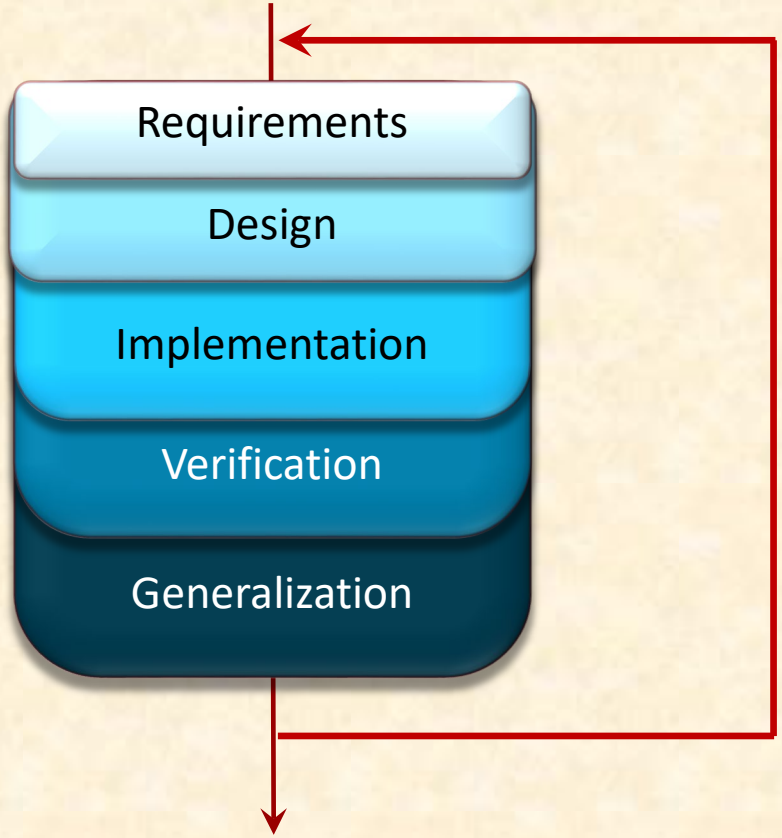
TEST_DRIVER…

TABLE…

Example classes

Seamless, reversible development

- Single concepts, principles, notation, tools
- Continuous, incremental development
- Keep model, implementation, documentation consistent
- Correct earlier steps as needed (reversibility)



Multirequirements

Multirequirements, 2013

Intertwined discourses:

- Natural language
- Graphics
- Formal (Eiffel)
- Tables

1. The basic idea (self-referentially)

1.1 Requirement, requirements, project

1.1.1 A software project /*PROJECT*/ exists to address some needs and must satisfy some constraints. The term "requirements" denotes the description of these needs and constraints. Any project, even one that most enthusiastically follows an agile, specify-as-you-go process, has requirements /*REQUIREMENTS*/; and conversely any "requirements" is relative to a project:

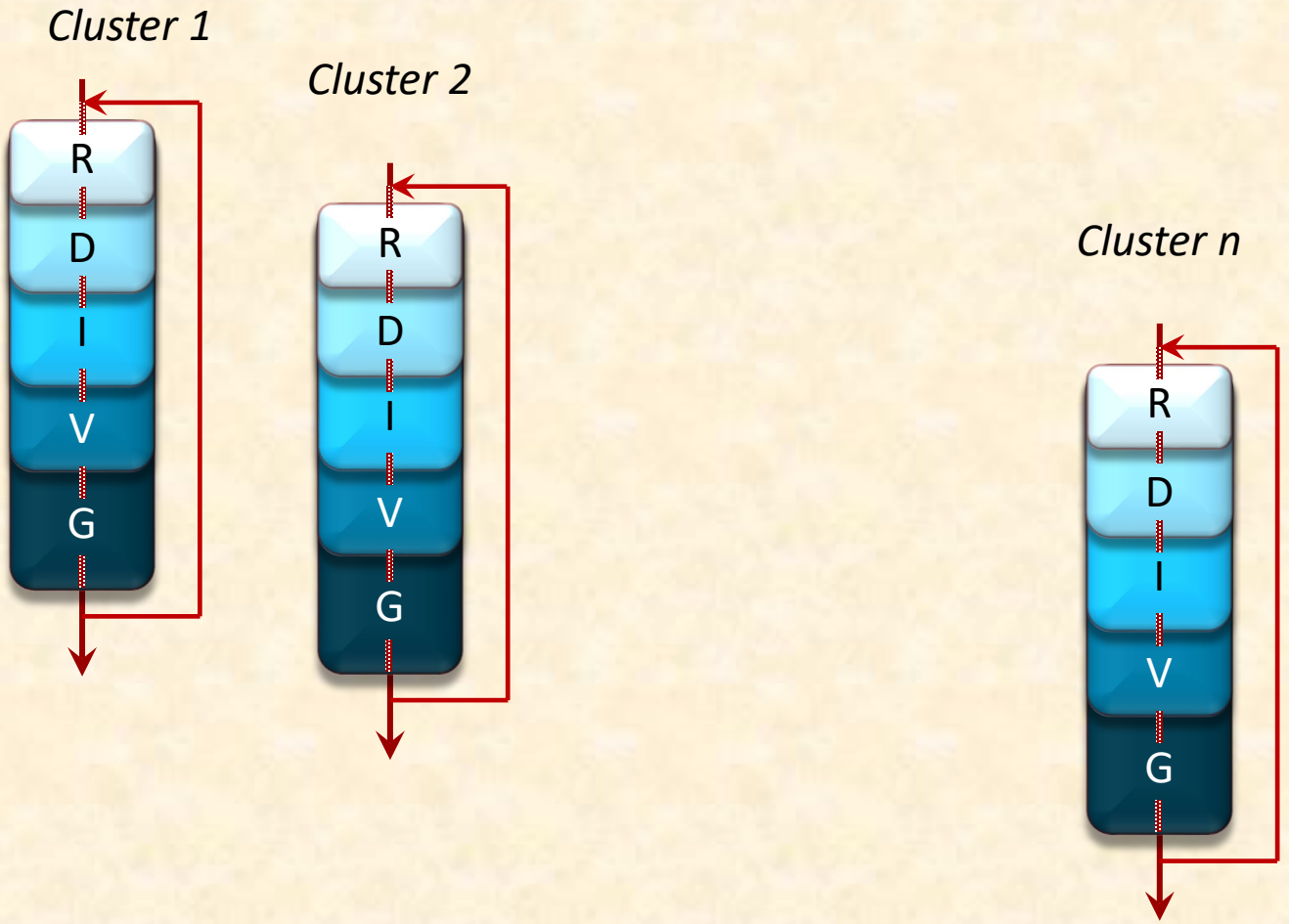
```
class PROJECT feature                                    -- E1.1.1
        requirements: REQUIREMENTS
invariant
        requirements.project = Current
end

class REQUIREMENTS feature
        project: PROJECT
invariant
        projects.requirements = Current
end
```

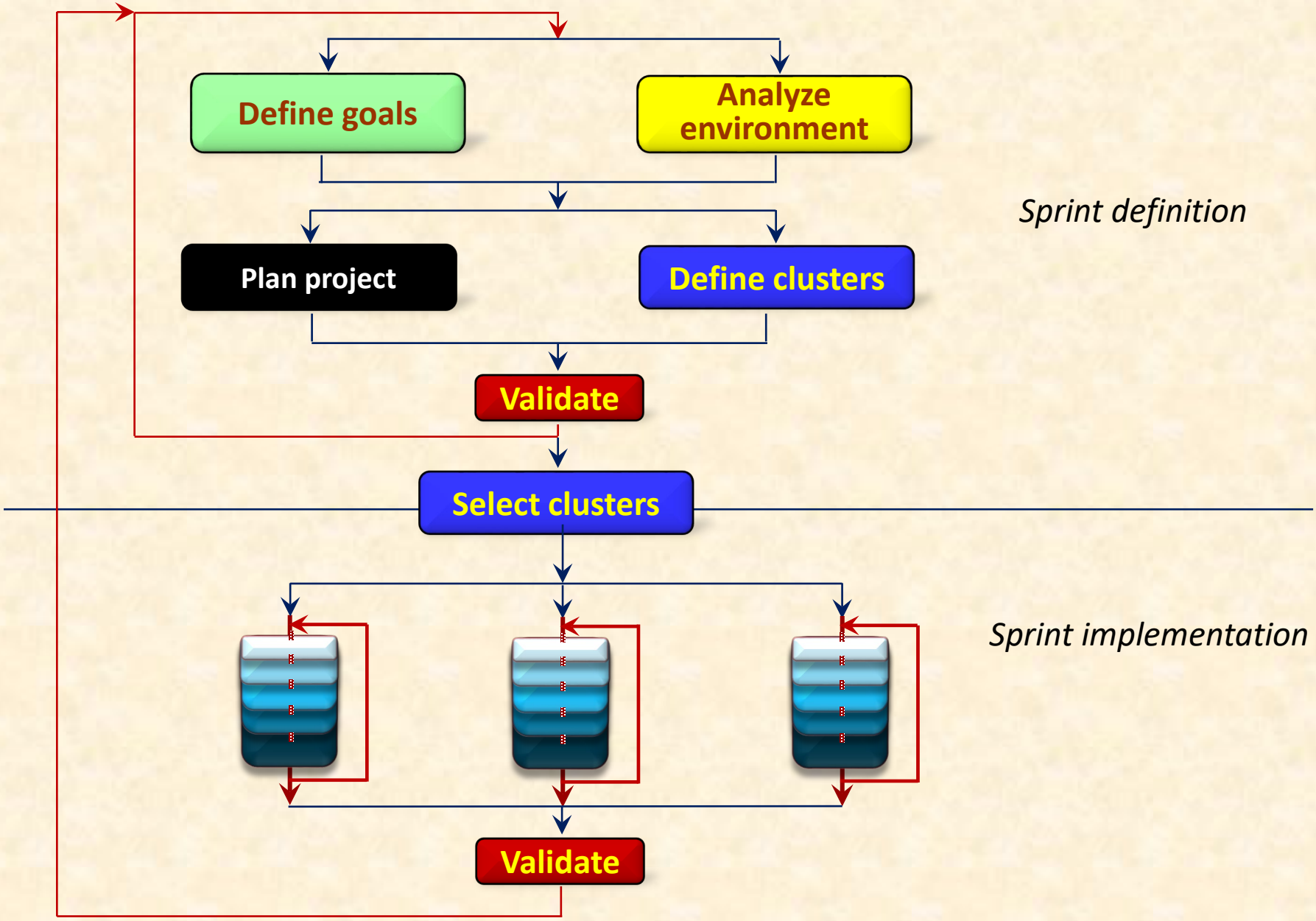
I1.1.1 (*Informative text*) It is convenient, the first time the requirements text introduces a term such as "project" describing an important abstraction that will be described by a class, to mention the class name in slashes, as in /*PROJECT*/. A tool should be available to collect all such occurrences automatically into an index.

1.1.2 We can enter the above information directly into our tools:

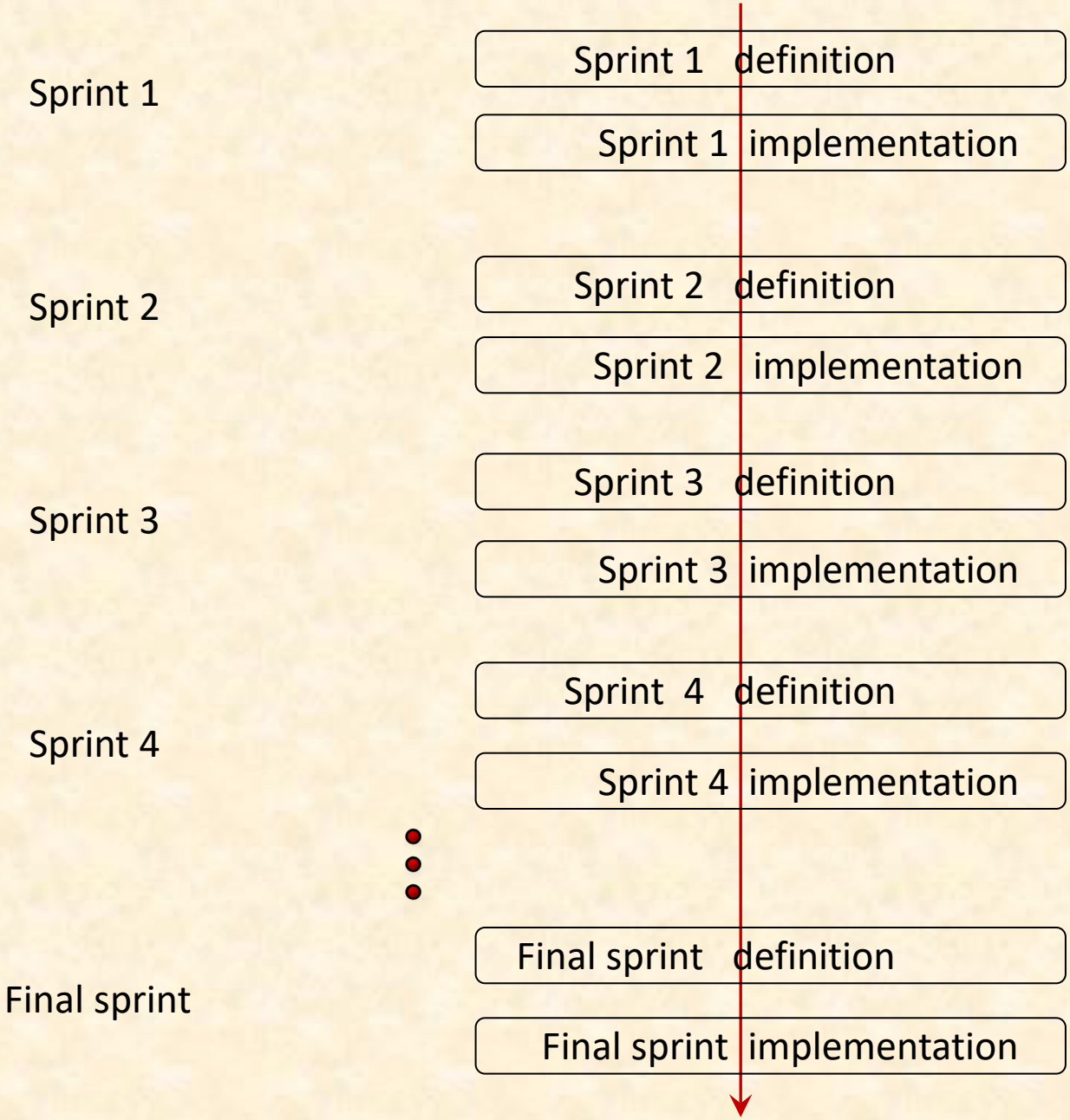
The cluster model



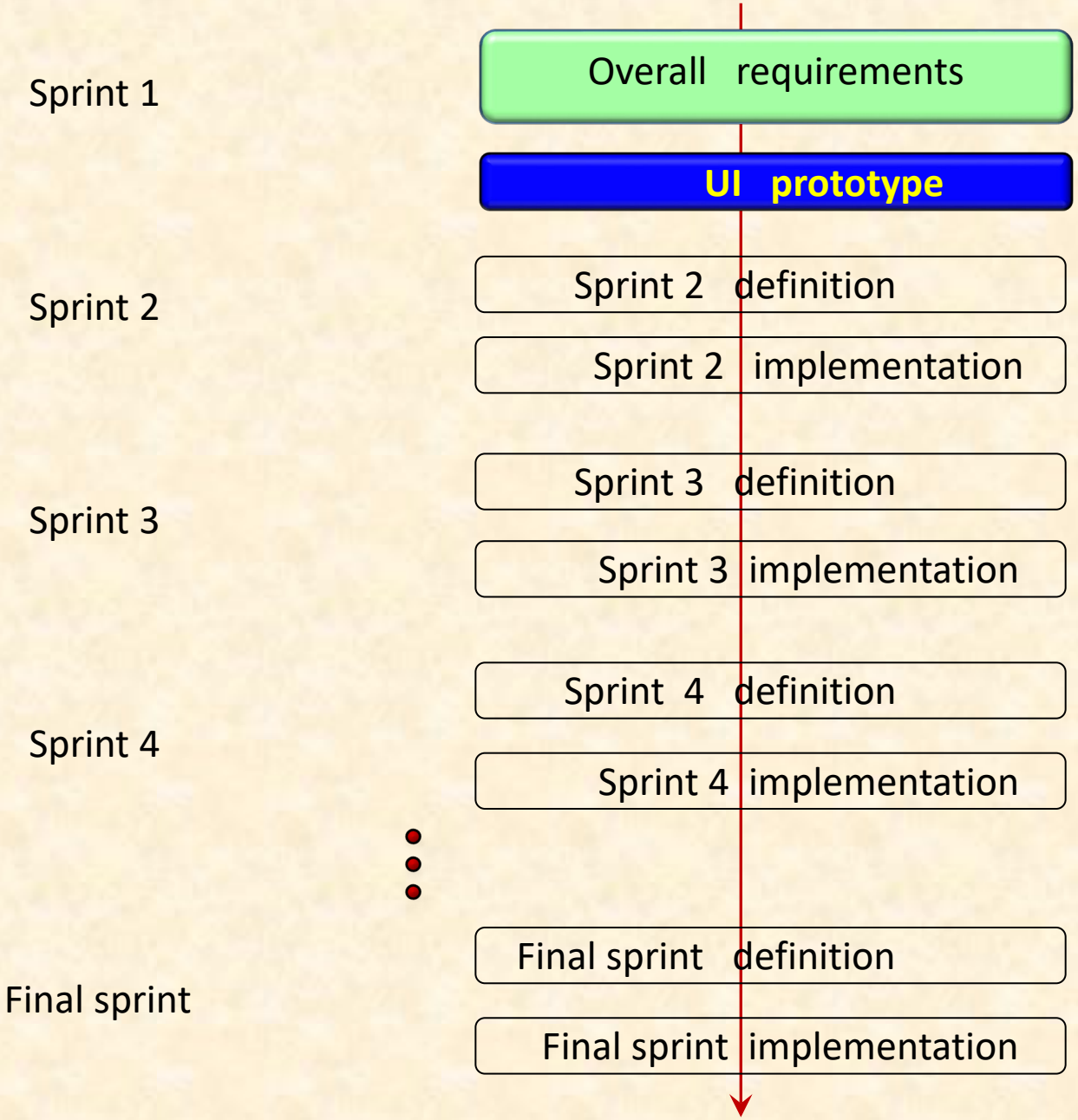
The PEGS lifecycle model



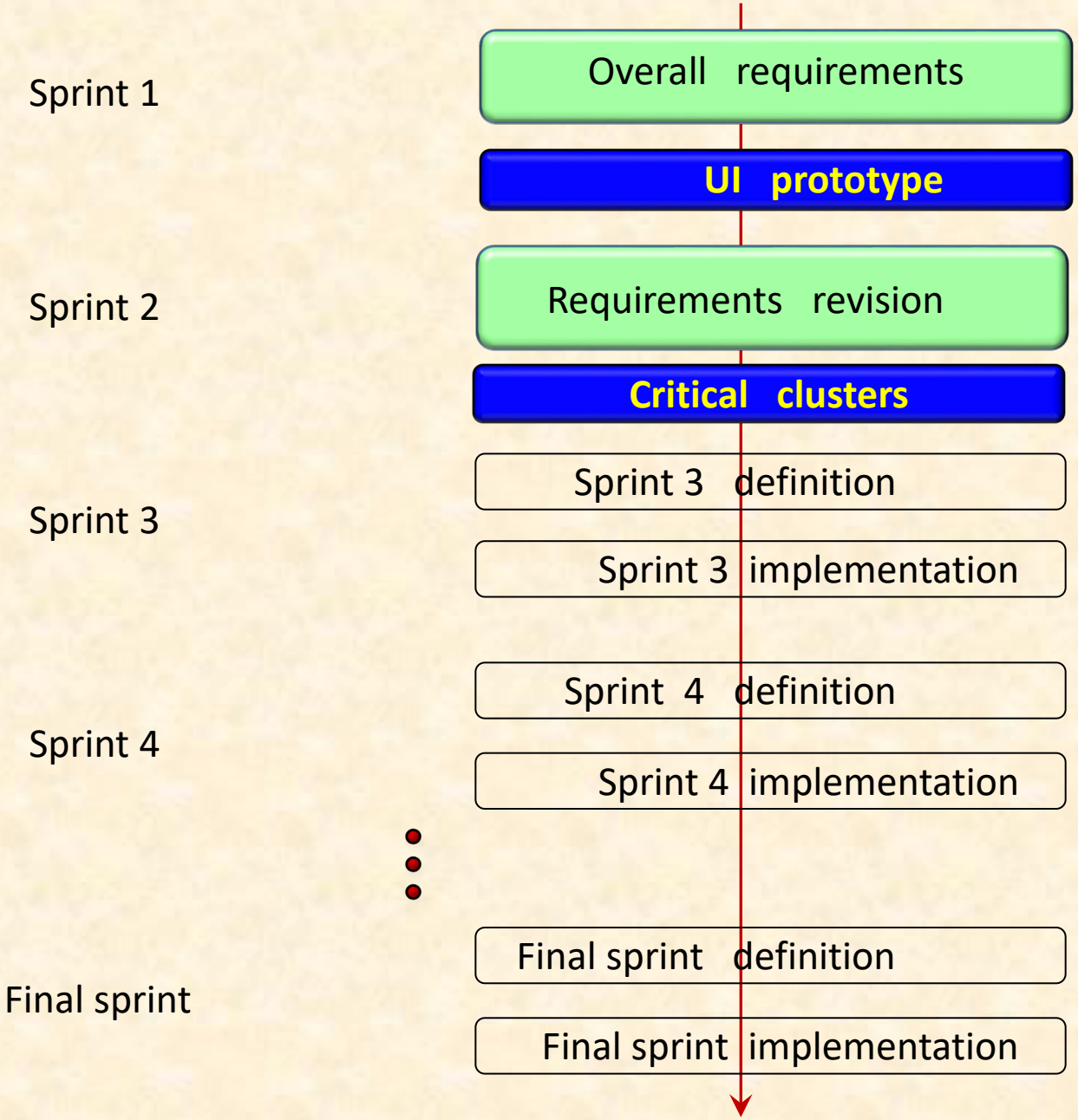
Over the project's timeline



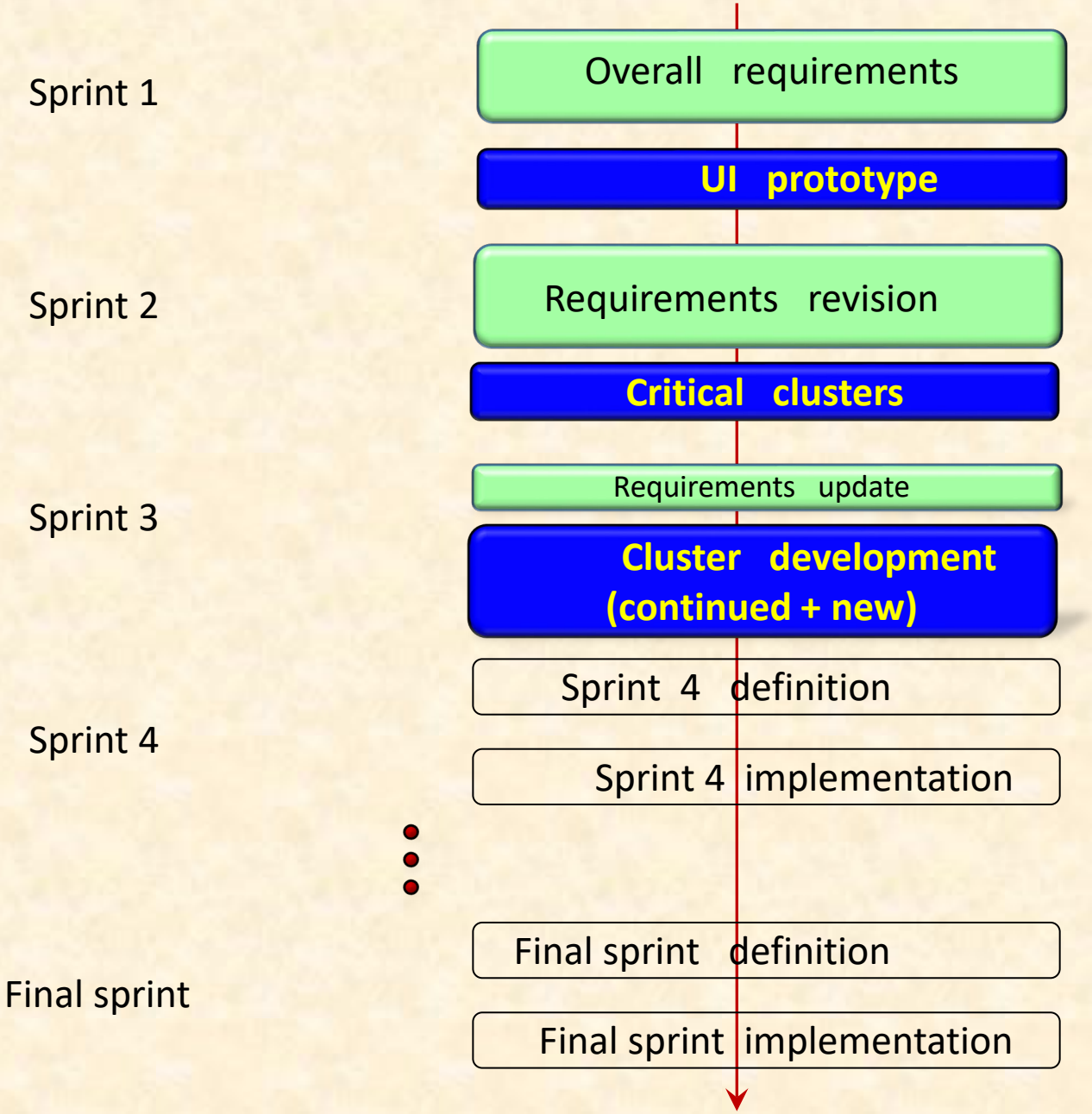
Over the project's timeline



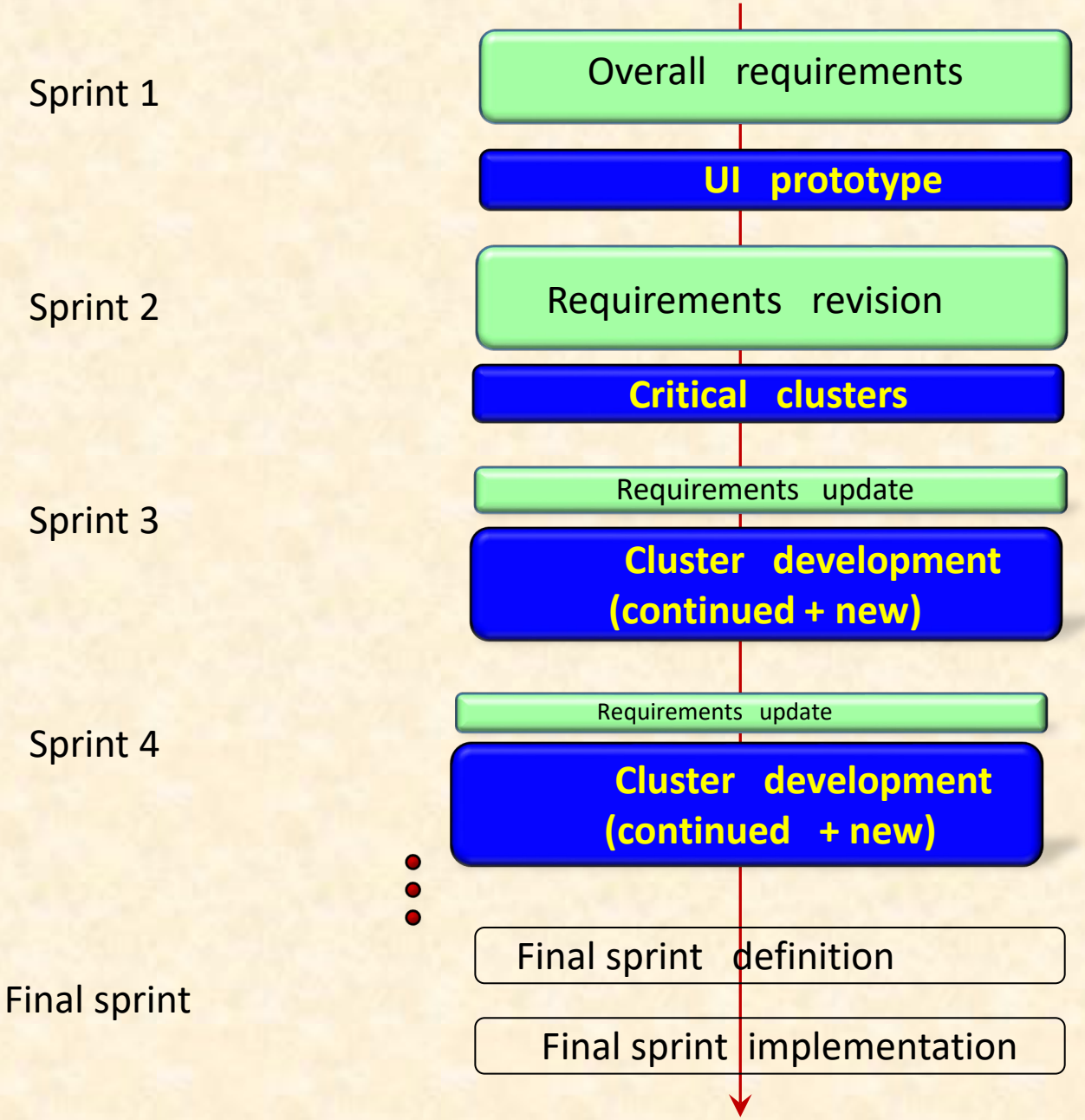
Over the project's timeline



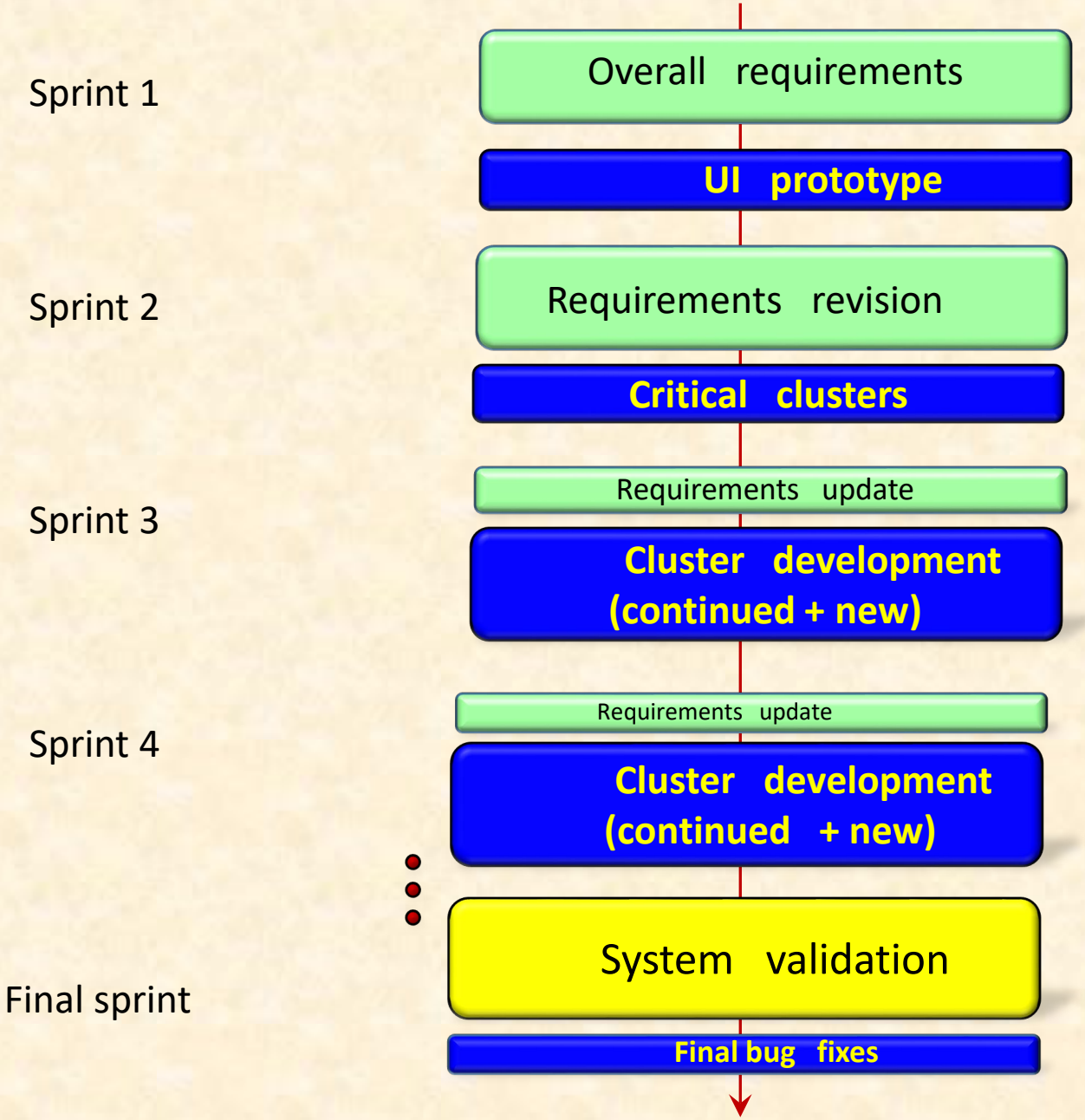
Over the project's timeline



Over the project's timeline



Over the project's timeline



In what notation do we state requirements?

For goals: mostly English

Everything else: Eiffel whenever possible, English otherwise (or other notations, e.g. graphics)

Connections/dependencies are recorded:

- Automatically for Eiffel texts
- Semi-automatically for Eiffel to something else
- Manually (in our RSS) otherwise

Object-oriented requirements

OO ideas are as productive for requirements as for the rest!

What does “object-oriented” mean?

- Modularize around object types
- Describe them as abstract data types (ADTs)
- Information hiding
  (Interface includes only subset of properties)
- Contracts
  (Interface is specified, from ADT)
- Inheritance
- Polymorphism and dynamic binding

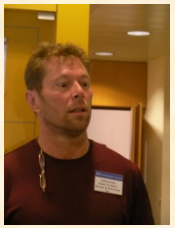
A place for everything

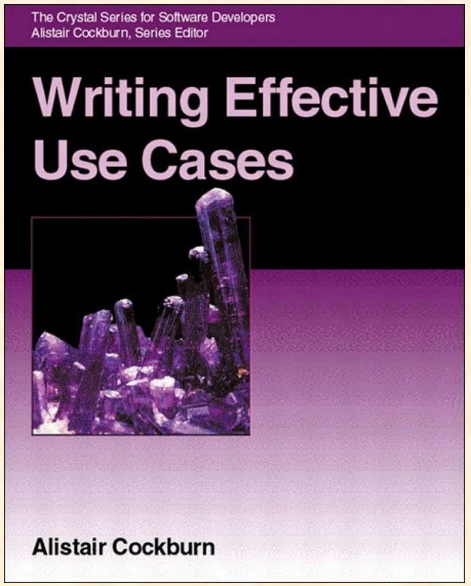
OO modeling applies to a wide variety of “objects”:

- Concrete object from environment — *VEHICLE*
- Abstract concept from modeling environment — *POLICY*
- Design abstraction — *COMMAND*
- Implementation object from system — *HASH_TABLE*
- Scenario of system usage — *CLAIM_PROCESSING*

“Specification drivers” (Naumchev) are objects that specify the behavior of other objects, overcoming limits of strict OO modeling

Use case: process insurance loss claim





Name	Process_loss_claim
Scope	Insurance company operations
Level	Business summary
Primary actor	Claims adjuster
Context of use	Claims adjuster handles claim
Preconditions	A loss has occurred
Trigger	A claim is reported to the insurance company

Main success scenario

- 1. A reporting party who is aware of the event registers a loss to Insurance company.
- 2. Clerk receives and assigns the claim to a claims agent.
- 3. The assigned claims adjuster:
- 3.1 Conducts an investigation.
- 3.2 Evaluates damages.
- 3.3 Sets reserves.
- 3.4 Negotiates the claim.
- 3.5 Resolves the claim and closes it.

Success guarantee Claim is resolved and closed.

(etc.)

A scenario as object

```
deferred class CLAIM_PROCESSING feature
        insurance: COMPANY
        affected: CUSTOMER

        process_complete_claim (c: CLAIM)
                require
                        cìall_documents_in_order
                deferred
                ensure
                        cìis_processed
                end
        process_incomplete_claim
                ...
invariant
        affectedìinsurer = insurance
end
```

References

On Formalism in Specifications (B. Meyer), in IEEE *Computer*, 1985

http://se.ethz.ch/~meyer/publications/computer/formalism.html

Multirequirements (B. Meyer), in *Modelling and Quality in Requirements Engineering*, eds. Seyff and Koziolek, MV Wissenschaft, 2013

http://se.ethz.ch/~meyer/publications/methodology/multirequirements.pdf

The Anatomy of Requirements (J-M Bruel, S. Ebersold, F. Galinier, A. Naumchev, B. Meyer), TOOLS 2019

https://arxiv.org/abs/1906.06614

The Formal Picnic Approach to Requirements (B. Meyer), blog article, 2020

https://cacm.acm.org/blogs/blog-cacm/232677-the-formal-picnic-approach-to-requirements/fulltext

The Role of Formalism in Software Requirements (J-M Bruel, S. Ebersold, F. Galinier, B. Meyer, A. Naumchev), accepted for publication in 2021 in ACM *Computing Surveys*

http://se.ethz.ch/~meyer/publications/requirements/formalism.pdf

Handbook of Requirements and Business Analysis (B. Meyer), to appear in 2021

Schaffhausen Institute of Technology

New university based in Switzerland https://sit.org

Areas of interest,
focus on interdisciplinarity:

- Computer science
- Advanced physics: new materials, quantum
- Digital business

Currently:

- Two research chairs in software engineering
- Master program in CS-SE, strong leadership focus

Next *SIT Insights* event (next Wednesday, March 10, 17 CET/11 AM EDT)

- Graduation of first students
- (At 17:30) talk by me on “The Beauty of Software”
- Free registration: https://sit.org

Lessons

Requirements are a key part of the software process

They have to be managed like software

They should not cause “analysis paralysis”

They cover four aspects of equal importance

The documents should reflect this 4-way structure

They should have a seamless relationship with other products

They determine a modern, flexible development model